Leaking the Privacy of Groups and More: Understanding Privacy Risks of Cross-App Content Sharing in Mobile Ecosystem

Jiangrong Wu
Sun Yat-sen University
wujr28@mail2.sysu.edu.cn

Yuhong Nan*
Sun Yat-sen University
nanyh@mail.sysu.edu.cn

Luyi Xing
Indiana University Bloomington
luyixing@indiana.edu

Jiatao Cheng
Sun Yat-sen University
chengjt6@mail2.sysu.edu.cn

Zimin Lin
Alibaba Group
shike.lzm@alibaba-inc.com

Zibin Zheng
Sun Yat-sen University
zhzibin@mail.sysu.edu.cn

Min Yang
Fudan University
m_yang@fudan.edu.cn

***Abstract*—*Cross-app content sharing* is one of the prominent features widely used in mobile apps. For example, a short video from one app can be shared to another (e.g., a messaging app) and further viewed by other users. In many cases, such Cross-app content sharing activities could have privacy implications for both the sharer and sharee, such as exposing app users' personal interests. In this paper, we provide the first in-depth study on the privacy implications of Cross-app content sharing (as we call *Cracs*) activities in the mobile ecosystem. Our research showed that during the sharing process, the adversary can not only track and infer user interests as traditional web trackers but also cause other severe privacy implications to app users. More specifically, due to multiple privacy-intrusive designs and implementations of *Cracs*, an adversary can easily reveal a user's social relations to an outside party, or unnecessarily expose user identities and her associated personal data (e.g., user accounts in another app). Such privacy implications are indeed a concern for app users, as confirmed by a user study we have performed with 300 participants. To further evaluate the impact of our identified privacy implications at large, we have designed an automatic pipeline named *Shark*, combined with static analysis and dynamic analysis to effectively identify whether a given app introduces unnecessary data exposure in *Cracs*. We analyzed 300 top downloaded apps collected from app stores in both the US and China. The analysis results showed that over 55% of the apps from China and 10% from the US are indeed problematic.**

I. Introduction

Cross-app content sharing (as we call *Cracs*) is one of the most frequently used features in mobile apps. Figure 1 shows a typical procedure of *Cracs*, in which Alice (as we call *sharer*) shares a privacy-sensitive video with Bob (as we call *sharee*). Here, Alice clicks the `Share` button in the video app YouTube (Step 1). The source app (YouTube) pops up a list of available sharing options and asks Alice which target app she would like to share (Step 2). After selecting the target app (i.e., Facebook Messenger) and the sharee (i.e., Bob), the shared content is displayed as a specific message in the chat box of the target app (Step 3). Essentially, the technical implementation of *Cracs* is to transfer a URL hosting the shared content from the source app to the target app.

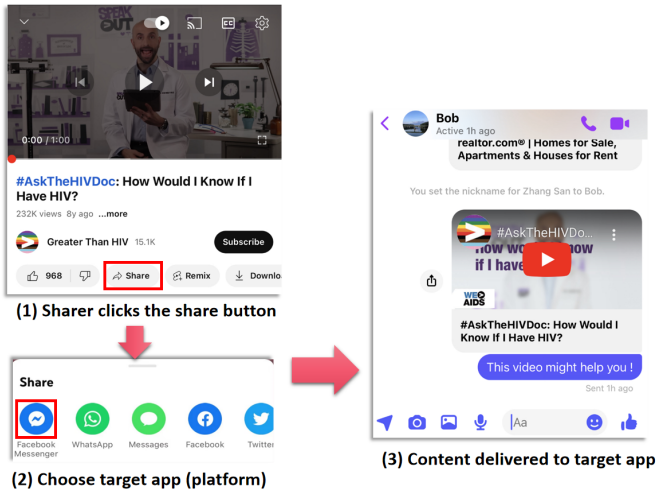

Fig. 1: The process of cross-app content sharing (*Cracs*).

