SIMulation: Demystifying (Insecure) Cellular Network based One-Tap Authentication Services

Ziyi Zhou
Shanghai Jiao Tong University
Shanghai, China
jou.dzyi@sjtu.edu.cn

Xing Han
University of Electronic Science and Technology of China
Chengdu, China
x5tar@std.uestc.edu.cn

Zeyuan Chen
Shanghai Jiao Tong University
Shanghai, China
zechen@sjtu.edu.cn

Yuhong Nan
Sun Yat-sen University
Guangzhou, China
nanyh@mail.sysu.edu.cn

Juanru Li
Shanghai Jiao Tong University
Shanghai, China
jarod@sjtu.edu.cn

Dawu Gu
Shanghai Jiao Tong University
Shanghai, China
dwgu@sjtu.edu.cn

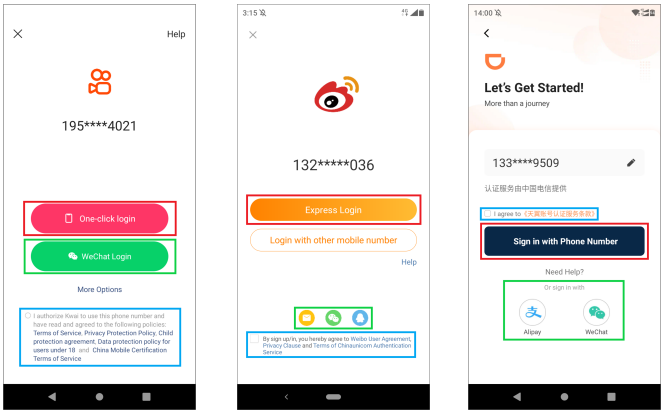
***Abstract*—A recently emerged cellular network based One-Tap Authentication (OTAuth) scheme allows app users to quickly sign up or log in to their accounts conveniently: Mobile Network Operator (MNO) provided tokens instead of user passwords are used as identity credentials. After conducting a first in-depth security analysis, however, we have revealed several fundamental design flaws among popular OTAuth services, which allow an adversary to easily (1) perform unauthorized login and register new accounts as the victim, (2) illegally obtain identities of victims, and (3) interfere OTAuth services of legitimate apps. To further evaluate the impact of our identified issues, we propose a pipeline that integrates both static and dynamic analysis. We examined 1,025/894 Android/iOS apps, each app holding more than 100 million installations. We confirmed 396/398 Android/iOS apps are affected. Our research systematically reveals the threats against OTAuth services. Finally, we provide suggestions on how to mitigate these threats accordingly.**

***Index Terms*—mobile security, mobile network operator, cellular network, malware, SIM card based authentication**

I. Introduction

Password-less authentication facilitates users especially those smartphone users disturbed by remembering and inputting different passwords for various mobile apps. Recently, a new type of authentication scheme, **cellular network based One-Tap Authentication** (OTAuth in short), has rapidly emerged. The OTAuth scheme allows smartphone users to log in an account with just one tap on the screen. In particular, when a user would like to log in an app with OTAuth enabled, she simply launches the app and let it automatically communicate with the Mobile Network Operator (MNO) that manages the cellular network and distributes the SIM card. During this process, the phone must use cellular network instead of a Wi-Fi network. Then the app prompts information as Figure 1 shows, and the user just needs to click the "Login" button (marked by the red box in Figure 1) to finish the authentication and log in to her account.

As a representative instance of *Mobile Connect* [1], a universal digital identity service proposed by the Global System for Mobile Communications Association (GSMA) [2], OTAuth is being adopted around the world [3]. Compared with traditional schemes (e.g., password based or SMS based authentication), OTAuth significantly simplifies the login process by reducing more than 15 screen touches and 20 seconds of operation [4], [5] each time.



(a) China Mobile OTAuth (b) China Unicom OTAuth (c) China Telecom OTAuth

Fig. 1: Examples of OTAuth interfaces in popular apps supported by different MNOs.

More importantly, the use of OTAuth liberates the users from creating and remembering a large number of login credentials for multiple apps. While the use of OTAuth eases the traditional login process, it also gives attackers more convenience to circumvent user authentication. Intuitively, the OTAuth process automates many steps of authentication (e.g., password input) that are originally executed by the user herself, and it is highly possible that malicious code could abuse this feature to execute unauthorized behaviors. Unfortunately, we seldom find a corresponding research against OTAuth despite many recent studies focusing on other login security issues in mobile apps such as vulnerable Auto-Login functions [6], weak Push-2FA authentication schemes [7], insecurely use of Android SMS One-Time Password APIs [8],

and flawed MNO SIM swap services [9]. Therefore, an in-depth security analysis of the underlying mechanisms in current OTAuth schemes are expected.

Our work. In this paper, we conducted the first comprehensive investigation against OTAuth schemes that are widely used by mobile apps. Particularly, as instances of this type of scheme, we focused on world's top 30 MNOs [10] and investigated three OTAuth schemes supported by MNOs in mainland China, namely China Mobile, China Unicom, and China Telecom. These three MNOs have a total subscription of more than 1.6 billion [10]. We recovered the design and implementation details of them by reverse engineering apps with OTAuth functionality and inspecting the corresponding SDKs.

We abstracted the OTAuth schemes as a three-phase process and analyzed its design flaws. We proposed a new attack, called SIMULATION attack, which exploits a fundamental design flaw of OTAuth we identified: *During the authentication process, the remote (both MNO and app) servers could not identify which app starts the authentication if the apps are on the same smartphone.* In SIMULATION attack, an adversary could easily exploit this design flaw and bypass the authentication scheme. Our attack only requires a local malicious app without any privileged permission, or a malicious device that shares the cellular network of the victim smartphone (e.g., by connecting the Wi-Fi hotspot established by the victim smartphone). A successful SIMULATION attack directly leads to unauthorized login of various app accounts of the victim user, and in many cases could be abused to query user identities, register new accounts without user consent, and interfere OTAuth services of legitimate apps.

To fully understand the impact of SIMULATION attack, we conducted a large-scale measurement against 1,025 Android apps and 894 iOS apps, all of which are highly popular in the mobile app market. As of July 2021, each app holds more than 100 million installations. Among these top popular mobile apps, we identified 396 Android apps and 398 iOS apps were affected by SIMULATION attack. 17 of these affected apps have more than 100 million Monthly Active Users (MAU) and 230 of them have more than 1 million MAU in September 2021[1].

In addition, our research identified a total number of 22 SDKs (including SDKs provided by MNOs, as well as third-party agents) that are affected by the SIMULATION attack. Combining the analysis results of SDKs and apps, we have also identified several additional implementation weaknesses that can bring security risks to the OTAuth scheme, such as insecure token usage, authorization without user consent, and plain-text storage of sensitive information (see Section IV-D). Lastly, we provide suggestions on how to mitigate such threats.

In a nutshell, this paper makes the following contributions:

- We uncovered several design and implementation flaws of a new mobile login scheme, OTAuth, which has a large usage and high popularity among real-world apps.
- Exploiting the flaws of OTAuth scheme, we designed several attacks in which an adversary can fully bypass the authentication and perform malicious actions to the target app.
- We performed a large-scale measurement to evaluate the impact of these threats. To achieve this, we proposed a pipeline that integrates both static and dynamic approaches for effectively detecting potential affected apps. Our results showed that a large portion of highly popular apps are vulnerable to the attacks (38.6% for Android and 44.5% for iOS, respectively).
- We discussed feasible modifications to enhance the security of the OTAuth scheme.

Ethical Considerations. The SIMULATION attack related experiments were conducted using phone numbers and app accounts of the authors, hence did not affect other users. We have informed three affected MNOs through CNCERT/CC[2] , the authority for vulnerability coordination in mainland China. In the meantime, we provided solutions to help them fix related issues. CNCERT/CC has verified our findings and related vulnerabilities were documented under CNVD-2022-04497, CNVD-2022-04499, and CNVD-2022-05690. All of them are rated as high severity (scoring 8.3 out of 10 in CVSS 2.0).

II. BACKGROUND

In this section, we introduce the background knowledge about One-Tap Authentication scheme, including its overview, the protocol design, as well as the ecosystem of its usage in mobile apps.

A. One-Tap Authentication (OTAuth) Scheme

One-Tap Authentication is essentially a third-party-based authentication scheme supported by Mobile Network Operators, similar to other login options such as Single-Sign-On [13]. For an app that integrates this service, the user can log in to her app account with the *local phone number* of the device with just one click. Here, the *local phone number* specifically refers to the phone number that is bound to the SIM card on this mobile device. More specifically, as one of the login options, the app displays the *masked local phone number* of the device (e.g., "195****4021" in Figure 1(a)). The user can opt to log in in this way, without typing any credentials (e.g., username and password), or manually choose to log in in other ways, such as by interacting with other apps marked by green boxes in Figure 1 (similar to performing Single-Sign-On with Google or Facebook). If the *local phone number* is not associated with any account, in most cases, the app will automatically create a new account and bind it to this *local phone number*. Compared with other authentication schemes on mobile platform, such as Single-Sign-On [13], or SMS One-Time-Password (OTP) [14], OTAuth provides a much better user experience since it requires significantly less user interaction. More importantly, OTAuth scheme allows app

[1] According to the statistical results of IiMedia Polaris [11].

