# Maginot Line: Assessing a New Cross-app Threat to PII-as-Factor Authentication in Chinese Mobile Apps

Fannv He*, Yan Jia†, Jiayu Zhao*, Yue Fang*, Jice Wang*, Mengyue Feng*, Peng Liu‡, and Yuqing Zhang*§‖¶
*National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, China
†DISSec, College of Cyber Science, Nankai University, China
‡College of Information Sciences and Technology, Pennsylvania State University, USA
§Hangzhou Institute of Technology & School of Cyber Engineering, Xidian University, China
‖School of Cyberspace Security, Hainan University, China

***Abstract*—Authentication is one of the established practices to ensure user security. Personally identifiable information (PII), such as national identity card number (ID number) and bank card number, is used widely in China's mobile apps as an additional secret to authenticate users, i.e., *PII-as-Factor Authentication* (*PaFA*). In this paper, we found a new threat that calls on the cautiousness of *PaFA*: the simultaneous usages and business-related interactions of apps make the authentication strength of a target app weaker than designed. An adversary, who knows fewer authentication factors (only SMS OTP) than a *PaFA* system required, can break the authentication by gathering information or abusing cross-app authorization from other apps. To systematically study the potential risks, we proposed a semi-automatic system, *MAGGIE*, to evaluate the security of *PaFA* in target apps. By measuring 234 real-world apps in Chinese app markets with the help of *MAGGIE*, we found 75.4% of apps that deployed *PaFA* can be bypassed, including the popular and sensitive ones (e.g., AliPay, WeChat, UnionPay), leading to severe consequences like hijack user accounts and making unauthorized purchases. Additionally, we conducted a survey to demonstrate the practical implications of the new risk on users. Finally, we reported our findings to the vendors and provided several mitigation measures.**

## I. Introduction

Mobile applications (Apps) today support some most popular daily tasks like payment, financial transactions, and email access. In 2022, China had 1.04 billion smartphone users, accounting for 15% of the world's users [1]. A report revealed that the average Chinese smartphone user installed 73 apps on their phones and mobile app usage stood at 7.6 hours per day [2]. The boost of apps results in the simultaneous usage of multiple apps. Authentication, as one of the most important established practices to protect sensitive operations (e.g., login and payment), identifies the users by authentication factors. Besides passwords, apps also leverage other authentication factors to verify identity, including short messaging service one-time passwords (SMS OTP), personally identifiable information (PII), security questions, biometric data, etc. Among these authentication factors, PII, such as national identity card number (ID number), is commonly used as an additional secret to authenticate users. This type of authentication method is called *PII-as-Factor Authentication* (*PaFA*) in this paper. For example, two popular payment apps, `UnionPay` and `AliPay`, demand a user's bank card number, full name, ID number, and SMS OTP to meet the authentication requirements of payment passwords recovery. That is, app vendors deploy additional *PaFA* mechanisms in order to gain higher **authentication strength**. However, whether the *PaFA* mechanism in modern apps effectively provides the intended level of authentication strength in real-world scenarios remains not clear.

As being observed in recent years, in China, PII is widely gathered and stored by mobile app vendors. Many apps require users to provide relevant PII when providing services, such as real-name information for booking air tickets and bank card numbers for receiving provident funds. Meanwhile, many users use multiple apps developed independently by various vendors with different security considerations. The authentication requirements of apps with no highly sensitive functions (like `Kindle` app for reading e-books) are generally less restrictive compared with apps for online banking. In our observation, the fact that different apps have different authentication strengths could *provide the attacker with an opportunity* to launch **cross-app attacks** in which the PII obtained by breaking the authentication mechanism of one app with lower authentication strength could be exploited as an authentication factor to break the authentication mechanism of another app with higher authentication strength. Besides this opportunity, we observe that the attacker could enjoy *a second opportunity* by abusing the **cross-app authorization** mechanism of an app with lower authentication strength. Therefore, given an app, its authentication system could be potentially weakened by other apps, which means attackers could leverage the weaknesses of business partners' apps to threaten the business of the target.

Existing works have examined the *personal security questions* (e.g.,favorite food, father's middle name and ZIP code) for account recovery authentication and found that the answers are vulnerable to random guessing, social networking sites and potent sources (such as club membership rosters) searching, acquaintances [3], and statistical attacks [4]. However, infor-

¶ Yuqing Zhang is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24241
www.ndss-symposium.org

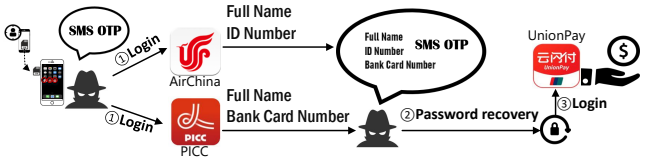


Fig. 1. An attack path for `UnionPay`.

mation required by the *PaFA* in Chinese apps nowadays is of a totally different nature: it is generally infeasible to guess and would not be intentionally made public by users or shared among other people. In contrast, this study found **a new threat** that reveals an unexpected security weakness of *PaFA*: the simultaneous usages and business-related interactions among apps make the **authentication strength** of a target app weaker than designed.

**A motivating case.** Here, we provide a manually found case. An attacker, who can only access SMS OTP (e.g., direct contact with SIM cards by frequent mobile phone theft and loss [5–10]), bypasses the strong *PaFA* of a payment app (`UnionPay`) that requires bank card number, full name, ID number, and SMS OTP, and successfully transfers money from the victim's bank account.

Firstly, by accessing SMS OTP, the attacker can log in to the victim's accounts on several apps with lower authentication strength and gather useful PII, although he cannot log in to the `UnionPay` app. Specifically, the attacker can find the victim's full name and ID number on the UI where users book flights in the `AirChina` app and find bank card numbers on the screen where users purchase insurance in the `PICC` (People's Insurance Company of China) app. With this PII, the attacker can reset the victim's login and payment passwords of the `UnionPay` app by abusing the password recovery process. Finally, with the two passwords, the attacker can transfer money from the victim's `UnionPay` account. Fig. 1 shows the **attack path** and the attacker's operations.

**Method.** In order to systematically study the potential risks when users use multiple apps simultaneously, we proposed a semi-automatic system, called *MAGGIE*, to check whether the **authentication strength** of a given app can be weakened by other apps. Our idea is to act as the adversary, who only knows incomplete and inadequate authentication factors (e.g., only SMS OTP) of the victim, to operate other apps (through impersonating the victim) for breaking the target authentication. The challenge here is how to automatically identify the security risk and efficiently find all needed operations to achieve the attack, given millions of possible operations on hundreds of apps. For this purpose, we proposed a finite state model to describe the combined use process of multiple apps and leverage model checking to effectively find operation sequences that the attacker can follow to break the target app. The model starts with a set of "weak authentication" apps to obtain some seed PII and then uses the seed PII to achieve the **snowball effect** until the adversary cannot obtain additional PII or has tried all the potentially relevant apps. Moreover, we modeled cross-app authorization rules. The attacker could leverage the apps with weak authentication strength as the pedals to compromise the target app business. To ease human effort, we implemented *XHelper* module to automatically gather the PII shown on mobile apps' screens and extract the relevant authentication and authorization rules, which helps us build the finite state model from hundreds of apps. Additionally, to assess the actual risk on real users, we conducted a survey by distributing 281 questionnaires to the general public to determine what apps they had installed and used.

**Findings.** After analyzing 234 real-world apps (evenly distributed across types) from Chinese app markets using *MAGGIE*, we found *PaFA* is widely deployed (by 65 apps, 27.8%) and PII is ubiquitous in various apps that are developed to process data/information with different degrees of sensitivity. More importantly, we demonstrated that 49 apps (out of 65) using *PaFA* are less secure than designed when users use multiple popular apps simultaneously. With fewer authentication factors than the app designed, an attacker can Bypass Authentication by utilizing Cross-App Information gathering and Business relationships (*Bypass Authentication by Cross App Exploitation, Bacae* attack). For instance, `AliPay` requires four authentication factors (SMS OTP, bank card number, full name, and ID number) to login. However, through the authorization of `Taobao`, an attacker can log in `AliPay` with only two factors (SMS OTP and ID number). This means that the attacker can bypass the login authentication without knowing all the authentication factors required during a normal login process. Another example is that ID number and user's full name, the authentication factors used by `Etax` for login, are exposed by 154 (65.8%) apps (e.g., after logging into `Qunaer`, an attacker can view the ID number and full name in historical order.). Statistically, powered by *MAGGIE*, our measurement study of 234 popular mobile apps reveals that 95.7% of these apps display PII on one or more UI pages. Additionally, out of the total number of apps, 27.8% of them employ *PaFA* mechanism. Among these apps, a staggering 75.4% of them are susceptible to *Bacae* attack, which can result in severe consequences like account hijacking, unauthorized purchases, and fraudulent activities, etc. Note that in our measurement, the apps are distributed evenly across all types. For apps simultaneously used by a real user, our survey results suggest that the *Bacae* attack could have more severe consequences: we observed all users had used one or more apps with *PaFA* and 94.2% of users were subject to at least one attack path.

**Contributions.** In summary, we make the main contributions as follows:

- *New findings*. We conducted the first systematic empirical study on the *PaFA* mechanisms of highly popular apps in the scenario where the victim simultaneously uses many apps. Our research reveals that multiple apps may have serious security implications on other apps, resulting in authentication strength of *PaFA* mechanisms equal to SMS OTP only in most cases. Vendors can learn from our new understanding to better communicate with each other and improve their authentication mechanisms.

- *Model and semi-automatic tool.* We proposed a model that described the combined usage process of multiple apps as a state transition system and defined a security property for the model checker to identify attack paths that can be

exploited to bypass the authentication of a target app with fewer authentication factors. Based on the model, we designed and implemented a tool, *MAGGIE*, to investigate whether the authentication strength of a given app can be weakened by other apps if a victim user is using them at the same time.

**Ethical Considerations.** The *Bacae* attack related experiments were conducted using phone numbers and app accounts of the authors, hence did not affect other users. For security reasons, we have hidden the version number of apps in the paper.

## II. Background and Threat Model

### A. Authentication Factors in App

Mobile apps extensively utilize SMS one-time passwords (OTP), user-chosen passwords, and personally identifiable information (PII) to verify users.

**User-chosen password.** User-chosen password is the most commonly used authentication method. The app user is required to enter the preset password that matches the account ID. Sometimes, apps may require users to set additional passwords for protecting sensitive operations like payment. In addition, when the user forgets the password, almost all apps provide a function to reset the password.

**SMS OTP.** SMS OTP is another widely used authentication factor by mobile apps. Users are required to enter the SMS OTP which is sent to the corresponding phone number bound with the account. At present, many apps use SMS OTP as an alternative authentication way instead of the complex login password. This mechanism gives convenience for users by avoiding setting up and remembering static passwords while bringing more security risks to users. Previous studies showed that attackers could obtain SMS verification codes from various channels, such as SIM swaps [11], installing malicious app. Since such malicious apps only need READ_SMS and SEND_SMS permissions [12], it is a fairly practical attack to steal SMS OTP.

**PII-as-factor authentication.** In addition to the above mechanisms, the PII of the account owner, (such as ID number, bank card number, birthday, and name) is also often used to authenticate users, especially for password recovery. For example, before providing the retrieve account service, `Alipay` first prompted users to enter their name and ID number to find a linked account, and then to enter a bank card number for identity verification. This kind of authentication usually is used together with other authentication methods, aiming to provide additional security protection. In this paper, we call the authentication schemes that use PII as *PII-as-Factor Authentication* (*PaFA*).

### B. Authentication and Authorization in Apps

Real-world mobile apps consist of complex function logic involving multiple parties and authentication mechanisms. Fig. 2 typically depicts how a third-party payment app authenticates users and interacts with other apps.

First of all, ① the app user registers an account by creating an account ID with a password and providing additional information such as a phone number and ID number. Note

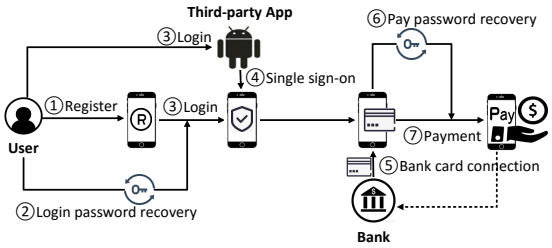


Fig. 2. An example of authentication and authorization in apps.

that, phone numbers and email addresses are often chosen as usernames. ③ Afterwards, the user can log into the app by entering their password or an SMS-OTP. In case the user forgets the password, the app provides the password recovery function: ② a user who answers some specific questions (e.g., SMS OTP, ID number, or bank card number) can reset the password. Instead of creating a totally new account, ④ a user can use the account of another app to log in via a single sign-on (SSO) process. For example, a user can sign in to `Alipay` using his/her `Taobao` account. After login, ⑤ the user needs to link a bank card to the app through authorization and authentication of the bank before making the payment. This process connects the bank payment business to the third-party payment app. Additionally, the user can set a payment password to protect sensitive operations, which would be asked for and verified by the app before making a payment. ⑥ The app also designs corresponding payment password recovery processes. With the payment password, ⑦ the user can achieve the goal of money transfer.

Each scenario in the whole process requires the user to accomplish different types of authentication. In summary, modern mobile apps often involve various authentication mechanisms and complex interactions with multiple third-party businesses, which brings potential security risks.