In many scenarios of *Cracs*, the shared content may reveal users' highly sensitive data such as personal interests, political leaning, and even sexual orientation. Previous works have shown that traditional websites tend to implant specific trackers to monitor and collect users' online activities (e.g., browsing history), which raises severe privacy concerns to related users [47]. As countermeasures, various mechanisms are proposed to block such tracking activities in both academia [38], [35] and industry (e.g., AdBlock [8], AdLock [10], and AdGuard [9]) However, it is less clear how such tracking behaviors exist in the mobile ecosystem, particularly for privacy-sensitive scenarios such as *Cracs*. Moreover, with the increasing privacy protection enforcement such as GDPR [64],

* Corresponding author: Yuhong Nan

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.2024.24138
www.ndss-symposium.org

CCPA [3] and PIPL (privacy regulation in China) [21], little attention has been made to the data practice of *Cracs* in mobile apps.

Even worse, privacy implications in *Cracs* are more than just revealing privacy-critical personal information of the sharer. We find that improper design and practices of sharing activities in mobile systems introduce a new avenue to social relations of groups of people such as the content sharer with their sharees who viewed the content (see the threat details in Section III). In the scenario of *Cracs*, many source apps come up with illicit behaviors that can infer the social relations of individual groups of users, such as the content sharer and her/his friends (e.g., friends in WeChat), which otherwise are not available to the apps that host the shared content. The privacy implications concerning groups of people, such as social relations, can seriously go beyond the privacy impacts on individuals' privacy, as shown by recent studied [43], [52], with serious impacts on social good, public interest, and democratic ideals. For example, the privacy of groups of people is not supposed to be extended to surveillance/advertising activities against a racial, political, religious, or business group. Although prior research [67], [53] for mobile apps has extensively studied privacy threats to an individual user, little work has been done to understand privacy threats against users' social relations.

Our work. In this paper, we focus on such an imperative privacy domain in the less explored context of *Cracs* in the mobile ecosystem. To enable a systematic study and identification of the privacy risks in real-world apps, we first summarize the following two guidelines to achieve privacy assurance in *Cracs*: *data minimization* as required by privacy regulations [13], and *least privilege* [22] which is applicable for security and privacy protection.

To this end, we identified and generalized three patterns of privacy exposure behaviors in particular apps from which the contents are shared, namely, *sharing behavior tracking* (*SBT*), *sharing data interception* (*SDI*), and *sharer data exposure* (*SDE*). The problems have been introduced by diverse, improper design and data practices by app developers and supported by current mobile platforms. In these scenarios, the sharing activity goes beyond the sharer's original intention which shares the in-app content, but also exposes other sensitive information to adversaries which is fundamentally not necessary.

More specifically, with *sharing behavior tracking*, the popular content-hosting apps intend to track all users who viewed specific contents shared by the sharer, so the apps can infer social relations between the sharer and the groups of sharees. With *sharing data interception*, contents shared by the sharer are continuously monitored and collected by either the source app or third-party components the source app hosts. With *sharer data exposure*, the improper sharing activity over-exposing sharer's personal information of the source app to the sharee, such as personal homepage, browsing history records/posted content, and the following/followers list.

To further understand normal users' experience, perceptions, and susceptibility concerning *Cracs* in the wild and the underlying privacy threats we identified, we performed a comprehensive user study with 300 participants. Our study reveals that while *Cracs* is frequently used by mobile users, most of them place a high value on their private data, such as online identities, and shared content, particularly, their social connections. Additionally, we observed a significant decline in users' willingness to use the *Cracs* feature once they comprehended the actual privacy implications, such as the exposure of social relationships. (see Section III-A), and profiles accessed by strangers (see Section III-C).