[2] National Computer Network Emergency Response Technical Team/Coordination Center of China, the national CERT of China and responsible for handling severe cyber-security incidents [12].

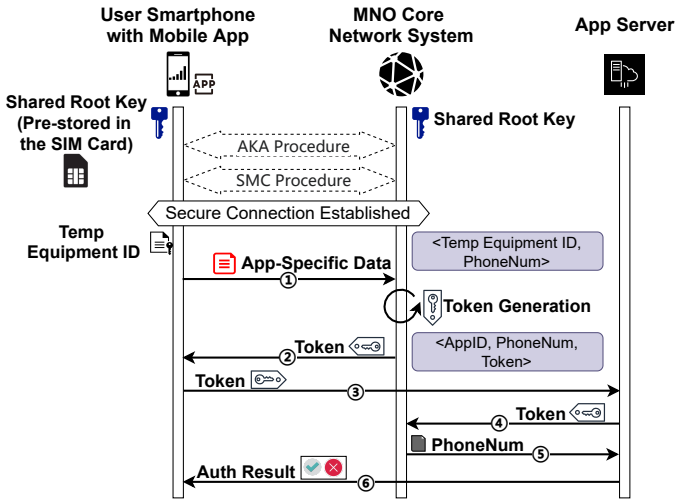


Fig. 2: Key design of the OTAuth Scheme.

developers to pay less fee for user authentication [4], [15], [16], which provides a strong motivation for developers to integrate this service.

Key design. The most unique part in this OTAuth scheme is that the *local phone number* is obtained neither through user input nor by requiring any system permissions (i.e., READ_PHONE_STATE or READ_PHONE_NUMBERS). Instead, the *local phone number* here is obtained based on the MNO's capability of recognizing phone number. The only requirement for this OTAuth scheme is (1) the app has introduced MNO's service, and (2) the smartphone has access to the cellular network.

Figure 2 presents the high-level design of OTAuth services. Before the OTAuth procedure actually starts, the user's smartphone needs to interact with the MNO Core Network System to perform the Authentication and Key Agreement procedure (AKA procedure) and the Security Mode Control procedure (SMC procedure). The instances of such procedures may vary in different networks [17]–[19]. After this, the user's smartphone and the MNO Core Network System have established a secure connection based on a shared root key [20].

The OTAuth procedure begins right after the secure connection is established. First, the app on user's smartphone sends app-specific data to the MNO server through cellular network. Since MNO has the capability of recognizing phone number, the MNO's server can generate a token that is associated with this phone number, and transfer it back to the user's smartphone. Then, to perform authentication, the user's smartphone needs to send this token to the app server. The app server will forward this token to the MNO server in order to get the phone number related with this token. In this way, the app server can know the phone number of user's smartphone, and decide whether to allow its login or sign-up request.

B. OTAuth Scheme Details

Figure 3 shows the protocol flow of the OTAuth scheme step-by-step. The whole process can be divided into three phases:

(1) **Initialize.** In this phase, the app first detects whether the runtime environment supports OTAuth. If this statement is true, it then tries to obtain the masked local phone number (not the full local phone number), in order to display it on the user interface (see Figure 1).

Specifically, the user starts the OTAuth flow by tapping on the login (or sign-up) button (step 1.1), which actually sends an OTAuth request to the app. After receiving the user's request, the app calls a specific API of the MNO SDK (e.g, the API *loginAuth* in the SDK of China Mobile), together with *appId* and *appKey* as the parameters (step 1.2). Here, both *appId* and *appKey* are exclusive to a specific app, which is pre-assigned to

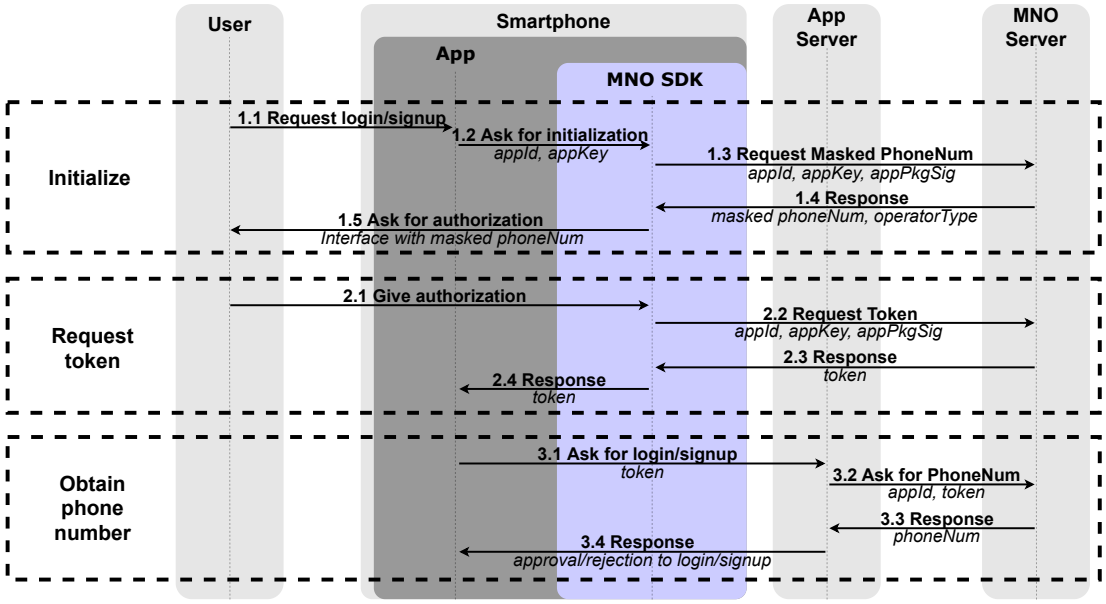


Fig. 3: The protocol flow of OTAuth based on MNO's SDK.

TABLE I: Cellular network based mobile OTAuth services worldwide (ranked by MNO's total number of subscriptions)

Product / Service*	MNO	Country / Region	Business Scenario
Number Identification [21]	China Mobile	Mainland China	Login, Registration
unPassword Identification [22]	China Telecom	Mainland China	Login, Registration
Number Identification [23]	China Unicom	Mainland China	Login, Registration
Operator Attribute Service [24]	Vodafone, O2, Three	UK	Identity verification
Mobile Connect [25]	América Móvil	Mexico	Login, Registration
Mobile Connect [1]	Telefónica Spain	Spain	Login, Registration
ZenKey [26]	AT&T, T-Mobile, Verizon	America	Login, Registration
Fast Login [27]	Turkcell	Turkey	Login
Mobile Connect [28]	Mobilink	Pakistan	Login, Registration
PASS [29], [30]	SKT, KT, LG Uplus	South Korea	Payment Identity verification
T-Authorization [31]	SKT	South Korea	Login, Registration Money transfer / Payment verification
Ipification-HK [32]	3 Hong Kong	Hongkong China	Login, Registration
Ipification-Cambodia [33]	Metfone	Cambodia	Login, Registration

* This table demonstrates the prevalence of mobile OTAuth services worldwide but does not imply all of them are vulnerable. In our research, we only confirmed the first three services [21]–[23] in mainland China are vulnerable for the SIMULATION attack. As of Mar 2022, we have got confirmation from the ZenKey experts, who told us that ZenKey for AT&T is not subject to this vulnerability as its authentication flow is different.

app developers by the MNO SDK vendor. The MNO SDK then collects the fingerprint of the signing certificate [34] inside its hosted app (i.e., *appPkgSig*), through the API *getPackageInfo* and sends it to the MNO server, together with the *appId* and *appKey* (step 1.3).

Since MNO has the capability of recognizing phone number, the MNO server already knows the phone number (i.e., *phoneNum*) after receiving the request data. Thus, after confirming that the *appId*, *appKey* and *appPkgSig* are legitimate, the MNO server returns the *user's masked phoneNum* to the MNO SDK, together with the *operatorType* (e.g., *CM* for China Mobile, *CU* for China Unicom, *CT* for China Telecom) to facilitate the app's display (step 1.4). Lastly, the MNO SDK pulls up an interface (like the ones shown in Figure 1) and asks for user's authorization (step 1.5). Here, the authorization refers to whether the user allows the app to obtain the *phoneNum*.

(2) **Request token.** In this phase, the app client obtains a *token*, which is associated with the *appId*, *appKey* and the *phoneNum*. With this token, the app server can learn the *phoneNum* in the next phase. If the user approves the obtainment of local phone number (step 2.1), MNO's SDK will send the *appId*, *appKey* and *appPkgSig* to the MNO server through cellular network again, in request for the *token* (step 2.2). After the *appId*, *appKey* and *appPkgSig* get verified, the MNO server will generate a *token* and send it back in response (step 2.3 and 2.4).