### C. Threat Model and Scope

Assuming that the victim, Alice, is a normal user. She has installed numerous popular apps from official sources, created accounts, and provided real personal information to these apps. As a result of Alice's usage, these app accounts contain her usage histories (e.g., air ticket booking records). According to Alice's/vendors' security expectation, an attacker who possesses insufficient authentication factors (e.g., only SMS OTP) should NOT be able to complete the password recovery process or authentication process for a target app.

The attacker, Mallory, can only obtain the victim's SMS OTP. This can be realized in various ways, e.g., mobile phone theft, OTP fraud and underground (cyber-crime) markets. Our threat model is realistic. First, in many regions, the SIM card in a mobile phone can be physically removed and inserted into another device, enabling the attacker to receive SMS without unlocking the former phone screen. Mobile phone theft and loss are frequent occurrences in many areas worldwide [5–10], for example, one in ten U.S. smartphone owners are victims of phone theft [9]. Second, attackers can also obtain SMS OTP by OTP fraud. Reports of OTP fraud have surfaced in various countries [13–15]. For example, over 2,000 cases of OTP fraud

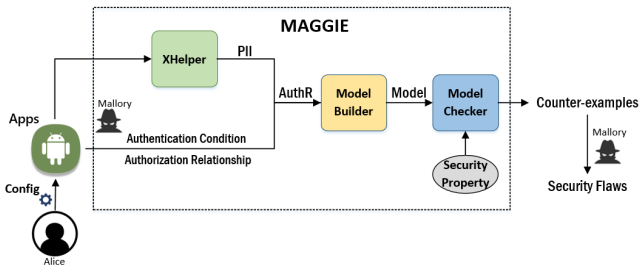


Fig. 3. The architecture and workflow of *MAGGIE*.

took place in India in 2021 [14]. Thirdly, there are instances where the obtained OTPs were on sale in underground (cybercrime) markets, for instance, customers' smartphone numbers and OTPs were sold by (corrupted) telecom operator staff in China [16][17]. These evidences clearly indicate that real-world criminals have already been utilizing stolen OTPs.

Mallory aims to bypass or compromise the authentication mechanisms of the target app in order to achieve illegal goals, e.g., hijacking Alice's accounts, gathering more PII of Alice, making purchases with Alice's funds, and taking out loans in Alice's name, etc. Note that, Mallory does not need to enumerate all the victim's installed apps. Instead, he can simply attempt to recover passwords for popular apps which the victim is likely to use. In some cases, Mallory knows little about Alice but he knows that `UnionPay` is very popular so there is a fair chance for Alice to use it. In addition, the target apps do not have any implementation-level vulnerabilities. We do not make any specific assumptions on the risk control deployed by vendors, as it is a black box and operates on a probabilistic basis. The partial impact of risk control is discussed in Section V-E.

## III. Design and Implementation of *MAGGIE*

In this section, we elaborate on the design and implementation of *MAGGIE*. As mentioned earlier, the attacker's goal is to gain additional authentication factors and later break the *PaFA* of a target app. He would try to leverage the limited authentication factors he knows to operate other apps on behalf of the victim. Given millions of possible orders in which an attacker could operate these apps, the challenge here is how to efficiently find **attack paths** that specify the operations necessary for completing the attack within such a large set of possibilities.

Our approach is building a state machine model that describes the adversary who simultaneously operates hundreds of apps and leverages model checking to find **attack paths** that gain additional authentication factors or find vulnerable cross-app authorizations towards breaking the target *PaFA*. To extract the model from real-world Android apps, we implemented an auxiliary tool *XHelper* that gathers the PII displayed in mobile apps and assists a human analyst in extracting the authentication requirements.

### A. Overview

To achieve our goal, we model the combined usage process of multiple apps as a state transition system and utilize a model checker to verify whether pre-defined security properties of target apps hold in the model. The counter-examples reported by the model checker demonstrate the attack paths and are verified manually.

**Architecture.** *MAGGIE* includes three core modules: an auxiliary tool *XHelper* that explores the UI to help gather available PII in apps, a *Model Builder* that generates the state machine model specified in NuSMV language, and a *Model Checker* that analyzes the model with security property we proposed and outputs counter-examples (attack paths), as outlined in Fig. 3. The apps considered in *MAGGIE* have already been used by the victim, as described in the threat model. With the output of *XHelper* and manually extracted authentication conditions and authorization partners, we get the Authentication&Reward (**AuthR**) metadata (defined in Section III-B) of each app, which then are used by the *Model Builder* to generate the specific code of the model. Based on the user-specified security property, the *Model Checker* would generate counter-examples if they exist. These counter-examples are further verified manually.

### B. Extracting Authentication&Reward

**Authentication&Reward (AuthR)** is 4-tuple metadata, which is composed of operations, SUCCESS condition, authorization, and reward. The format of **AuthR** is as follows. It describes the authentication requirements of an app operation and the PII (i.e., the reward) that could be obtained after authentication is completed.

**AuthR::=(App.Op, Cond, Authorz, Reward)**

**App.Op** refers to the operations that can be performed within an app. In our model, an app involves a series of operations that require authentication, such as login, payment, or bank card connection.

**Cond** is the set of authentication factors, indicating the SUCCESS condition of passing the *app.op*'s authentication. Note that **Cond** not only includes direct authentication factors required by the current operation but also includes the factors required by pre-operations. For example, the **Cond** of payment includes not only the payment password but also the password for login.

**Authorz** records third-party operations that have delegated authorization relationship with current *app.op*. For example, `Taobao`'s user account can log in `Alipay` by single sign-on, the *Authorz* record the authorization relationship {Taobao.SSO}.

**Reward** is a set of PII, which refers to PII that can be obtained from the app after completing this *app.op*'s authentication. For example, one can view the connected bank card number after passing the login authentication in `PICC` app. The bank card number should be put into the set *Reward*.

Automatic extraction of the **AuthR.Cond** and **AuthR.Authorz** is difficult because most apps have anti-automation settings such as captchas and security keyboards during authentication process. Luckily, the **Cond** and **Authorz** are easy to collect manually since the pages of authentication are limited and they are clearly displayed in UIs. However, complex apps with many pages, especially those that expose

**Algorithm 1: Exploring UI widgets and Matching PII**

```
launch_App();
current_page ← MainActivity_Page;
page_stack.push(current_page);
while page_stack.size()! = 0 do
    if current_page in page_repository then
        page_stack.pop();
        current_page ← page_stack.top();
        back_to(current_page);
        continue;
    else
        match_PII(current_page);
        page_repository.add(current_page);
    end
    if page_stack.size() < Stack_max_depth and
     current_page.unvisited_widgets.size()!=0 then
        click(random_pop(unvisited_widgets));
        current_page ← get_current_page();
        page_stack.push(current_page);
    else
        page_stack.pop();
        current_page ← page_stack.top();
        back_to(current_page);
    end
end
```

PII in deep paths, make manually extracting the **Reward** (PII displayed on UIs) a big challenge. Specifically, user-configured PII in apps is usually displayed differently on UIs with how it is initially input. For example, a configured ID number may be partially obscured (e.g., 123******4567) or not displayed at all on the configuration page, but be fully exposed on other pages. Meanwhile, apps may sometimes display more PII than a user has configured manually. For instance, an ID number may be obtained from other third parties through authorization. Thus, it is necessary to traverse apps pages to find potential PII leaks, but the process of manual analysis is tedious and may involve random negligent mistakes. To address this challenge, we developed an auxiliary tool *XHelper* to efficiently detect potential PII leaks within apps, reducing the difficulties of manually traversing numerous pages.

**XHelper.** *XHelper* automatically traverses app UIs based on depth-first strategy and gathers PII displayed with regular expressions. The design of the two components is illustrated below.

• *Depth-first UI exploration.* As shown in the Algorithm 1, *XHelper* launches the app and starts the exploration from the `MainActivity` page. It does depth-first exploration on all clickable widgets. The exploration process is maintained in *page_stack*. Pages that need to be explored are pushed into the stack. Once all clickable widgets on a page have been visited, the page is popped from the stack, indicating that the page's exploration is complete. Thus, the `MainActivity` always stays at the bottom of the stack, and its popup indicates the end of the whole exploration. To improve search efficiency, we empirically set the maximum depth of the stack to 4. Increasing the depth to 5 dramatically doubles the test duration of an app, but barely increases the number of PII detected. To avoid *XHelper* being trapped in a loop, we designed a *Page Repository* to store UI pages.

Specifically, we stored the structure and content information of the explored pages and proposed a Tree Matching algorithm to compare this metadata with that of new pages. The algorithm starts from the root nodes of two trees and then recursively compares the attribute values of each node, which are related to the structure, function, content, and so on. Finally, a numerical value is computed to represent the similarity of the two trees. If a new page has a high similarity in structure or content with a certain page in Page Repository, it will be popped out without exploration. For example, shopping apps like `Amazon Shopping` contain countless pages of commodities with similar structures. The similarity comparison help avoids repeatedly visiting these pages.

In addition, we set a *blocklist* to make some widgets not be clicked by *XHelper*, such as "Logout", "Upload photograph", and "account deletion". Because these operations only involve deleting or uploading information, they do not provide new PII for the attacker. Also, we empirically assign higher searching priority to certain widgets, such as "Personal center", "Order list" and "Mine", etc. During the exploration, if such widgets exist on a page, *XHelper* will continue to explore until no such widgets are found, even if the maximum stack depth is reached.

• *Matching PII.* Before inputting the app into *XHelper*, a tester (acting as Alice) uses the app and sets up the account information. The recorded information is then used for further comparison with the text in the app. *XHelper* collects widget texts from the UI page and first applies regular expressions to filter out potential PII. We created 24 regular expressions that can classify texts into 9 types of PII. For example, expression ["(z.+@test.com—.+n@test.com)"] would filter out texts which likely is "z****n@test.com". Next, these suspected PII displayed on the UI page will be further matched with the stored tester's information byte-by-byte.

Based on our observations within the dataset, these 9 specific types of PII have generally proven to fulfill the authentication factors established by these apps. This determination is based on the specific requirements and practices within the Chinese apps ecosystem. For instance, in China, it is common for the ID number to serve as both a national identification number and a driving license number. It is important to acknowledge that the sufficiency of PII can vary depending on specific circumstances, cultural norms, and legal frameworks in different countries and regions. However, our study conducted in the Chinese app ecosystem can provide valuable guidance and serve as a reference for research in other countries and regions.

### C. Formal Modeling with Model Builder

**The state machine model.** We model the usage process of multiple apps by an attacker $\mathcal{A}$ as a state machine, $M = (S, O, T, s_0, s_s)$. $S$ is the set of states. Among them, $s_0$ ($s_0 \in S$) is the initial state where the attacker starts exploring apps. He only has inadequate authentication factors, as illustrated in the threat model. $s_s$ ($s_s \in S$) is the SUCCESS state where the attacker has achieved his goals. $O$ is a finite set of operations on apps. $T$ is a transition function indicating the operations of apps that drive the system to transit from one state to the next.

**Def. 1. State.** A state $s_j\,(s_j \in S)$ has two variables: $s_j = \{FV, app.op\}$. *FV* is a set of authentication factors, indicating the attacker's knowledge on authentication factors. $FV = \{f_0, f_1, ..., f_i\}$, where $f_i$ stands for a certain authentication factor, such as phone number, SMS OTP, and ID number. The *app.op* records the operation performed by the attacker, which would help generate the access paths (see Def 4.). Note that in each state, we only record the last operation performed by the attacker. The initial state $s_0 = \{FV_{init}, null\}$, where $FV_{init}$ holds the attacker's initial ability.

**Def. 2. Operation.** The definition of Operation is the same as **App.Op** in **AuthR**. An operation *app.op* ($app.op \in O$) performed by the attacker would cause a state transition. At state $s_i$, the attacker can take one of any available operations associated with any apps (e.g., the *app.op* in $AuthR_n(app.op, Cond, Authorz, Reward)$), if the attacker holds adequate authentication factors to pass the authentication of $AuthR_n.app.op$ or the third party operation ($AuthR_m.app.op$) related to the $AuthR_n.app.op$.

$$s_i.FV \supset AuthR_n.Cond$$

or

$$\begin{cases} s_i.FV \supset AuthR_m.Cond \\ AuthR_m.app.op \in AuthR_n.Authorz \end{cases}$$

The operations would expand the $FV$ in $s_i$ (authentication factors known by the attacker).

$$\begin{cases} s_j.FV := s_i.FV \cup AuthR_n.Reward \\ s_j = \{FV, AuthR_n.app.op\} \end{cases}$$

As mentioned in Section III-B, set $O$ and corresponding **AuthRs** are constructed with the assistance of *XHelper*.

**Def. 3. Transition:** $T : S \times O \rightarrow S$ is a function that drives the transition from one state to another. For example, $T(s_i, app_x.login) = s_j$ indicates that the "login" operation of $app_x$ drives the system from state $s_i$ to $s_j$.

**Model builder.** Our *model builder* models the continuous use of multiple apps and generates the model specified using the NuSMV language. More specifically, the states are determined by the attacker's knowledge, and the state transitions are determined by operations the attacker can perform. However, manually modeling authentication operations for 234 apps is a complex task requiring substantial human effort. To address this, we designed a model generator to create the NuSMV model automatically. The generator takes necessary modeling information and uses it to customize each operation in the template code. The template code includes the state transitions, the attacker's knowledge, and the operation code framework.