Empirical measurement study. To enable a systematic study and identification of the privacy risks in *Cracs* in real-world apps, we have designed *Shark*, an automatic pipeline that can efficiently identify whether the sharing mechanisms in a given app indeed contain privacy leakage behaviors (i.e., *SBT*, *SDI*, and *SDE*). *Shark* is a combination of static and dynamic app analysis framework specifically designed to analyze sharing activities in Android apps. It is featured with a set of key components, allowing us to (1) accurately locate sharing APIs and the corresponding sharing widgets from the app user interfaces, (2) automatically trigger the sharing action from the entry point of the app (e.g., the app landing page), and (3) confirm whether a privacy leakage indeed happens. Particularly, we design a unique runtime testing scenario, which employs content inspection with differential analysis [54] to confirm the user identity indeed sent out with the shared content (see Section VI-E).

With *Shark*, we analyzed 300 top downloaded apps collected from app markets in both the US and China (i.e., 150 apps from each region, respectively), with a set of interesting discoveries. Our study shows among the 186 apps with *Cracs* functions, 74 of them have at least one privacy leakage pattern. These apps cover various app categories, such as short video social, shopping, news, and health. In the meantime, for popular apps in China, the scale of such a privacy leakage is significantly larger than US (i.e., 55.83% compared to 10.60%). In addition, we identified two widely used third-party libraries that aggressively collect users' sharing data. Due to the stealthy nature of the exposure pattern, such behaviors are barely noticeable to app users. Additionally, by manually inspecting the iOS version of identified apps, we confirmed that at least 58/71 (81.69%) of apps in iOS are affected by such privacy exposure behaviors as well.

Finally, we provide comprehensive discussions on how to mitigate and regulate such privacy implications from the perspective of various involved parties, such as app users, system vendors, as well as app markets. We believe these insights could help to eliminate the two general categories of privacy exposures related to *Cracs*, such as the inference of social relations and political/racial groups of mobile users.

Contributions. We summarize our contributions as follows:

- We provide the first study on privacy implications of cross-app content sharing in mobile apps. We introduce key new understandings and generalize three privacy-exposure venues that exist in the wild.
- We conduct a comprehensive user study to understand normal users' perceptions about *Cracs* and its underlying privacy threats.
- We propose *Shark*, a pipeline combined with static analysis

and dynamic analysis to detect various privacy-exposure venues in *Cracs*.

• We analyzed 300 top downloaded apps to reveal the pervasiveness of the privacy threats in *Cracs* in the real world.

Ethical considerations and responsible disclosure. The user study in our research was approved by the Research Ethics Committees of our institution(s). The user study is fully anonymous, and we did not collect any identifiable information about participants. All experiments and evaluations are performed with our own accounts which are specifically used for research. As responsible disclosure, we reported the identified risks to all related parties (app developers and third-party SDKs) through an email notification campaign. The content of the email can be accessed via Appendix B. Unfortunately, as of the final submission of the paper (Nov 2023), we have not received any feedback from them yet. We suspect that legal counsel would likely advise against responding with any acknowledgment or admission about the issues we raised.

Roadmap. The rest of the paper is organized as follows. Section II presents the background of *Cracs* and our research motivation. Section III highlights the data exposure scenarios identified in our research. Section discusses the privacy implications of *Cracs* (Section IV). Section V elaborates on the user study which showed app users' perceptions regarding *Cracs*. Section VI introduces the pipeline of *Shark*. Section VII shows our analysis results over the 300 top popular apps. Section VIII discusses the countermeasures. Section IX discusses related work and Section X concludes the paper.

II. Background and Problem Statement

In this section, we first lay out some background knowledge about *Cracs* on mobile systems across Android and iOS. Then we introduce our motivating example, as well as the scope of our research.

A. Background

Terminologies. We use the following terminologies to describe different subjects involved in *Cracs* process in (Figure 1): We call the user who shares content as *sharer* (e.g., Alice); we call the user who receives the shared content as *sharee* (e.g., Bob). We call the app that shares the content as *source* app (e.g., Youtube), and the app that receives the shared content as *target* app (e.g., Facebook Messenger). In *Cracs*, the source apps could be any app hosting customized content, such as YouTube, TikTok, and Instagram. The target apps are commonly social network apps such as Facebook Messenger, WeChat, etc.