(3) **Obtain phone number.** In this phase, the app server will obtain the user's *phoneNum* and decide whether to approve the user's login or sign-up request based on this. First, the app client will send the *token* to the app server in request for login or sign-up (step 3.1). After receiving the *token*, the app server will send it to MNO server in exchange for the *phoneNum* (step 3.2). After confirming that the app server's IP is legitimate (i.e., has been filed) and that the *token* and *appId* are corresponding, the MNO server will respond the *phoneNum* to the app server (step 3.3). Based on the *phoneNum*, the app server can decide whether to approve or reject the app client's request (step 3.4).

C. OTAuth Ecosystem in Mobile Apps

Given the huge convenience of OTAuth schemes, many popular mobile apps have fully integrated this service. Some of them even set OTAuth login as the default login option. Based on a recent report from China Mobile (the largest MNO in China) [35], as of October 2021, its OTAuth service has been called more than 1.69 trillion times.

There are two ways for an app to introduce MNO's service. The app can either integrate the SDK developed by MNO, or integrate a third-party SDK that includes functions of all MNOs' SDKs. In mainland China, there are three MNOs: China Mobile, China Unicom, and China Telecom. Note that SDKs of all the three MNOs support authenticating through an arbitrary operator. For example, an app could utilize the SDK developed by China Mobile to seamlessly supports OTAuth services of the other two MNOs (e.g., China Unicom) as well.

Other than the official SDKs made by MNOs, there are various third-party SDKs that support OTAuth as well. These SDKs typically integrate MNO's SDKs and provide easier-to-use APIs for app developers to integrate. Such SDKs also include other authentication functions as a syndicator, such as authentication based on SMS One-Time-Password.

D. Scope of Our Study

In this paper, we focus on the OTAuth scheme. Particularly, as instances of this authentication scheme, our research looked into the OTAuth services provided by all the three MNOs in mainland China. Table I presents a list of OTAuth services we have found in different countries and regions. While there are similar OTAuth services in other countries and regions, they are not included in this study, due to the following reasons. Firstly, due to locality constrain, it is difficult for us to obtain the real SIM cards and perform the testing for OTAuth services in other regions. Secondly, unlike the OTAuth services deployed in mainland China, the OTAuth services provided in some other countries have not yet been widely deployed by app developers [3]. We envisioned our analysis and findings could bring insights for securing similar OTAuth services in

other countries and regions. For example, our preliminary investigation showed that the Fast Login [27] developed by Turkcell (the largest MNO in Turkey) is similar to the OTAuth schemes of three MNOs in mainland China.

In addition, previous works [36] have discussed related issues under quite different assumptions (e.g., assuming the attacker has physical access to the victim's SIM card and can perform side-channel power analysis) and these problems do not belong to the scope of the SIMULATION attack we proposed.

III. EXPLOITING OTAUTH SCHEMES

In this section, we show how to perform a SIMULATION attack by exploiting a critical design flaw in such OTAuth schemes. We first illustrate our attack model. Then, we present the core idea of the attack as well as its implementation details.

A. Attack Model

We assume the adversary can perform the attack under either of the following two scenarios (shown in Figure 5). In the first scenario, we assume the attacker can install an innocent looking malicious app to the victim device. Here, the malicious app does not need to require any sensitive permissions other than the INTERNET permission. Note that since the INTERNET permission is widely used by a large portion of normal apps for app-server communication nowadays, this permission can be easily obtained from the victim user. This assumption is aligned with many previous works that perform attacks on mobile platforms [7], [8]. In the second scenario, we assume the attacker is within the same network as the victim's device. This typically happens when the adversary connects to the hotspot shared by the victim's device. We consider this scenario is more likely to happen in an attack that targets a specific individual. For example, the adversary is a colleague of the target victim in the same company, and the adversary aims at logging in to the victim's account and stealing sensitive information.

The key point here for the above two scenarios is that we assume the adversary can perform her actions under the same cellular network IP address as the victim, for communicating with the MNO server (see section III-D for more details).

In the meantime, we assume that the victim is under the legitimate usage scenario of OTAuth provided by MNOs. Specifically, there is a SIM card on the victim's smartphone and the Mobile Data switch has been turned on. Note that since the OTAuth scheme only takes the cellar network as the authentication channel, our attack can succeed regardless of whether the victim phone's WLAN switch has been turned on.

B. Attack Overview

Our research identified that, due to a fundamental design flaw in the OTAuth scheme, the MNO server cannot effectively validate whether the authentication request is sent from a legitimate client or a malicious one. Therefore, under certain scenarios, an attacker can easily obtain the authentication token that is associated with the victim's phone number. With this token, the attacker can log in to the victim's account on the attacker's own smartphone.

Root cause. The root cause of the flaw is the app's incapability of securely using mobile device identity. The operating system does not participate in the design architecture of OTAuth. Such a flawed design makes the MNO server unable to distinguish different apps on the same device, which makes the SIMULATION attack possible.

As mentioned earlier in Section II-B, the MNO server verifies the app client via three factors, namely, the *appId*, *appKey* and *appPkgSig*. Unfortunately, all such information are not confidential and can be easily obtained by an attacker. From the MNO server's perspective, there is *no way* to effectively identify whether the one requesting *token* is indeed a legitimate one. More specifically, if the attacker makes the authentication request under the same network environment (i.e., via a malicious app on the victim device, or connecting to the victim's hotspot), the MNO server will always return the valid token since the authentication factors it received are indeed correct.

Impacts. Exploiting this design flaw, the attacker can bypass the app's authentication and log in to the victim's account. In other words, the attacker can remotely log in to the victim's account through an app client (on her own device) and continue to perform malicious actions. When the victim's phone number is not bound to any account, the attacker can register new accounts on behalf of the victim. In addition, the attacker can also easily obtain the victim's phone number (e.g., log in a specific app that displays the phone number on the app's user-profile page).

C. Attack Details

We divide the whole attack process into three phases, as described in Figure 4. Here, the $appId$, $appKey$ and $appPkgSig$ are specific to the affected victim app; $phoneNum_A$ and $phoneNum_V$ refer to the attacker's local phone number and the victim's local phone number, respectively; $token_A$ and $token_V$ refer to the valid *token* distributed to the attacker and the victim by the MNO server, respectively.

(1) Token stealing phase. In this phase, the attacker launches the malicious app to obtain a $token_V$. Specifically, the attacker "simulates" the behavior of the MNO SDK and sends the $appId$, $appKey$, and $appPkgSig$ to the MNO server (step 1.1 and 1.3). As mentioned earlier, these three pieces of data are not confidential and can be obtained through various ways in advance. For example, many app developers hard-code the $appId$ and $appKey$ in the source code of their distributed apps, which can be trivially recovered via reverse engineering. The $appPkgSig$ can be obtained by Keytool [37] when the app (i.e., the APK file) is given. In addition, the attacker can also intercept the network traffic of the legitimate OTAuth scheme (e.g., on her own device) and obtain these information.

(2) Legitimate initialization phase. In this phase, the attacker performs the normal OTAuth process of the victim app on her own smartphone (step 2.1 to 2.7). This is because the attacker