Considering the efficiency of the model, we do the following optimizations: ① We merged apps with the same login *AuthR.Cond* and *AuthR.Reward* together, creating an *AppCluster*. Each *AppCluster* was treated as a single node on the access path, potentially representing multiple apps. As a result, the number of nodes on the path during model checking was significantly reduced. ② We pruned some insignificant authentication options by removing redundant paths. For instance, if the authentication factors required for the first login option are fully included in the second login option, we pruned the second path and kept the first one. ③ We removed meaningless state transitions, such as those where the authentication factor(s) (i.e., **Reward**) obtained after login are a subset of the authentication factors required for login authentication.

### *D. Detecting Security Flaws with a Model Checker.*

With the generated model, we leverage an off-the-shelf model checker, NuSMV, to verify the models against a generalized security property, which is elaborated as follows.

**Def. 4. Access path:** An access path from $s_i$ to $s_j$ is an ordered sequence of operations $(op_1, op_2, ..., op_n)$, such sequence of operations brings a series of transitions $T\,(s_i, op_1, op_2, ..., op_n) = s_j$, along which $s_i$ can reach $s_j$.

Intuitively, an access path indicates how an attacker is able to extend his authentication factors from $s_i$ to $s_j$.

**Def. 5. Security property:** Given a targeted operation ($AuthR_t.app.op$) of an app and an attacker $\mathcal{A}$ with $FV$, there should not be an access path from $s_0$ to $s_i$ that allows the attacker to expand his $FV$ to pass the authentication conditions of targeted operation and perform the sensitive operation.

The property can be specified as[2]:

$$\text{AG}^1\ \ (\mathcal{A}.FV \cap AuthR_t.Cond) < AuthR_t.Cond$$

$FV$ in $s_0$ and $AuthR_t.app.op$ are specified by the user of *MAGGIE*. For example, if we consider an attacker who can steal SMS OTP, the $s_0.FV$ should be {SMS OTP}. With respect to the property, our model checker reports a counterexample (security property violation) if it can find an access path that allows the attacker to operate other apps (through impersonating the victim) for gaining enough PII and bypassing the target *PaFA*. We called the counterexample an *attack path*.

**Attack path optimization.** When the model checker searches for a path, if *app.op* can expand the attacker's knowledge $FV$, *app.op* will drive the state transition and be stored in the access path. A transition is considered redundant if it expands an authentication factor that is neither the authentication factor required by the target $AuthR.Cond$ nor the authentication factor for the subsequent $AuthR.Cond$. However, it is difficult to distinguish redundant transitions until the model reaches the final state, as it depends on whether subsequent state transitions use the factors obtained in the transition. After obtaining the final access paths, we filter out paths with such redundant state transitions and obtain optimized attack paths.

### *E. Implementation*

In this section, we provide the relevant implementation details of *MAGGIE*. The full source code is released online [18].

We utilized the Python library *uiautomator2* to retrieve and interpret the App UI page. *uiautomator2* encapsulates `UIAutomator` [19] commands as *http* interfaces. It exposes the interface to Python programs by running an *http rpc* server

[2]"AG p" is a CTL (Computation Tree Logic) formula here, which means for all paths (**A**ll), for all states (**G**lobally) along them, p holds.

on the Android device. The *XHelper* functions were developed in *Python 3.8* and executed on *Windows 10*. We defined a NuSMV global variable "word[i]" to represent authentication factors, where each bit is either 0 or 1. Each bit in the variable represents one type of PII or portion of PII possessed by the attacker. For example, "1100000" indicates that the authentication factors held by the attacker in the current state are SMS OTP and the first 6 digits of the ID number[3]. The model judged whether an *AuthR.app.op* could be completed through "$FV \,\&\, AuthR.Cond$". If the model passed the *AuthR.app.op*, *FV* was expanded through "$FV|AuthR.Reward$", and the state transition was driven. We use a Python script to read apps' **AuthR** and automatically generate state transition code.

We described the security property through Computation Tree Logic (CTL). The security property was verified when we executed the NuSMV code: the model state variable *FV* can reach the expected state through $x$ different paths. Let $X$ be the maximum value of $x$ that can satisfy the property, that is, the number of attack paths we found. We first initialized $x = 1$, then expanded the value of $x$ exponentially, and continued to execute NuSMV code until the properties were not satisfied. Remember that the attempted value of $x$ at this time is $x_i$, and the last attempted value of $x$ is $x_{i-1}$. Therefore, it can be determined that $x_{i-1} \leq x < x_i$. Then, we tried $X$ by the dichotomy in the interval of $[x_{i-1}, x_i)$, and found the maximum value of $x$ that satisfied the property, that is, the number of attack paths.

We deployed NuSMV on Ubuntu 18.04 running in a VMware Workstation with configuration of 8-core and 16GB RAM to execute the model. We also developed some scripts for automatically running the model, which took approximately 8 hours to complete all the tasks.

**Evaluation of *XHelper*.** To evaluate the accuracy of *XHelper*, we randomly selected two apps from each of the 39 categories in our dataset (Section IV-A), for a total of 78 apps, and manually configured PII for each app. Two researchers independently explored each app's UIs for at least 30 minutes to identify any available PII, which minimizes careless omission. In total, this process took 83 manhours and got 362 instances of PII as the *ground truth*. Next, we used *XHelper* to extract PII from these apps and compared the identified PIIs with *ground truth*. The result shows that *XHelper* achieves a precision of 100% and a recall of 87.6% (317 out of 362). By manually inspecting the 45 missed PIIs, we found that 18 of them were located on WebView pages, since *XHelper* inherits the limitations of `UIAutomator` which is incapable of identifying WebView elements; 12 missed PIIs were caused by the exploration depth limitation set for *XHelper* (as illustrated in Section III-B); 9 PIIs were missed because loading times of these UI pages were too long and skipped by *XHelper*; 6 missed PIIs appeared at the bottom of the overlong pages and *XHelper* gave up to endless scrolling down such pages.

**Example.** Here we take a real-world case (Fig. 1) as an example to describe how *MAGGIE* works roughly. At beginning, the adversary only has the ability to receive SMS OTP. $s_0 = \{sms, null\}$. The target is the payment operation

[3]A Chinese ID number can be divided into three pieces of information - the first 6 digits, the middle 8 digits (representing the birthday), and the last 4 digits.

TABLE I. SOME **AUTHRS** IN THE EXAMPLE IN FIG 1.

| |
|---|
| $AuthR(AirChina.login, \{sms\}, \emptyset, \{fn, id\}^1)$ |
| $AuthR(PICC.login, \{sms\}, \emptyset, \{fn, bcn\}^1)$, |
| $AuthR(UnionPay.LPwdRecovery^2, \{sms, fn, id, bcn\}, \emptyset, \emptyset)$ |
| $AuthR(UnionPay.PPwdRecovery^2, \{sms, fn, id, bcn\}, \emptyset, \emptyset)$ |

[1] fn: full name, id: ID number, bcn: bank card number.
[2] LPwd: Login password, PPwd: Payment password.

of app `UnionPay` (UnionPay.pay). Given the three apps (`AirChina`, `PICC`, `UnionPay`), with the help of *XHelper*, we extract at least 4 **AuthRs**, as shown in Tab. I. More specifically, we login `AirChina` and `PICC` via SMS OTP. Next, after logging in, *XHelper* gathers the PII available: full name and ID number from `AirChina`, and full name and bank card number from PICC.

With **AuthRs**, the *model builder* generated the state machine model, and the *model checker* would find an access path that violates the security property. That is, an attack path can be $T(s_0, AirChina.login, PICC.login, UnionPay.login) = s_j$, $s_j = (\{sms, fn, id, bcn\}, UnionPay.pay)$. This path means an attacker, who only knows SMS OTP, can break the payment authentication of `UnionPay` through logging in `AirChina` and `PICC`.

## IV. EXPERIMENT SETUP

### A. Dataset

We selected 234 high-profile apps from the three most popular app stores - Huawei [20], Vivo [21], and Tencent App Centre [22] - based on their total download numbers in these app stores, as there is no dominant app store in China. To make a comprehensive measurement, we categorized the apps into 39 categories based on their business functions (e.g., map navigation, car-hailing, travel services, house rental, shopping, online payment, instant messaging, etc.), referring to mobile applications classification in the National Standard "GB/T 41391-2022". From each category, we selected top 6 apps with the highest total download numbers, resulting in a total of 234 apps for our test set. All the ranking lists were obtained in June 2022.

### B. Experimental Procedure.

Initially, we manually registered and configured the apps using a researcher's authentic personal information, including her name, phone number, id number, email address, and bank card details. These configurations often require Turing tests, such as graphic verification code. Therefore, this step was performed manually (Step ①). Secondly, we extracted the **AuthRs** of these configured apps with the help of *XHelper*. During the process of *XHelper* exploring the app UIs, we manually assisted it in completing the authentication process when anti-automation mechanisms occurred (e.g., verification codes, face recognition, and security keyboards) (Step ②). Thirdly, we used the *model checker* to obtain the attack paths based on the model generated by the *Model Builder* (Step ③). For every target *app.op*, we output the optimized attack paths from the first 1000. Finally, we manually verified the optimized attack paths using the researchers' own knowledge in the real world (Step ④). In our experiment, we take into account a

common scenario where individuals typically possess a single phone number. In scenarios where users employ multiple phone numbers for different apps, we consider different phone numbers as distinct users, which means there would be false positive attacks.

**Risk control considerations.** Risk control analyzes multiple factors such as device, IP address, location, and user behavior to determine the level of risk associated with an operation (e.g., a login attempt). Based on the risk level, the authentication system may require additional authentication factors or block the login attempt. Although this paper does not cover black-box risk control of apps, we adopted some measures to test its impact since they are deployed in some apps in reality. Specifically, we repeated the experiment from the second step with different devices and IP addresses to compare the results with the first test. Some interesting differences between the two deployments were observed and discussed in Section V-E.

Note that except for the comparative experiment of risk control, the rest of this paper discussed the test results after replacing the device and IP.

## V. Findings and Measurements

With the help of *MAGGIE*, in this section, we report the measurement results of *Bacae* risk. First, we summarize the primary causes leading to *Bacae* attack (i.e., ubiquitous PII in apps and cross-app business partnerships) along with realistic case studies that illustrate the process and consequence of *Bacae* attack. Then, we report the statistics on *PaFA* deployment, business partnerships, PII exposure, and attack opportunities (attack paths).

### A. Root Causes of Bacae Attack

**Ubiquitous PII in apps.** We observed lots of mobile apps collect and store users' PII for their business (e.g., hospital registration, hotel reservation, ticket reservation, and electronic banking.), which can be obtained by logging in to accounts and checking the UIs of apps. Due to the ubiquitous PII in mobile apps, an attacker in our threat model, who only can access the victim's SMS OTP, has opportunities to break a strong *PaFA* system by harvesting PII from other apps with lower authentication strength. That is, PII available in some apps is the authentication factor of some other apps, and the combined use of these apps with different authentication strengths results in *PaFA* becoming weaker.

• *Full exposure.* Many apps directly use PII and display it on normal UI pages, and an attacker can easily obtain these PII by checking UI pages as long as he logs into the victim's account. For example, after logging in to `Ctrip`, an attacker can access the "air ticket booking records" and extract PII from the order list, which may include the user's full name, ID number, phone number, and so on. Moreover, when using the booking service, the app provides the stored PII for the current user. It is convenient for users to double-check the information of passengers when booking flights and tickets. Besides displaying PII for business functions, apps also show PII on the pages like "personal center", which is designed for users to edit their information. More importantly, regional laws may require apps to show what data they have collected from users. Thus, many apps added an entry to the "privacy data collection list" which lists all users' information.

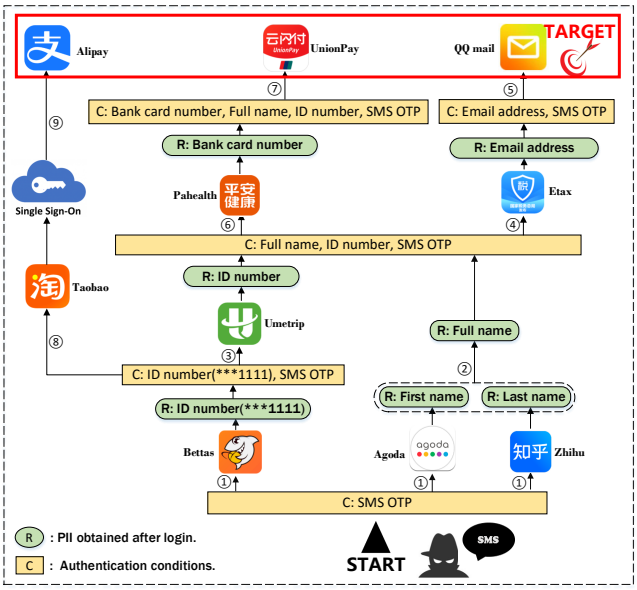


Fig. 4. Breaking *PaFA* by cross-app PII harvesting and business partnerships: a case study.

In addition to activities, some mobile apps also show PII on other external files (e.g., pictures or pdf files). For instance, `Qianzhan`, an app providing financial services, downloaded 510,000 times, displays the user's full name, ID number, and bank card number in the payment authorization agreement document. This pdf file belongs to the account and is stored on the cloud, which is always available after changing the phone.