Technical implementations of *Cracs*. To facilitate *Cracs*, there is a protocol between the source app (the shared content hosting app) and target app (social network app) to ensure that sharing data can be accurately and correctly transferred between the two apps. The shared data is mostly a URL referring to the shared content. which is further encapsulated under the specific data structure by the sharing SDKs. Finally, the shared content is transmitted between the source and target app via cross-app communication mechanisms provided by the system framework. For example, for Android apps, this is done via the IPC channel through Intent [11]. There are two types of SDKs that implement and support *Cracs*, including *target app SDKs* and *third-party SDKs*.

Typically, the vendor of the target app such as Facebook, or Twitter, provides a specific SDK, which integrates a set of APIs for sharing. When the user selects a specific target app, the source app prepares the content based on the SDK API of the target app. For example, to support sharing content from a source app to Facebook friends, the app must embed the Facebook sharing SDK [18] and invoke the corresponding API to deliver the content to the sharee (Facebook Messenger).

In the meantime, there are third-party SDKs that integrate multiple sharing SDKs of different target apps. This eliminates the tedious process of embedding multiple target app SDKs and invoking target app APIs with different formats, which is more convenient for developers of the source app. With such third-party SDKs, developers only need to invoke the third-party sharing API with the name of the target app as an additional parameter. Therefore, the third-party API automatically invokes the target app SDK API of the corresponding target app.

B. Problem Statement

The essence of *Cracs* is the URL of the shared content. Ideally, just like a simple web page hosting news posts, such shared content can be simply identified by an identifier of the shared content (e.g., such as content id, name of the resource). However, as we will further show in later sections, in the current design and implementations of *Cracs*, the actual URL for delivering the shared content often contains much more complicated data, which is not necessary for pinpointing the shared content itself. In this way, the process of *Cracs* exposes the user's sensitive data, and hence, raises various severe privacy implications.

Motivating example. Figure 2 shows a sample URL extracted from a real-world news app with more than 2.8 billion downloads* As can be seen, while the shared content can be identified by the URL's parameter `20230625A01ZKI00`, the sharing link also includes additional parameters such as user identity (`uid`). In this way, the app effectively records the view interests of both the shared content sender and receiver.

In the above motivating example, the source app explicitly violates the principle of data minimization as required by privacy regulations such as GDPR [64], CCPA [3] and PIPL [21]. For example, both Article 5(1)(c) of the GDPR and Article 4(1)(c) of Regulation (EU) 2018/1725 explicitly stated that *"personal data should be adequate, relevant and limited to what is necessary for relation to the purposes for which they are processed."* Given that parameters such as user ID can uniquely identify an app user, the shared content URL can be easily integrated with a tracker, allowing an adversary to monitor the sharing activities of a specific user.

Several previous research [69], [43], [54], [34] has shown that such tracking behaviors are highly privacy intrusive to users, as it brings various privacy implications such as information leaks [69] and data inference [43].

*We anonymize the app name to avoid potential ethical/legal issues.

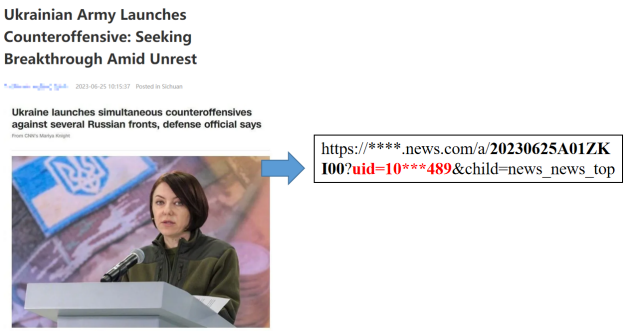

Fig. 2: A sample sharing URL extracted from the popular news app `App-CN-11`. In addition to the identity of the news (`20230625A01ZKI00`), the URL is also attached with other parameters that may expose user privacy.

Unfortunately, to the best of our knowledge, little has been done so far to understand the privacy implications of *Cracs*. More specifically, while previous research in mobile apps has discussed various user tracking and aggressive data collection scenarios [57], [54], the collection of sharing activities and its in-depth privacy implications are out of their scope.