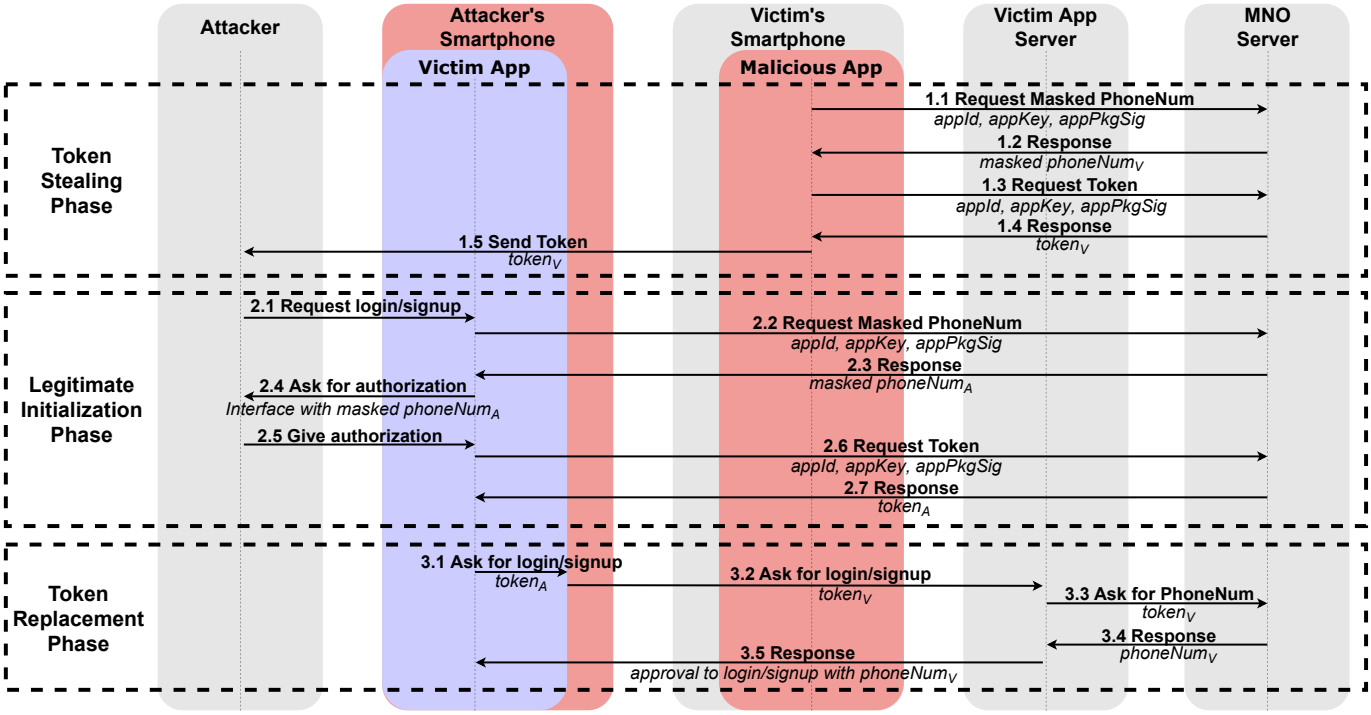


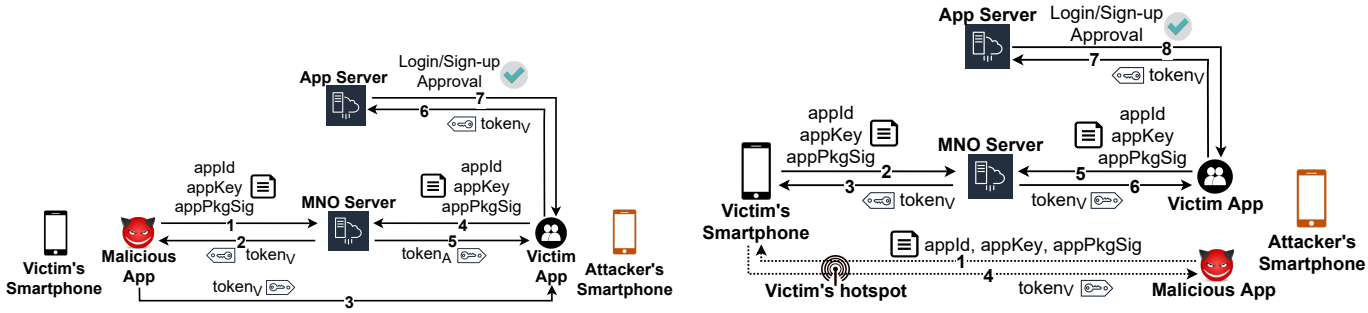
Fig. 4: The attack model against OTAuth scheme.

needs to launch a legitimate app client to communicate with the victim app's back-end server for her (future) unauthorized login. Note that, since the operations in this stage are performed entirely on the attacker's smartphone, the attacker has complete control over the entire process. Therefore, the attacker has the ability to initialize the authentication, and in the meantime, prevent the app client from sending $token_A$ to the app's back-end server. More specifically, the attacker can use the hooking technique [38] to intercept and block the legitimate authentication process. The $token_A$ will be further replaced by $token_V$ for the upcoming authentication scheme.

(3) Token replacement phase. In this phase, the attacker bypasses the authentication of the app's backend server by replacing the $token_A$ with the previously obtained $token_V$ (step 3.1 and 3.2). Since the $token_V$ is a valid token associated with the $appId$ (exclusive to **victim app**) and the $phoneNum_V$, the app's back-end server will get the $phoneNum_V$ when it tries to exchange the received $token_V$ for a phone number (step 3.3 and 3.4). In this case, the app's backend server mistakenly treats the attacker as the holder of the $phoneNum_V$ and approves the login (sign-up) request on the attacker's device.

D. Attack Implementation

Attack via a malicious app. The overall process of this type of attack is shown in Figure 5(a). In this attack, with the installed malicious app, the attacker can obtain the victim's token by sending app-specific data through the victim's mobile network. As an instance of this attack, we take Kuaishou [39], Chinese version of Kwai [40] and one of the most popular short video apps with more than 400 million active users [41], as the target app for exploitation. We implemented the malicious app and hard-coded the *appId*, *appKey* and *appPkgSig* of Kuaishou in it. We uploaded the malicious app to VirusTotal [42] on April 6 in 2022 and VirusTotal reported that "No security vendors flagged this file as malicious". To perform the attack, we installed the malicious app on a non-rooted Nokia X5 phone (as the victim's device) with Android 9 OS. The app installation process does not trigger any security



(a) Attack via a malicious app.

(b) Attack by connecting to victim's hotspot.

Fig. 5: Two attack scenarios against OTAuth scheme implemented by our research.

alert by the system. The entire token obtainment process does not require any interaction from the victim user, and will not cause any detectable phenomena (e.g., popping up permission requests or risk warnings). Lastly, we complete the attack by hooking and replacing the token sent by a genuine app on the attacker's device (tampering with the app). The demo video of this attack is presented in https://simulation.code-analysis.org.

Attack by connecting to the victim's hotspot. The main process of attacking through hotspot connection is very similar to the previous attack we have presented, as shown in Figure 5(b). In this scenario, we assume that the attacker can connect his own smartphone to the mobile hotspot of the victim's mobile phone. Thus, the attacker can send app-specific data to the MNO server through victim's cellular network. The attacker follows the same subsequent steps as the previous attack, and can successfully bypass the authentication scheme on the attack device side.

Note that in this attack, the OTAuth SDK checks the network status of the smartphone (e.g., checks the Operator Type of the device) and may expose that the attacker's device has a different cellular network status as the victim. However, this check can be easily bypassed since the attacker has full control over the system of the attacker's device. More specifically, since this check is implemented by the SDK through specific methods (e.g., *android.net.ConnectivityManager.getActiveNetworkInfo*, *android.telephony.TelephonyManager.getSimOperator*), we overloaded the corresponding methods to explicitly return true statements every time the check is performed. A demo video of this type of attack against the Sina Weibo app [43] (the most popular microblogging app in China, similar to Twitter) is also posted online (https://simulation.code-analysis.org).

IV. Large-scale measurement study

To better understand how real-world apps are affected by the issues identified above, we perform a large-scale measurement over a set of popular Android and iOS apps. Our results show a large portion of highly popular apps confirmed to be vulnerable due to the integration of such OTAuth Scheme.

A. Dataset

Our final dataset includes a total number of 1,025 Android apps and 894 iOS apps. These apps were downloaded between July 19, 2021 and November 20, 2021. As most app stores explicitly prohibit any form of automated app info collection [44] [45], we referred to Qimai Data [46], a third-party mobile app data analysis platform to help label the most popular apps.

Android app set. To build the Android app set, we first identified an app list containing 15,668 apps based on the 17 unique app categories provided by Huawei App Store [47] (i.e., top 1,000 apps for each category). Further, based on the download statistics collected from Qimai Data, we selected all those apps with more than 100 million downloads. Note that users in mainland China rarely use Google Play as their source for app downloading, and Huawei App Store is one of the most popular markets instead. As a result, apps collected from this source can represent well the prevalence of OTAuth SDK usage in Chinese markets. We consider such a dataset can sufficiently cover a wide range of highly popular app across different categories.

iOS app set. Since Apple does not reveal the number of downloads for each app in its App Store [48], we collected iOS apps by referring to our Android app set. The correspondence between the Android and iOS versions of an app is provided by Qimai Data [46]. We used a jailbroken iPhone 7 plus (with iOS 13.4.1) to manually install the corresponding apps and dumped their binary executables using flexdecrypt [49]. In this way, we finally collected 894 iOS apps corresponding to our Android app set. Note that since not every Android app in our data set has an iOS counterpart, the total number of apps in this dataset is not identical to the Android app set.

B. Measurement Approach

TABLE II: API signatures collected from the three MNO OTAuth SDKs

	MNO	API signature
Android	CM	com.cmic.sso.sdk.auth.AuthnHelper
	CU	com.unicom.xiaowo.account.shield.UniAccountHelper com.unicom.xiaowo.account.shieldjy.UniAccountHelper
	CT	cn.com.chinatelecom.account.sdk.CtAuth cn.com.chinatelecom.account.api.CtAuth cn.com.chinatelecom.gateway.lib.CtAuth cn.com.chinatelecom.account.lib.auth.CtAuth
iOS	CM	"https://wap.cmpassport.com/resources/html/contract.html"
	CU	"https://opencloud.wostore.cn/authz/resource/html /disclaimer.html?fromsdk=true"
	CT	"https://e.189.cn/sdk/agreement/detail.do"

"CM", "CU", and "CT" refer to China Mobile, China Unicom, and China Telecom respectively.