• *Risky partial exposure.* Some apps mask parts of the information displayed on UIs to avoid exposing full PII. However, because different apps adopted different masking strategies, an attacker could stitch the pieces together into full PII. For instance, with the last name displayed in `Didibike` and the given name displayed in `Anjuke`, an attacker can obtain the full name. Another representative is the ID number. It has 18 digits in total, and the 7th to 14th digits are the individual's birthday. Many apps display the first 6 digits and the last 4 digits of the ID number. E.g., `Airbnb` exposes the first 6 digits, and `MOMO` shows the last 4 digits. Meanwhile, many apps expose the user's birthday, like `Soul` and `Blued`. If a user registers several of them, an attacker could stitch together these pieces of information into the whole ID number. In this case, any of the apps above likely does nothing wrong, but the inconsistent security policies bring realistic risks to their users who use them simultaneously.

**Case study.** Here, we provide a case study that is likely to happen in the real world to illustrate the impact of this cause. As introduced in the threat model, we assume an attacker, who only could access the victim's SMS OTP (as discussed in Section II-C, by SIM card theft, OTP fraud, etc.), aims to transfer money from `UnionPay`, and access to victim's `QQ mail` account. As shown at the top of Fig. 4, the target `UnionPay` app requires full name, ID number, bank card

number, and SMS OTP to transfer money; the target QQ mail app requires the email address and SMS OTP to reset the login password. The attacker starts from three apps that only require SMS OTP for login (Step ①), i.e. Agoda, Zhihu and Bettas, which expose first name, last name, and last four digits of the ID number respectively after the attacker login as the victim. Note that these three apps all were downloaded more than 100 million times. The attacker then stitches the first name and last name together into the full name (Step ②). With the last four digits of the ID number and SMS OTP, the attacker can log into Umetrip to get the full ID number (Step ③). Next, the attacker is able to reset the login password of Etax app and log into it to get the victim's email address (Step ④). So far, with the email address and SMS OTP, the attacker can reset the password of the QQ mail account so as to access the QQ mail (Step ⑤). Meanwhile, the attacker can log into Pahealth to get the bank card number (Step ⑥). With the victim's full name, ID number, and bank card number, the attacker can reset the payment password of UnionPay app and transfer the victim's money (Step ⑦). Notably, each app in the attack path could be replaced by many other apps, which is demonstrated in our measurement results – an attacker has many attack path choices to bypass a target's authentication condition (Section V-B). Thus, the attack case is very likely to occur in real life.

This case study demonstrated that *PaFA* mechanisms, even those deployed in sensitive apps, do not provide the expected security as designed. More exactly, the designers of UnionPay and QQ mail do not expect a user who only knows SMS OTP can reset passwords and transfer money.

**Cross-app business partnership.** Besides directly breaking *PaFA* by providing the PII, cross-app business partnerships also threaten the security of *PaFA* system in mobile apps. An attacker can leverage apps with lower authentication strength that have a business partnership with the target app to bypass its authentication. We observed risks mainly come from two types of connections below.

• *Account sharing.* SSO provides significant convenience for users. For example, with one account of Taobao app, users can log into other apps through an easy authorization process that only requires logging in Taobao. However, when SSO connects with other apps that require higher authentication conditions, one could pass the authentication with fewer authentication factors.

Besides the widely SSO connection among companies, the account system of one big company is usually shared among its multiple products. For example, a Google account can be used in both Gmail and Google Calendar. In this situation, The inconsistent security designs of the services of one company also could potentially jeopardize apps that deploy strong authentication mechanisms. Here we illustrate another example of QQ and WeChat, which are both popular social software of Tencent Company and connect more than 1 billion people worldwide. WeChat provides a way for users to log their account into a new device by requiring an SMS OTP and scanning a QR code from the user's old device. By contrast, resetting the password of QQ only requires an SMS OTP. Meanwhile, WeChat allows users use QQ account and password to log into WeChat, along with sending an SMS verification code to a certain phone number. Thus, an attacker who can send and receive SMS OTP is able to reset the victim's password of QQ for logging in to the associated WeChat account (without the victim's old device).

Moreover, many apps enable users to synchronize their data from SSO providers. This makes the PII exposure unexpectedly, because different apps may not mask shared PII equally. For example, after a user logging in Qunar by Alipay account, Alipay shares the user's ID and name to Qunar. But the ID number masked by in Alipay will be fully displayed on Qunar's UI.

• *Business authorization.* An app can open service interfaces for other vendors, letting them serve as the third party to provide equivalent services through authorizations. For example, banks can grant payment authority to other third-party payment apps (e.g., eBay). However, the different authentication strengths of these cooperative apps may jeopardize apps with higher security requirements. For instance, Bank of China app holds high security: it requires multiple authentication factors (i.e., phone number, SMS OTP, ID number, bank card number, and PIN of the bank card) for login and payment authentication. By contrast, some third-party payment apps require fewer authentication factors than the bank app does. Therefore, by linking the Bank of China debit card to a third-party payment app like UnionPay, BestPay and Hebao, an attacker can pay with that debit card through an app with weaker authentication strength.

Moreover, many banks in China provide the "Add without entering card number" service to apps. Through this service, a user can check and link her bank card by only providing ID number, full name, and SMS OTP for authentication. Thus, there could be a serious attack on new users who use apps that provide such a service, like UnionPay. An attacker, who only knows SMS OTP, can first obtain the victim's ID number and full name by the *Bacae* attack mentioned in Section V-B, then register an account as the victim and link the victim's bank card through this service to UnionPay. We successfully did such a *Bacae* attack on one of the authors' bank cards, who had not registered UnionPay account, and made a payment by her bank card.

**Case study.** Here we provide another case study to illustrate that the combination of causes makes the *Bacae* attack more feasible and complicated. As shown in Fig. 4 (Step ①,⑧,⑨), an attacker, who only can access the victim's SMS OTP, aims to illegally hijack the victim's Alipay account. The target Alipay app requires the phone number, ID number, bank card number, and SMS OTP to log in. The attacker starts from Bettas app and obtains the last four digits of the ID number (Step ①). Next, with the last four digits of the ID number and SMS OTP, the attacker logs into Taobao (Step ⑧). Finally, the attacker logs into Alipay through Taobao SSO (Step ⑨). Obviously, the designers of Alipay do not expect a user who only knows SMS OTP can access an account.

### *B. Statistic Results*

To conclude, according to statistics on 234 apps of various types, a considerable number of apps deploy *PaFA* to protect sensitive operations, but *MAGGIE* found that 75.4% of them

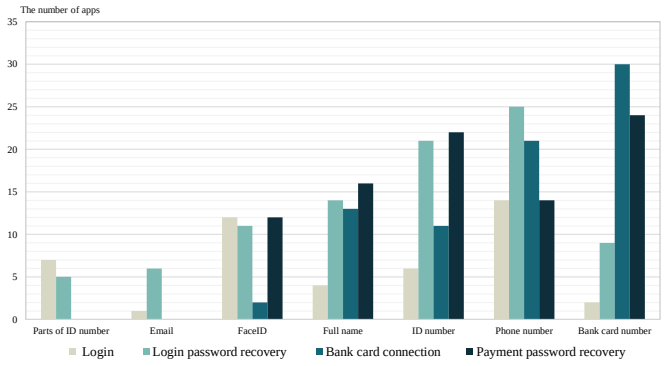


Fig. 5. PII usage in *PaFA* of 65 apps.

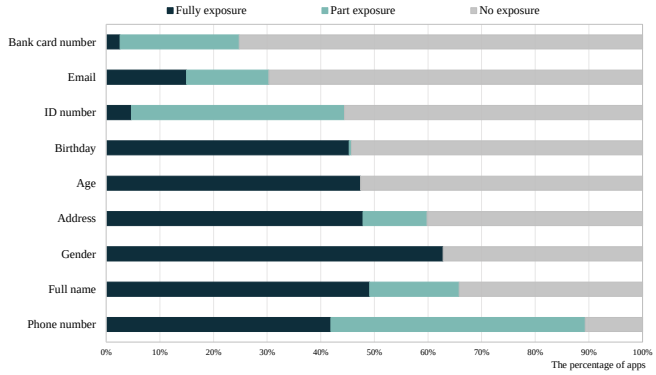


Fig. 6. The percentage of apps that exposed personal data in 234 apps.

are susceptible to *Bacae* if the attacker could obtain SMS OTP. This means that a total of 20.9% of apps in official markets perhaps are impacted. More details are illustrated below.

**PaFA deployment.** Apps using *PaFA* are targets of *Bacae* attack. We observed that 65 out of 234 apps deployed *PaFA*, accounting for 27.8%. These apps mostly belong to sensitive categories such as payment (6 of 6 apps), lending (6/6), shopping (5/6), traveling (5/6), messaging (4/6), etc. This indicates that sensitive apps seek a more secure design by deploying *PaFA*. Specifically, 20 apps employ *PaFA* to protect login, 29 apps for recovering login passwords, 30 apps use *PaFA* for bank card connection, and 27 apps for resetting payment passwords.

The first column of Tab. II shows the specific authentication factor combinations in *PaFA* we observed. Further, we summarized the PII in *PaFA* mechanisms of the 65 apps. As shown in Fig. 5, *PaFA* mechanisms of different operations use PII subtly differently. For login password recovery, some apps may choose PII that is easier to be gathered through *Bacae*, such as certain digits of the ID number. For payment password recovery, apps tend to choose some complex PII, such as a complete ID number and bank card number. However, the lower authentication strength of login enables an attacker easier to gather more PII after logging in as the victim and break a stronger authentication system.

In addition, phone numbers and email addresses are commonly used as usernames in apps. Among the 234 apps we studied, 186 apps use phone numbers and 76 apps use email addresses as username identifiers. Therefore, our threat model is capable of handling most apps. Another observation is that SMS OTP is widely used for app authentication. For apps that deploy *PaFA* mechanisms, SMS OTP combined with PII is a common pattern of authentication condition.

**Business connection.** SSO is a popular login method that allows users to log in with a single ID to multiple related software systems. Among the 234 apps, 172 apps allow users to log in to their systems by using SSO services from third-party apps. Therefore, the login authentication (login password recovery) strength of these 172 apps would be possibly downgraded by the SSO identity providers that they connect with. The mainstream SSO identity providers in China are the following popular apps: `Wechat`, `QQ`, `Weibo`, `Alipay`, and `Taobao`, which are supported by 95% of 172 apps. Finally, we found that the login authentication conditions of 12 apps could be weakened by third-party SSO.

Out of the 234 apps we analyzed, 41 offered third-party payment services. 30 of these third-party payment apps require PII as authentication factors when connecting bank cards. Another 8 apps required payment passwords for linking bank cards, but these passwords could be reset by *Bacae* attack. Consequently, these 38 third-party payment apps could likely be leveraged by *Bacae* attack (as a key node in attack paths) to circumvent the banks' payment authentication of a victim. In contrast, the 6 bank apps we tested required strong authentication factors that cannot be easily bypassed, e.g., bank branch details and PIN.

**PII availability.** PII exposure contributes most to *Bacae* attack. PII is widely exposed in apps and can be easily obtained after user login. 224 (95.7%) apps show PII in one or more UI pages and 86.3% of the apps have a complete display. Among them, only 16 apps require additional verification when a user tries to view PII. As shown in Fig. 6, nearly half of the apps fully exposed PII, such as phone number and full name. ID numbers and bank card numbers are partially displayed in most apps, indicating that some app vendors are aware that some PII like ID numbers and bank card numbers are relatively sensitive. However, using asterisks (*) in replacement of partial PII data in apps is not an efficient method to protect PII gathering when users use many apps with different masking policies simultaneously, as we discovered in Section V-A. In addition, we also observed that apps in different categories tend to expose different kinds of PII. Social apps expose more information about names and birthdays; shopping apps expose addresses more often; travel apps lack the protection of ID numbers; financial management apps are at great risk of exposing bank card numbers. These features of the distribution can help an attacker find an attack path more effectively.

Gaining more PII is a critical step in *Bacae* attack. Therefore, attackers pay special attention to apps that display more PII than is necessary for their login processes. Of all the 234 apps, 193 apps showed additional PII in their UIs post-login. The details are presented in the Appendix Tab. V. With SMS OTP and *Bacae* attack, an attacker could potentially log into 185 apps and gain additional PII.