To this end, our research aims to perform an in-depth analysis of the privacy threats of *Cracs* in the mobile ecosystem. Specifically, we want to seek answers to the following research questions.

- What are the possible privacy implications of *Cracs*?
- What are users' perceptions and responses regarding the data exposure in *Cracs*?
- How prevalent do such data exposure practices exist in the current mobile ecosystem?
- What are the possible countermeasures against the potential privacy risks?

Threat Model. In the process of *Cracs*, there are multiple parties involved, including app users such as sharer and sharee, the source app/service hosting the shared content, as well as sharing SDKs implemented by either the target app or third-parties.

We assume the adversary is a curious, opportunistic party that aggressively collects users' sensitive data which does not serve the purpose of completing *Cracs*. Therefore, any party that accesses or processes the shared content could be a potential adversary (the source app, target app, and third-party SDKs). Note that as we will further show in Section III-C, in certain scenarios, the adversary could also be the sharee, as she can obtain the sharer's private data which is not intended to share.

The typical victims under this assumption are sharer and sharee, who might be concerned about if their sensitive information (e.g., identities, personal interests, and social relations) is over-exposed to other parties without their awareness.

III. Privacy Threats in Cross-App Content Sharing

Privacy expectations. The root cause of privacy threats in *Cracs* is the privacy-intrusive design and implementation that violates two fundamental security principles, namely, data minimization [15] and least privilege [22].

- **Data minimization.** Only minimum information of the shared content should be shared between the sharer and sharee. The process of *Cracs* should only expose and transmit mandatory information for accessing the content itself. For example, the URL link that only refers to the shared content.
- **Least privilege.** Since the process of *Cracs* is done locally between the source app and target app with system-provided features, the back-end server of other parties (i.e., the source app, the target app, and third-party sharing SDKs) should not collect, or track information related to the shared content.

Following the above privacy expectations, in this section, we present three patterns of privacy threat we have identified in *Cracs*. Particularly, *sharing behavior tracking* (*SBT*), *sharing data interception* (*SDI*) and *sharer data exposure*(*SDE*). Note that these patterns are orthogonal to each other, meaning that a single app can have multiple privacy exposure patterns at the same time.

A. Sharing Behavior Tracking (SBT)

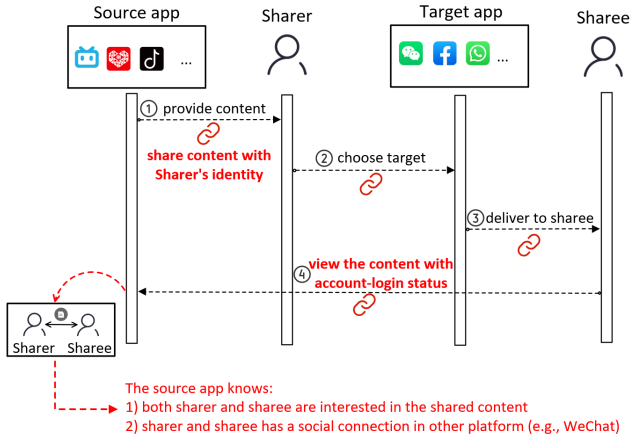

Fig. 3: The process of inferring social relationship via *SBT*. Once a tracker-enabled share link is accessed by two app users, the app's server can establish their social relationship.

There are two privacy-intrusive designs that the source app uses to track the sharing activities between the sharer and sharee, and further, infer their social connections.

- **Tracking the sharer.** The first one is to implant a tracker to the shared content, which is associated with the sharer's identity. More specifically, since the URL for *Cracs* is fully controlled by the source app, the source app can easily implant a tracker in the URL [54], allowing itself to persistently track the shared content with the sharer.