Challenges in automatic detection. To identify those vulnerable apps in our Android app set, a naive approach could be statically checking whether there are certain MNO SDK signatures in the decompiled app code. For example, such signatures could be a unique string that contains the package name and class name in the SDK. However, our preliminary analysis showed this mechanism is far from satisfying. Specifically, using the signatures collected from the three MNO OTAuth SDKs (see the list presented in Table II), we only located 271 out of 1,025 (around 26%) apps from our Android app set, a much lower percentage than what we observed. Our further manual inspection showed there are a number of reasons causing such a low coverage rate.

- **Code obfuscation and app packing.** A large number of apps integrate anti-reverse engineering techniques such as code obfuscation (e.g., ProGuard [50]) and app packing [51], [52] for intellectual property protection. Both of these mechanisms may cause false negatives during SDK signature detection.
- **Third-party SDKs without MNO SDK signatures.** It is also possible that some third-party SDKs implement

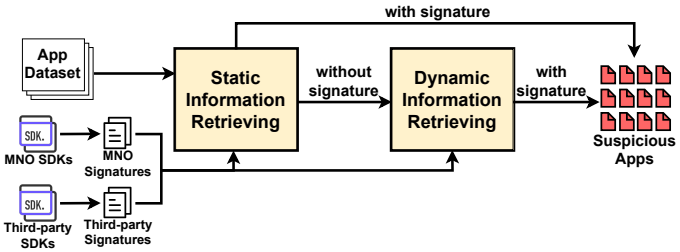


Fig. 6: Overview of our analysis pipeline for identifying potentially vulnerable apps.

their own app-level logic for service integration, instead of directly including the MNO OTAuth SDKs. For example, for third-party SDKs like U-Verify [53], while they indeed integrate the MNO OTAuth services, we did not find any relevant code of MNO OTAuth SDKs from their decompiled code.

Given the challenges above, in our research, we employ a mix of static and dynamic analysis mechanisms to more effectively locate those apps that are vulnerable to SIMULATION attack. Figure 6 shows the overall pipeline of our approach.

Static information retrieving. In the static analysis process, we searched for SDK signatures from the decompiled app code with the help of dexlib2 [54] library. If an app contains a class that matches one of our signatures (e.g., class name and package name), we consider that this app has integrated OTAuth service. As mentioned earlier, in addition to the selected signatures from MNO SDKs, we employ a set of heuristics to recover other OTAuth related signatures (i.e., the signature of other third-party SDKs) and include them to the SDK signature set.

The additional SDKs and signatures other than the MNO SDKs are collected through the following mechanisms.

- **Third-party SDKs' official website.** Some MNOs list part of the third-party agents they have collaborated with on their official website (e.g., China Telecom lists their collaborators on [55]). In addition, we also collect other third-party agents that are not listed by the MNOs by active searching. Then, we collect their released SDKs from their websites and select their signatures.
- **Individual apps highlighted by third-party agents.** While some third-party agents do not provide their SDKs for public download, such SDKs would typically list specific apps that integrate their services. To this end, we find their corresponding apps and manually identify their SDKs as well as the SDK signatures through reverse engineering such apps.

As a result, with the collected SDK signatures, we can effectively capture more affected apps. Note that our investigation showed code-obfuscation does not have significant impact on our analysis. Specifically, both the documentation of MNO SDKs and third-party SDKs explicitly asked the developers to not obfuscate the SDK code to ensure its usability, meaning that the SDK signatures can be preserved well in those apps.

Dynamic information retrieving. For the remaining apps that were not detected by static analysis, we perform an additional round of dynamic analysis and inspect whether the instances of OTAuth related classes (based on the collected SDK signatures) indeed exist at runtime. More specifically, for each app, we automatically install and launch the app through ADB [56]. Then, we use Frida [38] to load the specific classes in the SDK signatures via *ClassLoader*. If relevant SDK is not integrated into the app, then a *ClassNotFoundException* will be caught; otherwise, the API class will be loaded successfully, indicating that the corresponding SDK indeed exists in the tested app. In this way, we can find more candidate apps that are missed by the static analysis.

For iOS apps, we statically check whether the decompiled code contains the previously obtained SDK signatures that are generic to both Android and iOS. Particularly, here we inspect the URLs used by the OTAuth protocol since the package name and class name of the same SDK could be different in the two platforms. Note that for iOS apps, we do not need to perform the dynamic analysis since Apple does not allow apps with packed or obfuscated code to be published in App Store [48].

While such mechanisms cannot guarantee the reported suspicious apps to be indeed vulnerable, the results here provide a significantly more accurate scope for identifying affected apps.

C. Results and Findings

Affected Apps. The measurement results of apps are summarized in Table III. In total, among the 1,025 Android apps, we identified a total number of 396 apps that are affected by the SIMULATION attack. For the iOS dataset, 398 out of 894 apps are affected.

TABLE III: Overview of app measurement results

	Total	Detection Result	S	S&D	Verification Result		P	R
Android	1025	suspicious	279	471	TP	396	0.84	0.72
					FP	75		
		unsuspicious	746	554	TN	400		
					FN	154		
iOS	894	suspicious	496	\	TP	398	0.80	0.78
					FP	98		
		unsuspicious	398	\	TN	287		
					FN	111		

S indicates the result of static retrieving, **S&D** indicates the result of static and dynamic retrieving; **P** and **R** refer to the precision and the recall respectively. In the following rows, **TP**, **FP**,**TN**, and **FN** refer to True Positive, False Positive, True Negative, and False Negative respectively.

To detail, for Android apps, our static information retrieving process enables us to locate 279 candidates. The dynamic information retrieving process helps us to locate 192 additional candidates. To this end, we have a total number of 471 candidates through the automated analysis process. Lastly, our manual verification confirmed that 396 out of these 471 apps are indeed vulnerable to the SIMULATION attack. For iOS apps, our static analysis located 496 suspicious apps and we finally confirmed 398 of them are indeed vulnerable. Table IV

lists the most popular vulnerable apps based on the number of their MAU.

TABLE IV: Identified top apps in the Android and iOS dataset that have more than 100 million monthly active users (MAU) in September 2021, according to the statistical results of IiMedia Polaris [11].

App	Category	MAU*	App	Category	MAU*
TikTok	short video	578.85	Sina Weibo	community	311.60
Baidu Input	input method	569.46	WiFi Master Key	Wi-Fi	285.57
Baidu	mobile search	474.62	TouTiao	comprehensive information	265.21
Gaode Map	map navigation	465.27	Pinduoduo	integrated platform	237.26
Kuaishou	short video	436.50	Dianping	local life	156.63
Baidu Map	map navigation	379.58	DingTalk	office software	143.57
Youku	comprehensive video	367.19	Meitu	picture beautification	139.47
Iqiyi	comprehensive video	350.90	Moji Weather	weather calendar	122.61
Kugou Music	music	321.29			

* **MAU** refers to the amount of Monthly Active Users (in millions).

Compare to the naive solution that only considers statically retrieving the signatures of MNO SDKs, our mixed static and dynamic analysis mechanisms significantly improve the coverage for this task by finding 73.8% (271 v.s. 471) more suspicious apps for the Android dataset. In addition, our analysis pipeline also achieves high precision and recall. Particularly, 84.08% (396/471) and 80.24% (398/496) of the identified Android and iOS apps are indeed vulnerable respectively, while 72% (396/550) vulnerable Android apps and 78.19% (398/509) vulnerable iOS apps are pointed out by our analysis pipeline.

Below we discuss the false positives and false negatives during our automated analysis process. For simplicity, we here mainly discuss the apps in our Android dataset, and the iOS dataset follows a similar case.

For the Android dataset, our detection method resulted in 75 false positives. The main reasons for these false positives are as follows: (1) 5 out of 75 apps suspended user login or sign-up for various reasons (e.g., under national cyber security review), thus they will not be affected by the SIMULATION attack temporarily. (2) While some apps do integrate the OTAuth SDK, we found they do not actually call any SDK APIs when the user requests to log in. This may happen when an app has integrated an SDK supporting the OTAuth feature (e.g., Tencent Cloud SDK [57]) but the SDK is actually used by the app for other features (e.g., login with Tencent WeChat account). We found that 62 out of 75 apps belong to this case. (3) For the remaining 8 apps, while supporting OTAuth login feature, they also adopt additional verification for user authentication. For example, if the user tries to log in on a new device, Douyu TV [58] will require SMS One-Time Password and Codoon [59] will require the full phone number. We consider such apps are not vulnerable to the SIMULATION attack.

In terms of false negatives, our approach missed 154 vulnerable apps. This is mainly because some apps have integrated more advanced packing techniques to hide the code level semantics at runtime, making the valid signature undetectable by our mechanisms. Specifically, we automatically detected the common packing tool signatures in the 154 apps we missed and 135 of them are judged to be packed. A manual inspection of the remaining 19 missed apps showed that they have implemented more customized packing techniques. Although false negatives exist, our detection mechanism provides a concrete lower bound, showing the severity of our identified issues. Particularly, our detection mechanism showed that at least 38.63% (396/1025) of the apps in our Android dataset are vulnerable to the attack.