**Attack paths.** With SMS OTP and *Bacae* attack, the attacker

TABLE II. THE *PaFA* MECHANISM IN APPS AND ATTACK PATHS OBTAINED BY *MAGGIE*

| Authentication conditions[2] | App operations[1] (number of apps) | | | | Attack case | | The number of optimized paths |
|---|---|---|---|---|---|---|---|
| | L | LPR | BCC | PPR | Attack path example | Target App.op | |
| {sms, fn} | | 2 | | | {sms}->Huolala.L{sms,fn}-> | JD.LPR | 125 |
| {sms, bcn} | 1 | | 17 | 5 | {sms}->Pahealth.L{sms, bcn}-> Szzc.L{sms, bcn}-> | Szzc.BCC | 3 |
| {sms, id(0,4)[3]} | 2 | 1 | | | {sms}->Zhixing.L{sms, id(0,4)}-> | Umetrip.LPR | 45 |
| {sms, id(0,6)} | 2 | 2 | | | {sms}->QQ.LPR->QQ.SSO[1]> | Vmall.L | 257 |
| {sms, id(2,4)} | 2 | 2 | | | {sms}->MOMO.L{sms, id(0,4)}->Zhihu.L{sms, id(4,4)}-> | JD.LPR | 126 |
| {sms, id} | 1 | 5 | | 1 | {sms}->Sousou.L{sms, id(6,4)}->SF Express.L{sms, id}->Meituan.L{sms, id}-> | Meituan.PPR | 260 |
| {pn, e, ep} | | 1 | | | {sms}->Soul.L{sms, e}->QQ.LPR{sms, ep, e}-> | MailMaster.LPR | 53 |
| {fn, id} | 3 | | | | {sms}->Ebuy.L{sms, fn}->Youjiankang.L{sms, fn, id}-> | Keep.L | 303 |
| {sms, fn, id} | | 3 | | 7 | {sms}->Ctrip.L{sms, fn, id(1,1)}->Huajiao.L{sms,fn,id}-> | Etax.LPR | 303 |
| {pn, id, bcn} | | 2 | | | {sms}->Qunar.L{sms, id}->Unicom.L{sms, fn, id}->Hebao.L{sms, fn, id, bcn}-> | Shengbei.LPR | 78 |
| {e, ep, id} | | 1 | | | {sms}->Boss.L{sms, e}->QQ.LPR{sms, e, ep}->Tuniu.L{sms, e, ep, id}-> | Railway12306.LPR | 613 |
| {sms, id, bcn} | 1 | 1 | | 5 | {sms}->Tujia.L{sms, id}->Taobao.L->Taobao.SSO-> | Alipay.L | 78 |
| {sms, fn, bcn} | | | 2 | | {sms}->Xiaohongshu.L{sms, fn}->PICC.L{sms, fn, bcn}->Qunar.L{sms, fn, bcn}-> | Qunar.BCC | 28 |
| {e, ep, id, bcn} | | 2 | | | {sms}->Soul.L{sms, e}->QQ.LPR{sms, e, ep}->Gtgj.L{sms, e, ep, id}-> PICC.L{sms, e, ep, id, bcn}-> | JD finance.LPR | 813 |
| {sms, fn, id, bcn} | | 3 | 9 | 13 | {sms}->Qunar.L{sms, fn, id}->Pahealth.LPR{sms, fn, id, bcn}-> | Unionpay.LPR | 113 |
| {fn, id, faceid} | | 2 | | | {sms}->Airbnb.L{sms, fn, id(6,2)}->SoYoung.L{sms, fn, id}-> Hebao.L{sms, fn, id, bcn}->Alipay.L->Alipay.SSO-> | MISP.L | 113 |
| {sms, faceid} | 7 | | | | {sms}->Haohuan.L{sms, fn, id(6,4)}->JD.L->JD.SSO-> | JD finance.L | 126 |
| Total* | 15 | 18 | 28 | 24 | | | 3,437 |

[1] L: Login, LPR: Login Password recovery, BCC: Bank card connection, PPR: Payment password recovery, SSO: Single sign-on.
[2] pn: phone number, sms: SMS OTP + pn, e: email address, ep: email OTP, fn: full name, id: ID number, bcn: bank card number.
[3] id(0,4): the first 0 digit and the last 4 digits of the ID number.
* Note that apps may provide more than one *Auth.Cond*. Each *Auth.Cond* of *App.Op* is counted separately in table.

could jeopardize 49 of 65 apps that deployed *PaFA*: breaking the authentications of 15 apps' login, 18 apps' password recovery, 28 apps' bank card connection, and 24 apps' payment password recovery. Specifically, Tab. II displays the 17 types of *PaFA* conditions of vulnerable 49 apps and attack paths that can be leveraged by the attacker who can only access the victim's SMS OTP initially. Authentication conditions that require more factors try to make the system more secure, but an attacker is able to find many potential attack paths. For example, for the Authentication condition `ep + e + id + bcn`, the attacker has 813 attack path choices. Moreover, although the attacker cannot break "FaceID" authentication directly, he could bypass it by *Bacae* attack (the last two rows in Tab. II). In summary, *MAGGIE* discovered a total of 3,437 optimized attack paths. Some concrete attack paths are provided in table, showing the process by which an attacker can successfully breach the Authentication condition.

### C. Dissecting Attackers without SMS OTP

To evaluate *PaFA* more comprehensively, beyond our threat model, we ran *MAGGIE* with different attacker initial ability ($FV_{init}$) settings without SMS OTP. Our findings are as follows: ① If starting with email verification code only [23], the attacker could additionally break two apps' login authentication with 500+ attack paths. For example, `Railway 12306` requires email verification code and ID number to reset the login password, while `Tuniu` allows the attacker to gain the victim's ID number by exploiting the login password recovery with only an email verification code. The attack path could be $\{e, ep\}$->*Tuniu.LPR*$\{e,\ ep,\ id\}$->*Railway12306.LPR*. ② Face authentication systems have been demonstrated vulnerable to many attacks [24][25]. If starting with FaceID, the attacker could also break apps' *PaFA* through *Bacae* attack. For instance, an attack path could be $\{faceid\}$->*Alipay.L*$\{faceid,\ fn,\ id\}$->*MISP.LPR*. ③ If the attacker could initially obtain only some PII (possibly due to phishing attacks [26]), he could potentially compromise several apps that only require PII for authentication through *Bacae* attack. For example, an attacker with a user's ID number and bank card number can log in to two apps, `Vipshop` and `Shengbei`, through password recovery and obtain the full name. With the additional PII, the attacker could then reset the phone number associated with the target apps (such as `Keep`, `Yunmanman` and `YunmanmanConsignor`) and gain SMS OTP to log in to them. In addition, during the experiment we observed that some apps, like `Eleme`, allow users to authenticate without SMS OTP by providing their history order number. This interesting kind of secret can be found in users' payment apps. In conclusion, although under different threat models without SMS OTP, *PaFA* remains vulnerable to our *Bacae* attack.

### D. Impact

**Consequence.** Out of the total 65 apps that deployed the *PaFA* system, 49 apps (75.4%) are susceptible to *Bacae* attack. These vulnerable apps are found in various categories, such as "UnionPay" for payment, "Taobao" for shopping, "JD Finance" for finance, "Ctrip" for tourism, and "QQ mail" for communication. Exploiting the cross-attack on these apps can indeed lead to severe consequences. Some potential outcomes include: ① Hijacking user accounts. Attackers can manipulate user sessions, alter account credentials, and gain unauthorized access to sensitive information, thereby taking control of the user's accounts. ② Unauthorized purchases. Once the attacker gains control of a user's account in certain apps, they can make unauthorized purchases using the user's funds. This can result in financial loss for the victim and damage their reputation. ③ Fraudulent activities. By accessing a user's account and PII, attackers can engage in various fraudulent activities. This includes applying for loans, opening new accounts, or conducting transactions on behalf of the user. These actions can have significant financial and legal consequences for the victim. We successfully conducted several PoC attacks on our own accounts, as presented in the case studies in Section V-A.

**Magnitude.** In addition to exploring attack paths on the dataset that covers all kinds of apps evenly, we conducted a user study to estimate the real impact of *Bacae* attack on users. The questionnaire is designed simply: we showed the 234 apps to users (divided into 16 groups) and asked the participants

TABLE III. DEMOGRAPHICS OF THE QUESTIONNAIRE PARTICIPANTS

| | | n (sum=208) | % |
|---|---|---|---|
| **Gender** | M | 92 | 44.23 |
| | F | 116 | 55.77 |
| | No answer | 0 | 0 |
| **Age** | 18-25 | 29 | 13.94 |
| | 26-35 | 112 | 54.85 |
| | 36-45 | 49 | 23.56 |
| | 46-55 | 11 | 5.29 |
| | 56+ | 7 | 3.37 |
| | No answer | 0 | 0 |
| **Education** | Below bachelor | 43 | 20.67 |
| | Bachelor | 137 | 65.87 |
| | Master or above | 22 | 10.58 |
| | No answer | 6 | 2.88 |

to select the apps they had registered and used. In addition, we added one attention test question in the middle of the questionnaire to filter out those participants who did not answer seriously. Finally, we ran *MAGGIE* to discover attack paths from apps that were registered and used by every real user.

We recruited 281 participants via a popular online research platform `Wenjuanxing`[4]. The survey[5] was done on April 29, 2023. We rejected the responses that were completed in less than 90 seconds or wrongly answered the attention test question to ensure the quality of the results. Finally, we got 208 effective responses. On average, it took 4 minutes to finish the survey. Based on the local income, participants who completed the survey in China received 2 CNY. The participants' demographics are presented in Table III.

Based on the survey results, we found that the *Bacae* attack has a high success rate on real users. We discovered that 94.2% of participants had at least one attack path in the simultaneously used apps to break the authentication of an installed app. Although this success rate is potentially overestimated due to the incompletely/dishonestly filling in personal information and occasional account deletion [27], *Bacae* may still have a high probability of success. In the survey, we limited participants to selecting apps from our dataset of 234 apps. However, it is likely that individuals used other apps beyond this set, resulting in an increased number of potential targets and attack paths.

### *E. Discussion*

**SMS OTP becomes the sole protection.** Our study found that Chinese apps rely too heavily on SMS OTP for authentication. Approximately 27.8% of apps, particularly those related to financial transactions, have added the *PaFA* mechanism to obtain additional security protection. However, PII is not a secure authentication factor. In addition to frequent data breaches nowadays [28] and threats revealed by previous stuides [4][29][30], our study reveals a new concerning fact: when people use many apps simultaneously (which is true revealed by our user study), SMS OTP becomes the sole protection because of the PII exposure in apps and business-related interactions of two or more apps, rendering the *PaFA* mechanism useless. Striking a balance between security and usability remains a significant challenge.

Moreover, unfortunately as stated in the threat model, SMS OTP is not secure either and can be vulnerable to numerous attacks. For example, SIM-swap attacks [11] can allow attackers to redirect incoming SMS messages to their own devices, giving them access to OTPs. Also, SMS messages can be sniffed [31] or intercepted by malicious apps [32], potentially allowing attackers to obtain OTPs without even compromising the user's device. These vulnerabilities highlight the need for more secure authentication methods beyond SMS OTP. Using additional authentication factors in conjunction with SMS OTP is indeed recommended for stronger security. Biometrics such as fingerprint or facial recognition can provide an additional layer of security and reduce the reliance on SMS OTP. However, only 39 apps (out of 234 apps) provide biometric authentication, including face recognition and fingerprint recognition, and only 4 of them require biometrics to be mandatory. Many users may choose not to enable biometric authentication when using mobile apps, which may limit the effectiveness of this authentication method. On the other hand, the hardware requirements also limit the deployment of biometric authentication.

**Risk control.** During our experiment, we observed that some apps set up risk control systems to safeguard their users, but their security enhancement was limited. In total, 18 apps (out of 234) added additional authentication factors for login authentication when tested under changed device and IP address conditions (see Section IV-B). Taking `Taobao` as an example, after changing the device and IP address, it requires the user to additionally provide the last four digits of the ID number for login.

As shown in Tab. IV, 7 apps required PII as additional authentication factors besides SMS OTP. However, as revealed by this paper, PII gathering attacks can compromise these added factors. Therefore, such added authentications may not be effective in improving security. Moreover, 10 apps added biometric authentication, such as FaceID, to enhance login security, while 4 apps added different types of authentication, such as historical order numbers. These types of authentication cannot be broken by PII gathering. However, part of them can be compromised by SSO or account sharing, reducing its effectiveness. Overall, 15 out of the 18 apps that implemented risk control measures to enhance login security are vulnerable to *Bacae* attack, suggesting that the additional authentication for risk control provides limited benefits in terms of security.

**Inconsistent protection in websites.** Our findings demonstrate that different mobile apps may have inconsistent security policies regarding the display of PII. Inspired by this, we further manually examined the websites of some mobile apps that provide sensitive services and found that the display of the same PII was inconsistent with the display on the mobile apps. Take `Lifangtong` as an example. The login conditions of the website and mobile app are the same. However, the bank card number, which is partly displayed (first 6 digits and last 4 digits) in the `Lifangtong` app, is fully displayed on the website. Over half of the app vendors in our dataset provide a

[4]https://www.wjx.cn

[5]Our institution does not require IRB for online questionnaire-based studies. However, we adhere to standard ethical research procedures and are committed to safeguarding participants' information and maintaining its confidentiality. No personally identifiable data was collected from participants in the questionnaire.