- **Tracking the sharee.** The second one is to track the sharee who viewed the shared content by requiring the user to log in to the source app. More specifically, when the sharee clicks the shared content in the target app, it either (1) pops up a login page, requiring the sharee to log in with an account of the source app, or (2) redirects the sharee to view the shared content in the source app if the app is already installed on sharee's device. In both cases, the source app can bind the content-viewing activity to the sharee.

Establishing social connections. Figure 3 shows how the source app establishes social connections between the sharer and sharee by exploiting the above-mentioned two tracking mechanisms. Since the shared content is associated with the identity information of both the sharer (via the tracker) and sharee (via the login status), the source app can easily know (1) the two parties are interested in the shared content, and (2) the two parties are with social connections in the target app.

Note that the connection between sharer and sharee (e.g., friends in the target app) is supposed to be confidential and not accessible to the source app. For example, the contact list of a WeChat user is confidential to any party other than WeChat by default. Unfortunately, the implementation of *Cracs* provides a perfect channel, allowing the source app to infer the sharer's social relations (a group of users) in the target app. In Section IV, we will provide a more detailed discussion on the privacy implications of leaking users' social relations.

B. Sharing Data Interception (SDI)

In *sharing data interception* (*SDI*), the adversary explicitly intercepts the shared content and sends it out to its own server. Note that such an interaction logic is fundamentally not necessary for *Cracs*, as the process of *Cracs* is done locally via system-provided IPC mechanisms between the two apps (as we discussed in Section II-A).

Listing 1: An example of *sharing data interception* (*SDI*).

```
1  public static void shareToPlatForm(int i,
       ShareEntity shareEntity) {
2    switch(i) {
3      ...
4      //Share the content to WeChat
5      SNSManager.getInstance().share2WX(shareEntity)
6      //Send the content to app's own server
7      ShareResultApi.shareResult(shareEntity)
8      ...
9    }
10 }
```

Listing 1 shows an example of this illicit behavior implemented by a third-party sharing SDK hosted on the source app. As can be seen, every time the sharer performs *Cracs*, other than the legitimate content-sharing process (i.e., invoking the sharing API `share2WX` at line 5), the SDK makes an extra network request to send out the shared content together with the user associated information, to its own server (i.e., invoking `ShareResultApi` at line 7). As we will further show in Section VII-A, in addition to third-party sharing SDKs, we observed some source apps are also with *SDI*.

C. Sharer Data Exposure (SDE)

In addition to *SBT* and *SDI*, where the adversary is the source app or third-party SDK, we have also identified a potential risk: the sharing URL may expose the user's sensitive data from the source app to the sharee. More specifically, in some cases, the sharer's identity information (e.g., user ID or account name) in the source app is unnecessarily exposed to the sharee through the shared content. As a result, a privacy-curious sharee can visit and access the sharer's account-related information (e.g., profile) in the source app. However, such information is not supposed to be exposed to sharee.

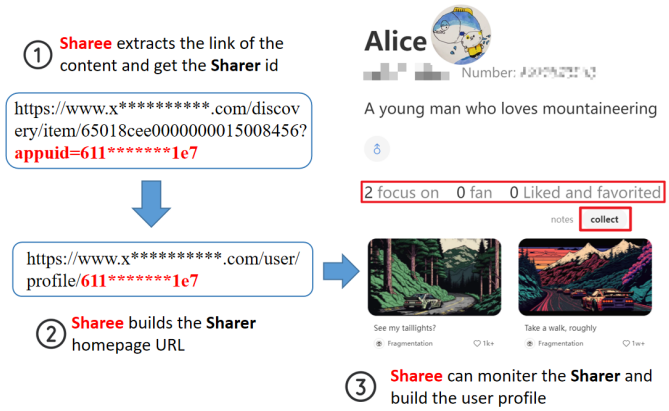

Fig. 4: Accessing the sharer's account of the source app by extracting her identity from the shared link.