Affected SDKs. In addition to the three SDKs provided by the official MNOs, our aforementioned SDK collection process has identified a total number of 19 third-party SDKs that integrate such services for app developers. The detailed information of such SDKs is presented in Table V. Among them, 8 SDKs are found to exist in our app dataset. Since the root cause of SIMULATION attack is the insecure design of the authentication scheme, all our investigated OTAuth SDKs (the MNO SDKs and third-party SDKs) are vulnerable to the SIMULATION attack.

TABLE V: Details about third-party OTAuth SDKs covered by our research

Third-party SDK	Publicity[1]	App Num	Third-party SDK	Publicity[1]	App Num
Shanyan [60]	✔	54	Jiguang [61]	✔	38
GEETEST [62]	✔	25	U-Verify [53]	✔	18
NetEase Yidun [63]	✔	10	MobTech [64]	✔	8
Getui [65]	✔	8	Shareinstall [66]	✔	4
SUBMAIL [67]	✔	0	Jixin [68]	✘	/
Emay [69]	✔	0	Qianfan Cloud [70]	✘	/
Tencent Cloud [57]	✘	/	Baidu AI Cloud [71]	✔	0
Up Cloud [72]	✔	0	Santi Cloud [73]	✔	0
Huitong [74]	✔	0	Weiwang [75]	✔	0
DCloud [76]	✔	0			
Total Num			163 [2]		

[1] **Publicity** indicates whether the third-party agent has published its OTAuth SDK or highlighted apps.
[2] **Two apps** integrate GEETEST [62] SDK and Getui [65] SDK at the same time.

Impacts of SIMulation Attacks. According to the statistical report of CNNIC [77], the total number of mobile internet users in mainland China has surpassed 1 billion by June 2021 and nearly all of them use services provided by the three major MNOs. Since OTAuth service is enabled by default, the SIMULATION attack could potentially affect all users of the three MNOs. Even worse, according to our observations, neither the MNOs nor the apps have provided an option that allows users to disable this login scheme, meaning that it's hard to alleviate this threat from the user side.

In the meantime, among the vulnerable apps we have identified, 17 apps have more than 100 million MAU (shown in Table IV) and 87 apps have more than 10 million MAU[3]. Therefore, if the SIMULATION attack could be conducted on an arbitrary mobile device (either with Android or iOS), it

[3] Based on statistics published by IiMedia Polaris [11].

is very likely that the phone number has been registered to several popular apps.

In addition to the major design flaw discussed in Section III, we also discovered several additional issues that lead to extra unexpected risks. We summarized them as follows:

- *User Identity Leakage*. The original design of OTAuth only returns a masked phone number (e.g., "186******98") when receiving the OTAuth request. This partially leaks the sensitive information of the user identity. Even worse, we found a further step could be leveraged to fully disclose the victim's phone number: when receiving a valid *token*, some app servers not only send it to the MNO server to obtain the phone number of the user but also respond this phone number to the user (app). Such an app server can be easily abused as an oracle to obtain the victim's phone number. Examples of affected apps in this case include ESurfing Cloud Disk [78], a highly popular private cloud storage app with more than 400 million users.
- *OTAuth Service Piggybacking*. To use OTAuth service, developers are required to register their apps and pay the corresponding fees. However, an app could abuse the OTAuth service of other registered apps to implement a free and unauthorized use. Similar to the user identity leakage cases, once a registered app could be abused as an oracle to retrieve user's phone number, the malicious app easily reuses the *appId* and *appKey* of the victim app to first obtain a *token* from the MNO server, and then uses the *token* to exchange phone number from the app server. In this way, the malicious app freely uses the OTAuth service without the permission of both MNO and app servers. More seriously, we noticed that the use of OTAuth service is not free. For the legitimate use of OTAuth, an app needs to pay a certain fee to the MNOs or third-party agents for each login. For instance, China Telecom charged a 0.1 RMB (around 0.016 USD) service fee for each OTAuth [79]. If the OTAuth service of a legitimate registered app is frequently abused by those unregistered apps, the legitimate app would suffer from a lot of unexpected expenses.
- *Account Registration without User Awareness*. We observed that a large portion of app providers not only integrated OTAuth services but also simplified their app account registration and activation processes when a user uses the OTAuth for the first time: if the used phone number has not yet been registered to the app service, it will be automatically registered without any user involvement. While this automated process facilitates new users, it actually expands the attack surface of the SIMULATION attack. Even if a user would not like to use a certain app, SIMULATION attack could exploit this (insecure) design to associate her phone number to a new account. In our research, our manual investigation confirmed that 390 out of 396 vulnerable Android apps allow an adversary to register a new account without any additional information. In other words, for these 390 apps, if the victim's phone number has not been used for registration, the attacker can register a new account with the victim's phone number.

D. Other Implementation Weaknesses

Our analysis of the identified apps and SDKs also revealed a set of additional implementation weaknesses of the OTAuth services, involving both SDK developers and app developers.

Insecure token usage. As an important credential, the use of *token* should have been strictly restricted. However, in reality, some MNO's restrictions on tokens are not strict enough, mainly including: (1) *Token* reuse. In theory, each *token* should be invalidated after being sent to MNO server by app server (in step 3.2 of Figure 3). However, after experiments, we found that in China Telecom's OTAuth service, a *token* can be used to complete multiple logins within its valid time. In addition, during the validity period of *token*, the *token*s obtained by multiple requests of the app client remain unchanged. (2) Multiple effective *token*s. Similar to SMS OTP, there should be only one valid *token* at a time. However, in China Unicom's OTAuth service, newly obtained *token* will not invalidate the older *token*. (3) Too long validity period. Among the three MNOs in mainland China, China Mobile, China Unicom and China Telecom have set *token* validity period to be 2 minutes, 30 minutes, and 60 minutes, respectively. We believe the latter two MNOs have set a too long validity period, which brings risks to security.

Authorization without user consent. OTAuth SDKs require the app to obtain user's mobile phone number only after obtaining user's authorization (step 1.5 and step 2.1 of Figure 3). MNO's SDKs and third-party SDKs do pop up an interface (see Figure 1) to ask for user's authorization. While MNOs and third-party agents ensure that the interface indeed pops up (e.g., through resource protection or manual review), we discovered that some popular real-world apps have retrieved the *token* before popping up the interface. With this *token*, these apps can easily obtain the user's phone number without user's authorization.

Plain-text storage of sensitive information. According to the design of MNO, the *appId* and *appKey* of an app are specific and fixed. Through analyzing real-world apps, we found that many apps have hard-coded their *appId* and *appKey* into program files in plain-text form, which makes it easy for an attacker to obtain the *appId* and *appKey*.

V. MITIGATION

We observed that both the OTAuth SDK vendors and the app developers have adopted some ineffective strategies to protect the OTAuth scheme. We summarized and analyzed typical (insecure) defenses as follows:

- **Using app hardening technique to hide *appId* and *appKey***. Many app developers applied app hardening techniques such as code obfuscation, packing, or anti-debugging to prevent their apps from being reverse-

engineered. However, such protection cannot fundamentally prevent attackers from retrieving *appId* and *appKey*.

- **Using *appPkgSig* to verify the client's legitimacy**. The MNO server asks the SDK to obtain the *appPkgSig* of the app client from the OS, and adds it in request, trying to identify whether the request is sent from a benign app client. Unfortunately, attackers could easily replace the *appPkgSig* sent by the malicious app client (e.g., through patching the SDK or impersonating the benign app client to send network packets).
- **Using UI-based confirmation to enforce user's involvement**. The OTAuth SDKs, as we observed, would prompt the user (as the blue boxes in Figure 1 show) during an OTAuth process. In addition, an app with OTAuth SDK must be vetted by MNOs to ensure there exists such a confirmation before the SDKs are invoked. However, such a design cannot guarantee that the user really involves in the OTAuth process, since it needs no user-related input to construct the login request.

We argue that the failure of the above-mentioned defenses is they cannot fundamentally prevent an attacker from impersonating a legitimate app. Correspondingly, we proposed the following countermeasures by adding certain factors that only a legitimate app and its user could generate:

- **Adding user-input data into the login request.** The OTAuth process could require users to provide some information that is unknown to the attacker (e.g., her full phone number or her family name). However, this may raise usability issues and affect the user experience.
- **Adding OS-level support.** OS has the capability of dispatching a *token* to the legitimate app (i.e., the app with the corresponding package name). Thus, even if a malicious app can send a login request, it cannot obtain the *token* and perform the SIMULATION attack. However, this may require a deeper cooperation between the OS vendors and the MNOs.

VI. Related Work

We discuss the related work to our research with the following two categories, namely, bypassing authentication on mobile platform, and MNO's service that is related to mobile authentication.