TABLE IV. COMPARISON OF LOGIN AUTHENTICATION BEFORE AND AFTER CHANGING DEVICE AND IP.

| App | Original login cond. | Additonal login cond. (New device and IP) | Account sharing |
|---|---|---|---|
| JD | {sms} | {id(2,4)[1]} | QQ/Wechat |
| Taobao | {sms} | {id(0,4)} / {faceid} | Alipay |
| Hebao | {sms} | {id(0,6)} | \ |
| Taobao.movie | {sms} | {id(0,4)} | Alipay |
| Alipay | {sms} | {id, bank card number} / {faceid} | Taobao |
| Vipshop | {sms} | {id}/{bank card number} | QQ/Wechat |
| Vmall | {sms} | {email OTP} / {id number(0,6)} | QQ/Wechat |
| DiDa | {sms} | {faceid} | Wechat |
| JD finance | {sms} | {faceid} | JD/Wechat |
| Kuaishou.nebula | {sms} | {faceid} | QQ/Wechat |
| Kuaishou | {sms} | {faceid} | QQ/Wechat |
| Zhenai | {sms} | {faceid} | \ |
| Railway12306 | {sms} | {faceid} | \ |
| QQ | {sms} | {fn, id, faceid} | \ |
| Hello bike | {sms} | {faceid} / {Alipay order number} | Alipay |
| Maoyan.movie | {sms} | {Meituan order number} | QQ/Wechat |
| Haokanshipin | {sms} | {username} | QQ/Wechat |
| WeChat | {sms} | {QR code} | QQ |

[1] id(2,4) indicates the first 2 digits and the last 4 digits of the ID number.

corresponding web version of their services, which means that attacks can originate from a wider range of surface. Therefore, we believe that expanding the attack idea to websites besides apps could potentially make the authentication mechanisms of apps become weaker.

### F. Responsible Disclosure

We have made responsible disclosures to the app vendors and CNCERT/CC (the authority for coordinating and handling cybersecurity threats and incidents in China). For app vendors that have Security Response Center or contacting emails, we report the issues related to their apps by the platforms or emails. For app vendors without public contact information, we reported the relevant risk reports to CNCERT/CC and requested their assistance.

`Tencent` and `Alibaba` acknowledged the issues we reported but did not make modifications. Some app vendors (e.g., `Unionpay`) have updated their app versions during our experiment and added FaceID or CVV to their *PaFA* mechanisms, which prevent *Bacae* attack. However, we received few responses from most vendors until the submission of this paper. This lack of response is likely due to the fact that our attack involves multiple apps, and each vendor may believe that they are not solely responsible for the risk.

## VI. SUGGESTIONS AND LIMITATIONS

### A. Suggestions

The most important lesson learned from this study is that the widely deployed *PaFA* mechanisms in (Chinese) mobile apps provide few security benefits. The imperfect/different strategies of app vendors in information protection and authentication mechanisms contribute to the success of *Bacae* attack. Therefore, a standardized data display mechanism and an authentication strength assessment (e.g., with the help of *MAGGIE*) before business cooperation agreed by cross-app vendors would be beneficial to users' security. However, a unified standard usually takes a long time to be carried out. For app vendors today who want to mitigate *PaFA* attack, we suggest they add additional biometric authentication mechanisms (e.g., FaceID) and do not rely on PII for authentication purposes, which provides limited security benefits but hurts usability.

### B. Limitation

**Further analysis on risk control.** As discussed in Section V-E, some risk control measures could mitigate *Bacae* attacks. However, the effectiveness of these measures is often difficult to assess due to the black-box and random nature of the system. Interestingly, during our experiment, for example, a certain risk control measure implemented by `Taobao` required us to select a friend from our `Taobao` contacts to confirm our identity. However, there was no friend associated with the test account, and we were able to bypass the risk control measure by clicking on any name displayed on the screen. While our method may not be able to perfectly measure the impact of risk control, our results show that *Bacae* attacks are still realistic in some cases where risk control is present. In the future, more effective methods for analyzing and understanding risk control will be necessary.

**Sounder implementation of *MAGGIE*.** First, our tool requires manual assistance to complete the authentication process when anti-automation mechanisms are present (e.g.,verification codes and security keyboards). In the future, computer vision processing technology could be used to reduce the need for manual assistance and make *MAGGIE* more automated. Second, *XHelper* is built based on the open source tool `UIAutomator`, so some of its functions are limited to the implementation of `UIAutomator`. Finally, *MAGGIE* could be made more efficient and effective by integrating state-of-the-art methods [33] from the software engineering field that focus on app UI exploration and testing.

**Limitations of dataset.** First, the scale of the number of apps analyzed could be improved. Although we selected the 234 apps with the highest downloads in 39 categories as comprehensively as possible, we cannot claim to have covered the entire app ecosystem. Second, we only studied the app market in China. The mobile app ecosystem in foreign countries is quite different from that of China, and the authentication systems may be designed differently. Additionally, PII in different countries may vary. For example, in China, the 7th to 14th digits of the ID number represent an individual's birthday. Thus, our model may require minor changes to be applied to the *PaFA* system in other countries. We believe that the mobile app market in China is huge and representative. However, we encourage researchers around the world to cooperatively study the potential risks in the *PaFA* system in the future. However, we believe that *PaFA* systems in other countries may also have the potential risks that need more research by checking apps from other sources. For instance, the password recovery process of `Outlook` requires name, birth date, region and other personal information, which can also possible be exposed by other apps. This means that our results are a conservative estimate of the actual attack surface.

## VII. RELATED WORK

**Authentication security of mobile apps.** Many works have been done to study the security of authentication mechanisms [34–38]. Password-based authentication schemes are vulnerable to guessing attacks, offline dictionary attacks, brute-force attacks, and credential stuffing attacks [39][40]. To avoid these attacks, multi-Factor Authentication (MFA) - i.e., the combination of two or more authentication factors, is being

increasingly advocated for securing online services. For example, many bank apps use face features to authenticate users. However, facial recognition can be invaded by trojaned model provided by attackers [25]. Another widely used authentication is SMS OTP which avoids users' creation and memorization of passwords [41]. However, existing techniques provide convenience for attackers to obtain SMS OTP. For example, attackers can read SMS OTP through malware [42], SIM card swapping attacks [11], and wireless network hijacking attacks [31]. By contrast, considering above risks in our threat model, we evaluated another authentication factor of mobile apps – *PaFA*.

**Account recovery.** Prior papers evaluated the personal security questions for fallback authentication and demonstrated that secret questions have poor security and reliability in account recovery authentication [3] [4] [30]. Security credentials can be obtained through several approaches, e.g., online guessing [3], publicly available answers [4] and social engineering [30]. Even some pieces of private information of users might be exposed during the process of recovering the password [29]. Besides, Guri et al. found the password recovery process could leak users' private information [29]. Prior studies examined the personal security questions and focus on the "vulnerability to random guessing" because the answers are shared by different users. However, sensitive security questions, like "ID number", required by the *PaFA* are generally infeasible to guess. Instead, this work propose a novel concept of "vulnerability to cross-app PII collection and interactions", which arises from inconsistent security policies in apps regarding their use and display of PII, as well as their business interactions with apps of different authentication strengths.

**Detection of privacy leakage in mobile apps.** Considering mobile apps have access to personal sensitive data, many approaches [43–48] have been proposed to detect privacy leakage in mobile apps. They leveraged many state-of-the-art techniques like formal models, network data flow analysis, NLP, program analysis, etc, to identify whether and what personal information is gathered/sent out by apps. In sharp contrast, we focus on the personally information exposed on app pages, which can be easily gathered by the attacker in our threat model. So *MAGGIE* uses Android GUI testing technology to traverse app UIs. Our goal is quite different from the detection of privacy leakage in apps.

## VIII. Conclusion

We performed the first study to analyze whether the simultaneous use of multiple apps will affect the *PaFA* mechanism of a target app. Our approach is building a state machine model that describes the adversary who simultaneously operates hundreds of apps and leverages model checking to find attack paths that break the target *PaFA*. We built a semi-automatic tool *MAGGIE* and measured 234 Chinese apps, revealing that the widely used *PaFA* of mobile app cannot effectively protect users as designed and SMS OTP becomes the single point of failure in most cases. Furthermore, a large number of users may be subject to *Bacae* attack considering the list of apps they registered and used, as found by our survey. Finally, we reported our finding to related parties and put forward some suggestions to mitigate such issues. Our new findings and understanding will lead to better protection of users and provide valuable insights towards app authentication mechanisms.

## Acknowledgment

This work was supported by the National Key Research and Development Program (2023YFB3106400, 2023QY1202), the National Natural Science Foundation of China (U2336203, U1836210), and the Key Research and Development Science and Technology of Hainan Province (GHYF2022010). Yan Jia is supported by the National Natural Science Foundation of China (62102198) and China Postdoctoral Science Foundation (2021M691673, 2023T160335).

## References

[1] "App industry china," https://daxueconsulting.com/china-app-market/, 2023.
[2] "Mobile apps in china," https://www.statista.com/topics/5577/mobile-apps-in-china/#topicOverview, 2023.
[3] A. Rabkin, "Personal knowledge questions for fallback authentication: Security questions in the era of facebook," in *Proceedings of the 4th Symposium on Usable Privacy and Security*, 2008, pp. 13–23.
[4] J. Bonneau, E. Bursztein, I. Caron, R. Jackson, and M. Williamson, "Secrets, lies, and account recovery: Lessons from the use of personal knowledge questions at google," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 141–150.
[5] "What to do if your phone is lost or stolen," https://www.asurion.com/connect/tech-tips/what-to-do-when-your-phone-is-lost-or-stolen/, 2023.
[6] "Mobile device security," https://www.channelpronetwork.com/article, 2023.
[7] "Phone reported stolen," https://www.bbc.com/news/uk-england-london-65105199, 2023.
[8] "134 cellphones lost, stolen daily in city," https://indianexpress.com/article/cities/mumbai/134-cellphones-lost-stolen-daily-in-city-fir-registered-in-3-pc-of-the-cases-rti-7873867/, 2022.
[9] "Smartphone theft statistics," https://www.identity-theft-awareness.com/smartphone-theft-statistics.html, 2022.
[10] "Mobile phone stolen," https://www.statista.com/statistics/1027661/brazil-share-population-mobile-phone-stolen/, 2022.
[11] 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.
[12] "Android permissions." https://developer.android.com/reference/android/Manifest.permission, 2023.
[13] "Otp frauds increases," https://www.cnbctv18.com/technology/otp-frauds-increases-three-times-post-pandemic-common-method-to-scam-people-16896221.html, 2023.
[14] "One time password frauds," https://www.statista.com/statistics/1097969/india-number-of-otp-frauds-recorded-by-leading-state/, 2022.
[15] "Sms pumping fraud epidemic," https://www.infosecurity-magazine.com/news/experts-warn-of-sms-pumping/, 2023.
[16] "Sms sold," https://new.qq.com/rain/a/20220722A09WAZ00, 2022.
[17] "Sms sold," https://www.thepaper.cn/newsDetail_forward_18935983, 2022.
[18] M. Line, "Maggie," https://github.com/ncnipc0/MAGGIE, 2023.
[19] "Uiautomator," https://stuff.mit.edu/afs/sipb/project/android/docs/tools/help/uiautomator, 2022.
[20] "Huawei app gallery," https://appgallery.huawei.com/Featured, 2021.

[21] "Vivo app store," http://info.appstore.vivo.com.cn/detail/47998, 2021.
[22] "Tencent app centre," https://sj.qq.com/, 2021.
[23] Y. Li, H. Wang, and K. Sun, "Email as a master key: Analyzing account recovery in the wild," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1646–1654.
[24] Y. Yan and Z. Yang, "Spoofing real-world face authentication systems through optical synthesis," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 882–898.
[25] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 113–131.
[26] Z. Alkhalil, C. Hewage, L. Nawaf, and I. Khan, "Phishing attacks: A recent comprehensive study and a new anatomy," *Frontiers in Computer Science*, vol. 3, p. 563060, 2021.
[27] Y. Liu, Y. Jia, Q. Tan, Z. Liu, and L. Xing, "How are your zombie accounts? understanding users' practices and expectations on mobile app account deletion," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 863–880.
[28] F. Schlackl, N. Link, and H. Hoehle, "Antecedents and consequences of data breaches: A systematic review," *Information & Management*, p. 103638, 2022.
[29] M. Guri, E. Shemer, D. Shirtz, and Y. Elovici, "Personal information leakage during password recovery of internet services," in *2016 European intelligence and security informatics conference (EISIC)*. IEEE, 2016, pp. 136–139.
[30] C. Karlof, J. D. Tygar, and D. A. Wagner, "Conditioned-safe ceremonies and a user study of an application to web authentication." in *NDSS*, 2009.
[31] W. Jin, X. Ji, R. He, Z. Zhuang, W. Xu, and Y. Tian, "Sms goes nuclear: Fortifying sms-based mfa in online account ecosystem," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2021, pp. 7–14.
[32] Y. Shen, P.-A. Vervier, and G. Stringhini, "A large-scale temporal measurement of android malicious apps: Persistence, migration, and lessons learned," *arXiv preprint arXiv:2108.04754*, 2021.
[33] X. Zhang, L. Fan, S. Chen, Y. Su, and B. Li, "Scene-driven exploration and gui modeling for android apps," *In Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, 2023.
[34] I. Stylios, S. Kokolakis, O. Thanou, and S. Chatzis, "Behavioral biometrics & continuous user authentication on mobile devices: A survey," *Information Fusion*, vol. 66, pp. 76–99, 2021.
[35] C. Wang, Y. Wang, Y. Chen, H. Liu, and J. Liu, "User authentication on mobile devices: Approaches, threats and trends," *Computer Networks*, vol. 170, p. 107118, 2020.
[36] Y. Zhang, C. Xu, H. Li, K. Yang, N. Cheng, and X. Shen, "Protect: efficient password-based threshold single-sign-on authentication for mobile users against perpetual leakage," *IEEE Transactions on Mobile Computing*, vol. 20, no. 6, pp. 2297–2312, 2020.
[37] M. A. Ferrag, L. Maglaras, A. Derhab, and H. Janicke, "Authentication schemes for smart mobile devices: Threat models, countermeasures, and open research issues," *Telecommunication Systems*, vol. 73, no. 2, pp. 317–348, 2020.
[38] A. Acien, A. Morales, R. Vera-Rodriguez, J. Fierrez, and R. Tolosana, "Multilock: Mobile active authentication based on multiple biometric and behavioral patterns," in *1st International Workshop on Multimodal Understanding and Learning for Embodied Applications*, 2019, pp. 53–59.
[39] P. Markert, D. V. Bailey, M. Golla, M. Dürmuth, and A. J. Aviv, "This pin can be easily guessed: Analyzing the security of smartphone unlock pins," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 286–303.
[40] S. Agrawal, P. Miao, P. Mohassel, and P. Mukherjee, "Pasta: password-based threshold authentication," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2042–2059.
[41] 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 *Network and Distributed Systems Security (NDSS) Symposium 2021*, 2021.
[42] A. D. P. Center, "Additional requirements for the use of specific permissions." https://play.google.com/about/privacy-security-deception/permissions/, 2019.
[43] H. Lu, L. Xing, Y. Xiao, Y. Zhang, X. Liao, X. Wang, and X. Wang, "Demystifying resource management risks in emerging mobile app-in-app ecosystems," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security*, 2020, pp. 569–585.
[44] L. Shi, J. Ming, J. Fu, G. Peng, D. Xu, K. Gao, and X. Pan, "Vahunt: Warding off new repackaged android malware in app-virtualization's clothing," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 535–549.
[45] Y. Chen, M. Zha, N. Zhang, D. Xu, Q. Zhao, X. Feng, K. Yuan, F. Suya, Y. Tian, K. Chen *et al.*, "Demystifying hidden privacy settings in mobile apps," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 570–586.
[46] M. Zha, J. Wang, Y. Nan, X. Wang, Y. Zhang, and Z. Yang, "Hazard integrated: Understanding security risks in app extensions to team chat systems," in *Network and Distributed Systems Security (NDSS) Symposium*, 2022.
[47] Z. Yang, W. Pei, M. Chen, and C. Yue, "Wtagraph: Web tracking and advertising detection using graph neural networks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1540–1557.
[48] P. Dodia, M. AlSabah, O. Alrawi, and T. Wang, "Exposing the rat in the tunnel: Using traffic analysis for tor-based malware detection," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 875–889.