As shown in Figure 4, an adversary (i.e., sharee in this case) with moderate technical background can easily obtain the URL of the shared content, and extract the user identifier (i.e., `uid` in this case) which is associated with the sharer **(step 1)**. Then the sharee can build a URL that gains access to the sharer's homepage in the source app **(step 2)**. As a consequence, the sharee can profile the sharer's account-related content in the source app, such as viewing her likes/collections, and following/followers **(step 3)**.

In addition, some app developers even directly display the sharer's username of the source app on the top of shared content, which explicitly exposes the sharer's personal information to the public (an example is shown in Figure 5). Based on this information, a normal person without any technical background can also access and monitor the sharer's account of the source app.

IV. Privacy Implications

Given the above three data exposure patterns in *Cracs*, we now discuss their privacy implications for app users.

A. User Profile Inference

Firstly, all three identified data exposure patterns provide additional channels for the adversary to profile and infer the user's sensitive information. Particularly, both *SBT* and *SDI* allow the source app and third-party sharing SDKs to collect and monitor users' sharing activities. In addition, the *SDE* provides a privacy leakage channel to the content sharer, allowing the sharee to access and infer additional information from

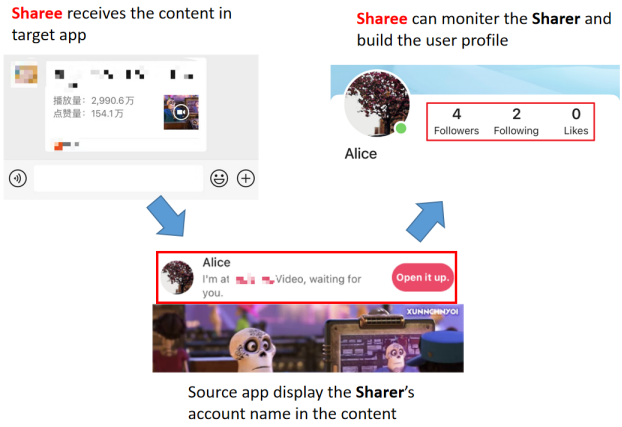

Fig. 5: Accessing the sharer's profile in the source app based on identity exposed with the shared content.

the sharer. Note that different from those profile information intentionally made public by some users via privacy settings, the capability of user profile inference in *SDE* is powered by the data association across different apps (from target app to source app), rather than the privacy settings of the specific platform. In other words, without *SDE*, an adversary may not have the chance to uncover the social network account of the victim sharer, whose account name is not a real name.

Previous research has shown that due to the advertising/marketing ecosystem, there are strong incentives for app developers to collect and picture users' demographic information [39]. In the case of *Cracs*, both *SBT* and *SDI* provide additional vectors for app developers or third parties to collect and infer users' demographic information. More specifically, leveraging the collected share activities of the app user, an adversary can easily infer the user's profile information such as the user's age, gender, political leaning, and even sex orientation [43], [65]. Actually, there is evidence indicating that advertising companies indeed utilize social circles to perform more precise, targeted campaigns [5].

B. Social Relation Inference

In addition to leaking and inferring the user's profile information, for all sharing activities with *SBT*, it naturally provides a perfect **side channel** to leak the user's social relation. Compared to the user's profile information, inferring the social relation of a particular user is more privacy intrusive. In other words, *SBT* enables the adversary to infer who are influencers (those sharing content) among a group of content viewers with specific interests in business, politics, or even religion.

Previous research has shown that social relation data is highly sensitive [33], [52], [63]. For example, the social link may infer political groups [52], target mobility [63]. In practice, in mobile devices, accessing similar information such as a contact list is strictly limited (i.e., as dangerous permission in Android [14]). Due to the extensive popularity and usage of social networks in modern society, relation information in social platforms can be even more valuable than data such as phone contacts.

As examples, the following scenarios highlight the impact of *SBT*, if an adversary continuously monitors the user's social relations over a certain period.

- **Identify friends of other social platforms** By listing all shared activities of user A via the social app WeChat, the app knows a subset of user A's friends in WeChat, as well as their corresponding accounts in the source app.
- **Identify family members** The adversary sees User A frequently shares kids' toys with User B, hence inferring that User A and B could be close family members.
- **Identify special relationships** The adversary sees User A frequently shares HIV prescriptions with User B and hence infers that both A and B may have a health issue with this particular disease.