Bypassing authentication on mobile platform. Many recent studies have paid attention to the security risks in authentication process on mobile platforms. These researches have achieved similar attacking results to our works, such as log in to the victim's account without authorization. The major difference between these works and ours is that they focus on different authentication schemes provided by apps, instead of third-parties (e.g., the MNOs). For example, Song et al. [6] developed an Android OS-level virtualization platform that is called VPDroid, to exploit the automatic login feature implemented by individual apps. Bianchi et al. [80] explored the unsafe login-less authentication schemes in which distinguishing information of user device are used. Lei et al. [8] revealed the insecurity of SMS-related new APIs provided by Android systems, and they showed how to abuse such features to perform user-interaction-free unauthorized login. Jubur et al. [7] demonstrated the feasibility of bypassing app's push-based authentication scheme by triggering human-indistinguishable notifications. Wang et al. [81], [82] focused on the security risks of apps that adopted the OAuth-based authentication schemes.

MNO's service for mobile authentication. Traditionally, most Mobile Network Operators provide SMS-based authentication services (i.e., the One-Time Password), which are widely used by mobile apps. Following this line of topic, there are a wide range of malwares (e.g., ZitMo [83], SPITMO & Tatanga MITMO [84], and Crusewind [85]) that target stealing the SMS message for bypassing the user authentication. In addition, Enck et al. [86] and Golde [87] conducted research on abusing SMS to perform DoS attacks. Their research showed that SMS can be exploited to prevent phone users from making calls or exhaust the user's phone balance.

Other than attacks targeting SMS, the SIM Swapping attack [9] exploits MNO's ability of seamlessly binding a phone number to a new SIM card, which can further route the authentication SMS to the attacker. SIM Clone attack allow attackers to obtain a SIM card, which is almost equivalent to the victim's. Liu et al. [36] presented how to copy a 3G/4G USIM card within 15 minutes through Side-Channel Analysis. Coletta et al. [88] revealed the risk of personal data leakage caused by MNO's self-authentication feature. Specifically, they studied seven major Italian MNOs and found that attackers could steal user data (such as phone number, the amount of phone calls, etc.) by visiting specific web pages.

VII. Conclusion

In this paper, we conduct the first in-depth security study on the Mobile Network Operator based One-Tap Authentication scheme (MNO-based OTAuth scheme). Our research has identified several fundamental design flaws in this authentication scheme. Such design flaws bring severe security implications to apps that integrate this type of service. For example, an adversary who bypasses this authentication scheme can gain full access to the victim's app account. To further evaluate the impact of our identified issues to the real world, we performed a large-scale measurement over a set of top popular apps in mainland China (including both Android and iOS apps). The measurement results showed that a large portion of highly popular apps are affected by this issue. Lastly, we discuss possible ways to mitigate such threats.

Acknowledgment

We are grateful to our shepherd for his support and suggestions. This work was partially supported by the National Key Research and Development Program of China (No.2020AAA0107803) and the National Natural Science Foundation of China (No.62002222). We especially thank Ant Group for the support of this research within the SJTU-Ant Security Research Center.

REFERENCES

[1] Telefónica Spain. (2017) Mobile Connect on Mi Movistar: Improving adoption and usage. [Online]. Available: https://mobileconnect.io/wp-content/uploads/2019/02/mc-Telefonica-Spain-Improving-adoption-usage.pdf

[2] GSM Association. (2021) GSMA: Representing the worldwide mobile communications industry. [Online]. Available: https://www.gsma.com/

[3] GSM Association. (2021) Mobile Connect. [Online]. Available: https://www.gsma.com/identity/mobile-connect

[4] China Mobile. (2020) Introduction to the One-Tap Login Authentication Service of China Mobile. [Online]. Available: http://dev.10086.cn/docInside?contentId=10000067529678

[5] China Telecom. (2021) Jingdong app accesses Passwordless authentication service. [Online]. Available: https://id.189.cn/partnerCase/detail?type=jd_details

[6] W. Song, J. Ming, L. Jiang, H. Yan, Y. Xiang, Y. Chen, J. Fu, and G. Peng, "App's auto-login function security testing via android os-level virtualization," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1683–1694.

[7] M. Jubur, P. Shrestha, N. Saxena, and J. Prakash, "Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 447–461.

[8] Z. Lei, Y. Nan, Y. Fratantonio, and A. Bianchi, "On the insecurity of sms one-time password messages against local attackers in modern mobile devices," in *Proceedings of the 2021 Network and Distributed System Security (NDSS) Symposium*, 2021.

[9] K. Lee, B. Kaiser, J. Mayer, and A. Narayanan, "An empirical study of wireless carrier authentication for {SIM} swaps," in *Sixteenth Symposium on Usable Privacy and Security ({SOUPS} 2020)*, 2020, pp. 61–79.

[10] Wikipedia. (2021) List of mobile network operators. [Online]. Available: https://en.wikipedia.org/wiki/List_of_mobile_network_operators

[11] IiMedia Polaris Developer Service Platform. (2021) IiMedia Polaris: the world's leading mobile network product benchmarking analysis platform. [Online]. Available: http://bjx.iimedia.cn/app_rank

[12] CNCERT/CC. (2021) National Computer Network Emergency Response Technical Team/Coordination Center of China. [Online]. Available: https://www.cert.org.cn/publish/english/index.html

[13] J. D. Clercq, "Single sign-on architectures," in *International Conference on Infrastructure Security*. Springer, 2002, pp. 40–58.

[14] S. Ma, R. Feng, J. Li, Y. Liu, S. Nepal, E. Bertino, R. H. Deng, Z. Ma, and S. Jha, "An empirical study of SMS one-time password authentication in Android apps," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 339–354.

[15] China Unicom. (2018) Passwordless Login for Xiaowo Account. [Online]. Available: https://saas.wostore.cn/saas/manager/userfiles/1/files/cms/article/%E5%B0%8F%E6%B2%83%E8%B4%A6%E6%88%B7-%E5%85%8D%E5%AF%86%E7%99%BB%E5%BD%95.pdf

[16] China Telecom. (2021) Industry Cases of China Telecom's Password-free login. [Online]. Available: https://id.189.cn/partnerCase/home

[17] 3GPP, "3GPP System Architecture Evolution (SAE); Security architecture," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 33.401, 12 2021, version 17.0.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2296

[18] 3GPP, "Security architecture and procedures for 5G System," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 33.501, 01 2022, version 17.4.2. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169

[19] 3GPP, "Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 24.301, 01 2022, version 17.5.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1072

[20] 3GPP, "3G security; Security architecture," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 33.102, 07 2020, version 16.0.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2262

[21] China Mobile. (2021) China Mobile Internet Capability Opening Platform. [Online]. Available: http://dev.10086.cn/numIdentific

[22] China Telecom. (2021) Tianyi Account Open Platform. [Online]. Available: https://id.189.cn/banner/unPassword

[23] China Unicom. (2021) China Unicom Communication Innovation Capability Platform. [Online]. Available: https://onlinebusiness.10010.com/product/5302?chnl=none

[24] O2, Three & Vodafone. (2017) Mobile Identification goes Live UK. [Online]. Available: https://mobileconnect.io/wp-content/uploads/2019/02/mc-Mobile-Identification-goes-Live-UK.pdf

[25] América Móvil. (2016) Increasing usage of Value-Added Services. [Online]. Available: https://mobileconnect.io/wp-content/uploads/2019/02/mc-America-Movil-Mexico-Increasing-VAS-usage-2.pdf

[26] ZENKEY LLC. (2021) Documentation, APIs & Portal | ZenKey Developer. [Online]. Available: https://myzenkey.com/developer/

[27] Turkcell. (2021) What is Fast Login? All About Turkcell Fast Login! [Online]. Available: https://hizligiris.turkcell.com.tr/en/fast-login/what-is-fast-login

[28] Mobilink. (2016) Success Story: Increasing Registrations on Operator Services. [Online]. Available: https://mobileconnect.io/wp-content/uploads/2019/02/mc-Mobilink-Increasing-reg-operator-services.pdf

[29] ATON Inc. (2019) PASS: Life Innovator Group. [Online]. Available: http://www.atoncorp.com/aton/en/pass-en/

[30] KT Co., Ltd. (2021) PASS | KT. [Online]. Available: https://fido.kt.com/ktauthIntro

[31] SK Telecom Co., Ltd. (2021) SKT ID. [Online]. Available: https://www.skt-id.co.kr/memberportal/login/main

[32] IPification. (2020) IPification Secure Mobile Authentication Available to 3 Hong Kong Subscribers. [Online]. Available: https://www.ipification.com/press/ipification-secure-mobile-authentication-available-to-3-hong-kong-subscribers

[33] IPification. (2021) IPification: Passwordless, One-Click Mobile Authentication Partners with Metfone in Cambodia. [Online]. Available: https://www.ipification.com/press/ipification-passwordless-one-click-mobile-authentication-partners-with-metfone-in-cambodia

[34] Google Developers. (2021) Sign your app. [Online]. Available: https://developer.android.com/studio/publish/app-signing

[35] China Mobile. (2021) China Mobile Capability Store. [Online]. Available: https://open.10086.cn/#/capability/14

[36] J. Liu, Y. Yu, F. Standaert, Z. Guo, D. Gu, W. Sun, Y. Ge, and X. Xie, "Small tweaks do not help: Differential power analysis of milenage implementations in 3G/4G USIM cards," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 468–480.