## APPENDIX

### A. Detailed Experiment Results

Tab.V details the dataset and the experiment results for each app. Besides the apps' basic information, the "One of the vulnerable authentication conditions" column lists the vulnerabilities found by MAGGIE for each app, and the corresponding authentication factors required by the vulnerable app operations. An attacker could find at least one attack path to bypass these authentication conditions. Note that for each operation, there may be multiple authentication conditions available for the attacker to choose from. This table only lists one of the (vulnerable) authentication conditions. We also listed additional PII that an attacker can gain after logging in, which he can use to build attack paths. To make it easier to identify the 185 apps which show additional PII post-login (as discussed in Section V-B), the table also presents the authentication condition {sms} in the **Login** and **LPR** columns. Apps that deploy *PaFA* but are vulnerable to *Bacae* attack may further weaken the authentication strength of other apps that do not deploy *PaFA*, so that the number of apps impacted by *Bacae* attack in Tab.V is more than 49. Note that the experiments were conducted during 2021 to 2022 and some apps were updated. For security reasons, we have hidden the version number of apps. The attack paths for each vulnerable authentication condition are displayed in Tab.II.

TABLE V: Apps in dataset and identified vulnerabilities under the SMS OTP-only threat model

| Category | App | | One of (vulnerable) authentication conditons[1] | | | | Additonal PII post-login[2] | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | App name | Downloads | Login | LPR | BCC | PPR | fn | bir | id | e | bcn |
| Map navigation | Amap | 11.6B+ | {sms} | | | | ● | ● | ○ | ◑ | ○ |
| | Baidumap | 10.3B+ | {sms} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| | Tencentmap | 2.7B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Omap | 130.9M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | BaiduCarlife+ | 66.2M+ | {sms} | | | | ◑ | ○ | ◑ | ○ | ○ |
| | Beidoumap | 44.5M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| Ride-hailing | DiDa | 1.1B+ | {Wechat.SSO} | | {sms, fn, id, bcn} | | ◑ | ○ | ◑ | ○ | ◑ |
| | Huolala | 696.1M+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Huaxiaozhurider | 382.2M+ | {sms} | | {sms, fn, id, bcn} | {sms, fn, id, bcn} | ○ | ○ | ○ | ○ | ○ |
| | Caocaokeji | 251.8M+ | {sms} | | | | ◑ | ● | ◑ | ● | ○ |
| | T3go | 250.6M+ | {Alipay.SSO} | | | | ◑ | ○ | ◑ | ○ | ○ |
| | Yigaosu | 214.3M+ | {sms} | | {sms, fn, id, bcn} | | ● | ○ | ○ | ○ | ◑ |
| Instant messaging | WeChat | 25.4B+ | {send sms, QQ.acct[3]} | | | | ● | ○ | ○ | ● | ◑ |
| | QQ | 23.3B+ | | {sms} | {sms, bcn, ppwd[3]} | {sms, id, bcn} | ◑ | ● | ○ | ● | ◑ |
| | Weplay | 234.1M+ | {sms} | | | | ○ | ○ | ◑ | ○ | ○ |
| | Ifreetalk | 156.4M+ | {sms} | | | | ◑ | ● | ◑ | ○ | ○ |
| | Palmchat | 154.8M+ | {sms} | | {sms, bcn, ppwd} | {sms, fn, id, bcn} | ● | ● | ◑ | ○ | ◑ |
| | WanBa | 112.4M+ | {sms} | | | | ◑ | ● | ◑ | ○ | ○ |
| Online community | Weibo | 20.4B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Xiaohongshu | 11.8B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | MOMO | 3.6B+ | {sms} | | | | ◑ | ● | ◑ | ○ | ○ |
| | Qzone | 1.7B+ | {QQ.SSO} | | | | ○ | ● | ○ | ● | ○ |
| | Baidutieba | 1.4B+ | {sms} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| | Douban | 362.0M+ | {sms} | | | | ● | ○ | ○ | ◑ | ○ |
| Online payment | Alipay | 15.8B+ | {Taobao.SSO} | {sms, id, fn, bcn} | {sms, bcn} | | ◑ | ○ | ○ | ● | ◑ |
| | Etax | 3.5B+ | | {sms, fn, id} | | | ○ | ● | ○ | ● | ◑ |
| | Unionpay | 1.7B+ | | {sms, fn, id, bcn} | {sms, bcn} | {sms, fn, id, bcn} | ○ | ○ | ○ | ○ | ○ |
| | Bestpay | 1.0B+ | {sms} | | {sms, ppwd} | {sms, fn, id} | ◑ | ○ | ◑ | ◑ | ◑ |
| | Fenqile | 122.1M+ | {sms} | | {sms, fn, id, bcn} | | ● | ○ | ○ | ○ | ◑ |
| | Hebao | 114.5M+ | {sms, id(0, 6)[4]} | {sms, id} | {sms, fn, id, bcn} | {sms, fn, id} | ◑ | ● | ○ | ◑ | ● |
| Online shopping | Taobao | 22.9B+ | {sms, id(0, 4)} | | | | ● | ○ | ○ | ◑ | ○ |
| | JD | 18.8B+ | {sms, id(2, 4)} | {sms, fn} | {sms, bcn} | {sms, bcn} | ● | ● | ○ | ◑ | ◑ |
| | Vmall | 13.0B+ | {sms, id(0, 6)} | {sms, id(0, 6)} | | | ● | ○ | ○ | ● | ◑ |
| | Vipshop | 12.8B+ | {sms, id} | {pn, id, bcn} | {sms, bcn} | {sms, id, bcn} | ● | ● | ○ | ● | ◑ |
| | Taote | 6.0B+ | {Alipay.SSO} | | | | ● | ○ | ○ | ○ | ○ |
| | Ebuy | 2.8B+ | {sms} | {sms, id} | {sms, bcn} | {sms, id, bcn} | ● | ● | ○ | ◑ | ◑ |
| Food delivery | Meituan | 18.2B+ | {sms} | | {sms, bcn} | {sms, id} | ● | ● | ◑ | ○ | ◑ |
| | Eleme | 3.3B+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Meituantakeout | 2.5B+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Xiachufang | 1.8B+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Hippo | 944.0M+ | {sms} | | | | ● | ○ | ○ | ◑ | ○ |
| | KFC | 794.2M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| Mail and package delivery | Cainiao | 1.4B+ | {sms} | | | | ● | ○ | ◑ | ○ | ○ |
| | SF Express | 255.1M+ | {sms} | | | | ● | ● | ◑ | ○ | ○ |
| | Yunmanman | 156.0M+ | {sms} | | {sms, bcn} | {sms, bcn} | ◑ | ○ | ◑ | ○ | ◑ |
| | Ishansong | 141.3M+ | {sms} | | | | ● | ○ | ◑ | ○ | ○ |
| | YunmanmanConsignor | 109.1M+ | {sms} | | {sms, bcn} | {sms, bcn} | ◑ | ○ | ◑ | ○ | ◑ |
| | Sousou | 100.0M+ | {sms} | | | | ● | ○ | ◑ | ○ | ○ |
| Transportation ticketing | Rail12306 | 1.6B+ | | {e, ep[2], id} | | | ● | ○ | ○ | ○ | ○ |
| | Chelaile | 1.0B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Mybus | 800.0M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Zhixing | 684.0M+ | {sms} | | | | ● | ● | ◑ | ◑ | ○ |
| | Gtgj | 329.8M+ | {sms} | | | | ● | ○ | ● | ○ | ○ |
| | Lulutong | 329.2M+ | {sms, Rail.acct, id(0, 4)} | | | | ● | ● | ○ | ◑ | ○ |
| Dating and matchmaking | Tantan | 1.7B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Soul | 1.4B+ | {sms} | | | | ○ | ● | ○ | ● | ○ |
| | Ailiao | 266.4M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Yidui | 225.9M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Blued | 162.9M+ | {sms} | | | | ○ | ● | ○ | ◑ | ○ |
| | Zhenai | 150.9M+ | | | | | ◑ | ● | ◑ | ○ | ○ |
| Job search and recruitment | 58client | 8.5B+ | {sms} | | | | ● | ◑ | ◑ | ○ | ◑ |
| | Boss | 900.0M+ | {sms} | | | | ● | ● | ○ | ● | ○ |
| | Zhilianzhaopin | 833.3M+ | {sms} | | | | ● | ● | ○ | ● | ○ |
| | Ganji | 703.4M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | 51job | 570.0M+ | {sms} | | | | ● | ● | ○ | ● | ○ |
| | Maimai | 438.3M+ | {sms} | | {sms, bcn, fn, id} | {sms, bcn, id} | ● | ● | ○ | ● | ◑ |
| Online lending | JD finance | 1.3B+ | {JD.SSO} | {sms, id(2, 4)} | {sms, bcn} | {sms, bcn} | ◑ | ○ | ○ | ◑ | ◑ |
| | 360Loan | 336.2M+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | PaipaiDai loan | 200.2M+ | {sms} | {sms, id} | {sms, fn, bcn} | | ◑ | ○ | ◑ | ◑ | ◑ |
| | Haohuan | 183.3M+ | {sms} | | | | ● | ○ | ◑ | ○ | ◑ |
| | Ayh | 155.3M+ | | | | | ● | ○ | ◑ | ○ | ◑ |
| | Shengbei | 142.8M+ | {sms} | {pn, id, bcn} | | | ● | ○ | ◑ | ○ | ◑ |
| Used car trading | Guazihaoche | 442.6M+ | {sms} | | | | ● | ○ | ◑ | ○ | ◑ |
| | Renrenche | 64.0M+ | {sms} | | | | ◑ | ○ | ○ | ○ | ◑ |
| | Ershouche | 55.3M+ | {sms} | | | | ◑ | ○ | ○ | ○ | ○ |
| | Car300 | 48.4M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Uxin | 43.1M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Chaboshi | 17.2M+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |

**Table V continued from previous page**

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Rental housing | Anjuke | 1.9B+ | {sms} | | | ◑ | ● | ◑ | ○ | ○ |
| | Beike | 210.0M+ | {sms} | | | ○ | ○ | ◑ | ○ | ○ |
| | F100 | 610.0M+ | | | | ○ | ○ | ○ | ○ | ○ |
| | Lianjia_zhilian | 377.3M+ | | | | ○ | ○ | ○ | ○ | ○ |
| | Ziroom | 123.0M+ | {sms} | | | ● | ○ | ◑ | ● | ◑ |
| | Wawj | 90.7M+ | {sms} | | | ● | ○ | ◑ | ● | ○ |
| Medical services | Pahealth | 594.7M+ | {sms} | | | ● | ● | ● | ○ | ● |
| | MISP | 286.6M+ | {Alipay.SSO} | | | ● | ○ | ○ | ○ | ○ |
| | Threegeneyimiao | 256.0M+ | {sms} | | | ● | ● | ○ | ○ | ○ |
| | Youjiankang | 223.5M+ | {sms} | | | ● | ● | ● | ○ | ○ |
| | Haodf | 222.8M+ | {sms} | | | ● | ● | ◑ | ○ | ○ |
| | SoYoung | 191.4M+ | {sms} | | | ● | ● | ◑ | ○ | ○ |
| Travel services | Qunar | 4.8B+ | {sms} | | {sms, fn, bcn} | | ● | ● | ◑ | ○ |
| | Tianjitong | 3.9B+ | {sms} | | {sms, bcn} | | ● | ○ | ◑ | ● | ◑ |
| | Mafengwo | 825.8M+ | {sms} | | | | ● | ● | ◑ | ● | ○ |
| | Umetrip | 634.1M+ | | {sms, id(0, 4)} | {sms, bcn} | | ● | ○ | ● | ○ | ◑ |
| | Tuniu | 380.4M+ | {sms} | | {sms, bcn, fn, id} | | ● | ● | ● | ◑ | ○ |
| | Ctrip | 190.0M+ | {sms} | | {sms, bcn} | | ● | ● | ● | ◑ | ◑ |
| Hotel services | Booking | 4.9B+ | {WeChat.sso} | | | | ● | ● | ○ | ● | ◑ |
| | Htinns | 317.6M+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Airbnb | 316.5M+ | {sms} | | {sms, bcn, fn, id} | | ● | ○ | ◑ | ● | ◑ |
| | Agoda | 200.0M+ | {sms} | | | | ◑ | ○ | ○ | ○ | ○ |
| | Tujia | 115.5M+ | {sms} | | | | ● | ● | ● | ● | ○ |
| | Muniao | 95.9M+ | {sms} | | | | ● | ● | ◑ | ● | ○ |
| Online games | Wangzherongyao | 7.1B+ | {QQ.SSO} | | | | ○ | ● | ○ | ○ | ○ |
| | Miniworld | 3.8B+ | | {sms} | | | ○ | ○ | ○ | ◑ | ○ |
| | Happyelements | 2.8B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Hpjy | 2.7B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Minecraft | 1.5B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Wepiesnake | 650.0M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| Education and learning | Zhihu | 7.5B+ | {sms} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| | Baiduhomework | 5.4B+ | {sms} | | | | ● | ○ | ◑ | ○ | ○ |
| | Xuexi | 2.4B+ | | {sms} | | | ● | ● | ◑ | ○ | ○ |
| | Jiakao | 1.6B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Kuaiduizuoye | 1.3B+ | {sms} | | | | ◑ | ○ | ◑ | ○ | ○ |
| | Fenbi | 1.0B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| Local life | FleaMarket | 5.2B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Unicom | 2.3B+ | | {sms, id(0, 6)} | {sms, bcn} | {sms, id, bcn} | ● | ● | ○ | ○ | ◑ |
| | ChinaMobile | 2.1B+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Ctclient | 1.2B+ | | {sms, fn, id} | | | ● | ○ | ○ | ○ | ○ |
| | Zzmarket | 1.1B+ | {sms} | | | | ● | ● | ◑ | ○ | ○ |
| | Wzsearch | 500.0M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| Women's health | Seeyou | 1.0B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Qinbaobao | 573.7M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Babytreepregnancy | 506.8M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Mamapregnant | 331.0M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Bevol | 225.4M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Dayima | 131.7M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| Car services | Hellobike | 2.3B+ | {Alipay.SSO} | | | | ● | ○ | ○ | ○ | ○ |
| | Szzc | 126.4M+ | {sms} | | {sms, bcn} | | ○ | ○ | ◑ | ● | ◑ |
| | 1hai | 111.9M+ | {sms} | | | | ● | ○ | ◑ | ◑ | ○ |
| | Didibike | 69.5M+ | {sms} | | | | ◑ | ○ | ◑ | ○ | ○ |
| | Qhzc | 51.4M+ | {sms} | | | | ● | ○ | ○ | ● | ○ |
| | Ofo | 31.9M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| Email and cloud storage | Baidunetdisk | 5.7B+ | {sms} | | | | ◑ | ● | ◑ | ◑ | ○ |
| | QQmail | 3.4B+ | {QQ.SSO} | | | | ○ | ○ | ○ | ● | ○ |
| | 139mail | 421.5M+ | {sms} | | | | ◑ | ○ | ○ | ● | ○ |
| | MailMaster | 381.8M+ | | {pn, e, ep} | | | ● | ○ | ○ | ● | ○ |
| | Mcloud | 367.3M+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Neteasemail | 152.8M+ | | {sms} | | | ○ | ○ | ○ | ● | ○ |
| Online meetings | TencentMeeting | 2.3B+ | {sms} | | | | ● | ○ | ○ | ● | ○ |
| | Zhumu | 21.6M+ | {sms} | | | | ● | ○ | ○ | ◑ | ○ |
| | Scorpionteams | 15.4M+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | GNetMeetNow | 13.2M+ | {sms} | | | | ● | ○ | ○ | ● | ○ |
| | FastMeetingcloud | 8.5M+ | {sms} | | | | ● | ○ | ○ | ● | ○ |
| | HuaweicloudMeeting | 8.3M+ | {sms} | | | | ● | ○ | ○ | ◑ | ○ |
| Live streaming | Huya | 2.8B+ | {sms} | | | | ◑ | ● | ◑ | ○ | ○ |
| | Bettas | 1.8B+ | {sms} | | | | ◑ | ● | ◑ | ◑ | ◑ |
| | YY | 970.0M+ | {sms} | | | | ◑ | ● | ◑ | ○ | ○ |
| | Ingkee | 373.7M+ | {sms} | | | | ◑ | ● | ◑ | ○ | ○ |
| | Yangshipin | 362.4M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Huajiao | 300.0M+ | {sms} | | | | ● | ● | ● | ○ | ○ |
| Online videos and audio | Iqiyi | 14.3B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Kuaishou.nebula | 13.3B+ | {QQ.SSO} | | | | ● | ● | ○ | ○ | ○ |
| | Douyin.lite | 12.5B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Xiguashipin | 11.7B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Youku | 11.6B+ | {sms} | | | | ○ | ● | ○ | ◑ | ○ |
| | KugouPlayer | 11.4B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| Short videos | Douyin | 35.8B+ | {sms} | | {sms, ppwd} | {sms, fn, id} | ● | ● | ◑ | ○ | ◑ |
| | Kuaishou | 26.1B+ | {Wechat.SSO} | | | | ● | ● | ○ | ○ | ○ |
| | Haokanshipin | 7.0B+ | {Weibo.SSO} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| | Douyinhuoshan | 6.8B+ | {sms} | | {sms, ppwd} | {sms, fn, id} | ● | ● | ◑ | ○ | ◑ |
| | Weishi | 3.0B+ | | | | | ○ | ○ | ○ | ○ | ○ |

**Table V continued from previous page**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Meipaimv | 586.7M+ | | | | | ◑ | ● | ◑ | ○ | ○ |
| News | Jinritoutiao | 20.6B+ | {Douyin.SSO} | | {sms, ppwd} | {sms, fn, id} | ● | ● | ◑ | ○ | ◑ |
| | Tencent.news | 15.3B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Jinritoutiao.lite | 8.8B+ | {sms} | | {sms, ppwd} | {sms, fn, bcn, id} | ● | ● | ◑ | ○ | ◑ |
| | Netease.newsreader | 2.2B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Qukan | 2.2B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Sohu.newsclient | 1.8B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| Input method | baidu.input_huawei | 5.4B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Sougou.inputmethod | 5.0B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Iflytek.inputmethod | 3.2B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Ohos.inputmethod | 1.2B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Baidu.input | 800.0M+ | {sms} | | | | ○ | ● | ○ | ◑ | ○ |
| | QQpinyin | 172.2M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| Browsers | Baidu.searchbox | 25.4B+ | {sms} | | | | ● | ● | ◑ | ◑ | ○ |
| | Tencent.mtt | 21.4B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | UCMobile | 14.0B+ | {sms} | | | | ● | ○ | ○ | ○ | ○ |
| | Baidu.searchbox.lite | 2.3B+ | {sms} | | | | ◑ | ● | ◑ | ◑ | ○ |
| | Quark | 1.6B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Sogou.mobile.explorer | 470.9M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| Security management | Copydata | 9.4B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Shoujiguanjia | 4.5B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | QQpimsecure | 3.7B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | 360mobilesafe | 680.0M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | QQsafe | 290.6M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Liebao | 200.0M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| E-books | Dragon.read | 3.7B+ | {sms} | | {sms, ppwd} | {sms, fn, bcn, id} | ◑ | ● | ◑ | ○ | ◑ |
| | Kmxs | 2.9B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Kuaikan | 1.2B+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Weread | 866.3M+ | {WeChat.SSO} | | | | ● | ○ | ○ | ○ | ○ |
| | QQ.reader | 800.0M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Mdwz | 760.0M+ | | | | | ○ | ○ | ○ | ○ | ○ |
| Photography | Mtxx | 7.9B+ | {sms} | | | | ● | ● | ◑ | ○ | ◑ |
| | Huawei.videoeditor | 5.7B+ | {Huawei.acct} | | | | ● | ○ | ◑ | ● | ◑ |
| | Lemon.lv | 4.5B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Meiyancamera | 3.8B+ | {sms} | | | | ○ | ● | ○ | ○ | ◑ |
| | Snowcamera | 2.2B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Lemon.faceu | 1.7B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| App stores | Tencentappcentre | N/A | | | | | ○ | ○ | ○ | ○ | ○ |
| | Huawei.appmarket | N/A | {Huawei.acct} | | {sms, bcn} | {sms, bcn, id, fn} | ● | ○ | ◑ | ● | ◑ |
| | Vivo.appstore | N/A | | | | | ○ | ○ | ○ | ○ | ○ |
| | Xiaomi.market | N/A | {sms} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| | Oppo.market | N/A | {sms} | | | | ● | ● | ◑ | ◑ | ○ |
| | Baidu.appsearch | N/A | {sms} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| Utility tools | Wifilocating | 8.7B+ | {sms} | | {sms, id, fn, bcn} | {sms, bcn, fn, id} | ● | ○ | ◑ | ○ | ◑ |
| | Shoujiduoduo.ringtone | 5.6B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Mjweather | 3.7B+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Camscanner | 2.6B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | QQpim | 2.3B+ | | | | | ○ | ○ | ○ | ○ | ○ |
| | Kugou.ringtone | 825.1M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| Show ticketing | Taobao.movie | 249.0M+ | {sms, id(0, 4)} | {sms, id} | | | ● | ● | ○ | ○ | ○ |
| | Maoyan.movie | 151.8M+ | {QQ.SSO} | | | | ● | ● | ● | ○ | ○ |
| | Damai | 100.0M+ | {sms} | | | | ● | ● | ◑ | ○ | ○ |
| | Wandafilm | 71.9M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Mtime | 24.7M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Juqitech.niumowang | 24.0M+ | {sms} | | | | ● | ● | ◑ | ○ | ○ |
| Mobile banking | CCB | 9.4B+ | | | | | ● | ○ | ◑ | ○ | ● |
| | ICBC | 5.7B+ | | | | | ◑ | ○ | ◑ | ○ | ◑ |
| | BOC | 3.3B+ | | | | | ◑ | ● | ◑ | ○ | ● |
| | ABC | 2.1B+ | | | | | ● | ○ | ◑ | ○ | ◑ |
| | PSBC | 1.3B+ | | | | | ● | ○ | ○ | ○ | ◑ |
| | CMB | 1.3B+ | | | | | ● | ○ | ○ | ○ | ◑ |
| Investment and finance | Pinganlifeinsurance | 2.1B+ | {WeChat.SSO} | {sms} | | | ● | ○ | ◑ | ● | ○ |
| | Tonghuashun | 1.0B+ | {sms} | | | | ● | ○ | ◑ | ● | ◑ |
| | Sinafinance | 305.5M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Fengshouhulian | 148.3M+ | {QQ.SSO} | | | | ◑ | ○ | ◑ | ◑ | ○ |
| | Tiantianfund | 139.4M+ | | {sms, id, bcn} | {sms, bcn} | {sms, fn, id} | ● | ○ | ○ | ◑ | ○ |
| | PICC | 119.7M+ | {sms} | | | | ● | ○ | ◑ | ○ | ● |
| Sports and fitness | Huaweihealth | 20.6B+ | {Huawei.acct} | | {sms, bcn} | {sms, fn, id, bcn} | ● | ● | ◑ | ● | ◑ |
| | Keep | 2.4B+ | {fn, id} | | | | ○ | ● | ○ | ○ | ○ |
| | Tangdou | 641.5M+ | {QQ.SSO} | | | | ○ | ● | ○ | ○ | ○ |
| | Walkmonkey | 515.7M+ | {sms} | | | | ○ | ● | ○ | ○ | ○ |
| | Codoon | 356.8M+ | {sms} | | | | ● | ● | ○ | ○ | ○ |
| | Zhibo8 | 225.9M+ | | | | | ○ | ○ | ○ | ○ | ○ |

●: fully exposure, ◑: part exposure, ○: No exposure.

[1] LPR: Login password recovery, BCC: Bank card connection, PPR: Payment password recovery.

[2] pn: phone number, sms: SMS OTP + pn, e: email address, ep: email OTP, fn: full name, bir: birthday, id: ID number, bcn: bank card number.

[3] acct: account number and login password, ppwd: payment password.

[4] id(0, 6): the first 0 digit and the last 6 digits of the ID number.