[37] Oracle. (2021) keytool. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

[38] Oleavr. (2021) Frida: A world-class dynamic instrumentation framework. [Online]. Available: https://frida.re/docs/functions/

[39] Kuaishou Technology. (2022) Kuaishou Short Video App. [Online]. Available: https://www.kuaishou.com/

[40] Kuaishou Technology. (2022) Kwai, Fantastic Social Video Network. [Online]. Available: https://www.kwai.com/

[41] Kuaishou Technology. (2022) Kuaishou Technology Announces Fourth Quarter and Full Year 2021 Financial Results. [Online]. Available: https://ir.kuaishou.com/news-releases/news-release-details/kuaishou-technology-announces-fourth-quarter-and-full-year-2021

[42] VirusTotal. (2021) Analyze suspicious files and URLs to detect types of malware, automatically share them with the security community. [Online]. Available: https://www.virustotal.com/gui/home/upload

[43] Sina. (2021) SINA English. [Online]. Available: https://english.sina.com/weibo/

[44] Apple Inc. (2021) Apple Website Terms of Use. [Online]. Available: https://www.apple.com/legal/internet-services/terms/site.html

[45] Huawei Device Co., Ltd. (2021) Terms of Use of Huawei Consumer Business Official Website. [Online]. Available: https://consumer.huawei.com/en/legal/terms-of-use/

[46] Beijing Qimai Technology Co., Ltd. (2021) Qimai Data (formerly ASO100): Professional mobile product business analysis platform. [Online]. Available: https://www.qimai.cn/

[47] Huawei App Store. (2021) Huawei App Store. [Online]. Available: https://appstore.huawei.com/

[48] Apple Inc. (2021) App Store. [Online]. Available: https://www.apple.com/app-store/

[49] JohnCoates. (2021) Flexdecrypt: Decrypt iOS Apps and Mach-O binaries. [Online]. Available: https://github.com/JohnCoates/flexdecrypt

[50] Guardsquare N.V. (2022) ProGuard: Java Obfuscator and Android App Optimizer . [Online]. Available: https://www.guardsquare.com/proguard

[51] Y. Duan, M. Zhang, A. V. Bhaskar, H. Yin, X. Pan, T. Li, X. Wang, and X. Wang, "Things you may not know about android (un) packers: A systematic study based on whole-system emulation." in *NDSS*, 2018.

[52] L. Xue, X. Luo, L. Yu, S. Wang, and D. Wu, "Adaptive unpacking of android apps," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 358–369.

[53] Umeng.com. (2021) Developer Center. [Online]. Available: https://developer.umeng.com/docs/143070/detail/145892

[54] MvnRepository. (2021) Maven Repository: org.smali | dexlib2. [Online]. Available: https://mvnrepository.com/artifact/org.smali/dexlib2

[55] China Telecom. (2021) Signed Agents. [Online]. Available: http://id.189.cn/contractUs

[56] Android Developers. (2021) Android Debug Bridge (adb). [Online]. Available: https://developer.android.com/studio/command-line/adb

[57] Tencent Cloud. (2021) Number Verification Service (NVS). [Online]. Available: https://cloud.tencent.com/product/nvs?from=14588

[58] Wuhan Douyu Network Technology Co., Ltd. (2021) Douyu TV Mobile Client: HD Game Interactive Video Live Broadcast. [Online]. Available: https://www.douyu.com/client?tab=client#mobile

[59] Chengdu Ledong Information Technology Co., Ltd. (2021) Codoon: Intelligent Sports Platform & Professional Equipment Shopping Guide. [Online]. Available: https://www.codoon.com/

[60] Shanghai Chuanglan Yunzhi Information Technology Co., Ltd. (2021) Chuanglan Shanyan. [Online]. Available: https://shanyan.253.com/

[61] Shenzhen Hexun Huagu Information Technology Co., Ltd. (2021) AURORA Verification. [Online]. Available: https://www.jiguang.cn/en/identify

[62] GEETEST. (2021) geetest-product overview. [Online]. Available: https://docs.geetest.com/onelogin/overview/prodes/

[63] NetEase Company. (2021) Number Verification: NetEase Easy Shield. [Online]. Available: https://dun.163.com/product/phone-verification

[64] ZhangTao Network. (2021) MobTech: Get the Phone Number on Operator's Gateway and Verify the Phone Number in One Second. [Online]. Available: https://www.mob.com/mobService/secverify

[65] Merit Interactive Co.,Ltd. (2021) Getui One-Tap Authentication: Passwordless Login. [Online]. Available: https://www.getui.com/verification

[66] Shareinstall.com. (2021) Shareinstall: Passwordless Login with Mobile Phone Number . [Online]. Available: http://www.shareinstall.com.cn/one-click-login.html

[67] SUBMAIL. (2021) [Passwordless Login] One-Tap Login, Local Authentication, Safe and Efficient. [Online]. Available: https://www.mysubmail.com/onepass

[68] Xuanwu Tech. (2021) Jixin Cloud Communication: One-Click Login. [Online]. Available: https://www.139130.com/productsandservices/info.aspx?itemid=14

[69] Beijing Emay Technology Co., Ltd. (2021) Emay: One-Tap Login. [Online]. Available: https://www.emay.cn/article957.html

[70] Changzhou Qianfan Network Technology Co., Ltd. (2021) Qianfan Cloud. [Online]. Available: http://www.qianfanyun.com/help/1270

[71] Baidu. (2021) Baidu AI Cloud Document - Phone Number Verification Service (PNVS): One-Click Login Process. [Online]. Available: https://cloud.baidu.com/doc/PNVS/s/gkc36e0aj

[72] Hangzhou Youpaiyun Technology Co., Ltd. (2021) Youpai Cloud: One-Click Login. [Online]. Available: https://www.upyun.com/products/one-click-login

[73] Santi Cloud. (2021) Santi Cloud Communication Platform: One-Tap Login, Passwordless Login, Local Number Login. [Online]. Available: https://www.santiyun.com/fastCheck.html

[74] LongXinTong. (2021) One-Tap Login. [Online]. Available: http://www.hiht.cn/yjdl.html

[75] Weiwang Liantong. (2021) One-Key Authentication. [Online]. Available: https://www.lmobile.cn/ch/yjrz.html

[76] DCloud. (2021) Official Website of Uni One-Click Login App. [Online]. Available: https://uniapp.dcloud.net.cn/univerify

[77] China Internet Network Information Center. (2021) The 48th Statistical Report on China's Internet Development. [Online]. Available: http://www.cnnic.cn/hlwfzyj/hlwxzbg/hlwtjbg/202109/P020210915523670981527.pdf

[78] ESurfing. (2021) Android Client of ESurfing Cloud Disk. [Online]. Available: https://cloud.189.cn/web/static/download-client/index.html

[79] China Telecom. (2021) Esurfing Account Open Platform: Password-free Login. [Online]. Available: http://id.189.cn/banner/unPassword

[80] A. Bianchi, E. Gustafson, Y. Fratantonio, C. Kruegel, and G. Vigna, "Exploitation and mitigation of authentication schemes based on device-public information," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 16–27.

[81] H. Wang, Y. Zhang, J. Li, H. Liu, W. Yang, B. Li, and D. Gu, "Vulnerability assessment of oauth implementations in android applications," in *Proceedings of the 31st annual computer security applications conference*, 2015, pp. 61–70.

[82] H. Wang, Y. Zhang, J. Li, and D. Gu, "The achilles heel of oauth: a multi-platform study of oauth-based authentication," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 167–176.

[83] DENIS MASLENNIKOV. (2011) ZeuS-in-the-Mobile: Facts and Theories. [Online]. Available: https://securelist.com/zeus-in-the-mobile-facts-and-theories/36424/

[84] Trusteer. (2012) The Song Remains the Same: Man in the Mobile Attacks Single out Android. [Online]. Available: https://www.globalsecuritymag.fr/The-Song-Remains-the-Same-Man-in,20120710,31306.html

[85] F-Secure Labs. (2021) Trojan:Android/Crusewind. [Online]. Available: https://www.f-secure.com/v-descs/trojan_android_crusewind.shtml

[86] W. Enck, P. Traynor, P. McDaniel, and T. La Porta, "Exploiting Open Functionality in SMS-capable Cellular Networks," in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005, pp. 393–404.

[87] N. Golde, "SMS Vulnerability on Feature Phones," Ph.D. dissertation, Master Thesis, Berlin Institute of Technology, 2011.

[88] A. Coletta, G. Maselli, M. Piva, D. Silvestri, and F. Restuccia, "My sim is leaking my data: Exposing self-login privacy breaches in smartphones," *arXiv preprint arXiv:2003.08458*, 2020.