While it is possible that for *SBT*, the sharee could choose to not open the shared content and safeguard her privacy, we consider this is less likely to happen. This is because in most cases, the sharer and sharee are in a mutually trusted environment (e.g., real-world friends). In addition, establishing such a connection is a one-time effort, which is particularly feasible for inferring social relations with high values (e.g., spouse, family members, and real-world friends) as such groups of people frequently use *Cracs* among themselves.

V. User Perception Analysis

With the privacy threats shown in Section III, this section reports our user study aiming at understanding mobile users' expectations and perceptions of the privacy risks associated with *Cracs*.

A. Design of the User Study

This user study seeks answers to the following research questions:

- RQ1: How prevalent are *Cracs* functionalities used by app users?
- RQ2: To what extent do users realize that their sharing activities result in privacy risks?
- RQ3: How do users perceive the sensitivity of their data in *Cracs*, and to what extent are they willing to tolerate the potential privacy threats?
- RQ4: How do users respond when they realize the real privacy implications within *Cracs*?

To make sure all participants understand the context and core objectives of our survey, we provide a detailed example to illustrate the *Cracs* scenario, at the beginning of the questionnaire. After that, we show 11 multiple-choice questions related to *Cracs* and ask the participants' opinions. Moreover, to better understand the context of the responses, we also collect basic demographic information about the participants, including their age range, and educational background at the end of the questionnaire (Q13 - Q16). We list the full questionnaire as well as the collected answers of the user study in Appendix A.

Recruitment and participant demographics. Under the Institutional Review Board (IRB) approval of our university, we leverage Wenjuanxing [28] (one of the most popular crowdsourcing platforms in China) to recruit online participants for our questionnaire. We asked each participant to finish the 16-question questionnaire, taking about 2.5 minutes with 2 RMB as compensation (comparable to the minimum hourly wage in our region). Our study only collects anonymized data. To ensure the quality of the collected responses, we intentionally added a Turing test question [25] to filter out potential robots (Q12).

According to the age range of the participants (i.e., Q13), 156/300 (52%) participants were from 18 to 30 years old, and 140/300 (46.67%) were from 31 to 45. In addition, more than 90% of the participants have a college degree or above (Q14).

Note that the percentage of our participants recruited from Wenjuanxing with a college degree or above is higher than the general public in China (37% of the population having an upper secondary education or above [16]). Therefore, we anticipate that our participants possess an equal or potentially greater level of knowledge and experience regarding privacy compared to the general public [48].

B. Results and Findings

In total, we collected valid responses from 300 participants in June 2023. The results showed that *Cracs* is actually commonly used in the wild. Although the participants were concerned about such privacy risks, they felt it is very difficult to recognize or avoid privacy exposure in their regular usage of *Cracs*, *Cracs*. We elaborate on the key findings from the user study as follows.

• **Finding-1: *Cracs* is commonly used by app users.** Based on the responses of Q1 and Q2, we see all of the participants use sharing functions in their daily lives. In the meantime, 36.34% of the participants share content at least once a week, and 94.67% of the users at least once a month.

• **Finding-2: App users are not fully aware of the data exposure of sharing activities.** Based on the responses of Q4, 86.22% of users stay in a login state by default before they use the sharing function provided by the source app. In such circumstances, adversaries can track the user more easily, which leads to *SBT*, *SDI*, *SDE* during *Cracs*. Furthermore, for certain apps that require sharers to log in before utilizing the sharing function, 59.67% of users opt to log in (Q5), thereby increasing the risk of potential privacy data leaks. Similarly, for certain apps that necessitate sharees to log in before accessing shared content, 65.33% of users choose to log in (Q6).

• **Finding-3: Users place a high value on their own private data, and deem privacy leakage that affects them as unacceptable.** We ask users three questions (Q10, Q7, and Q9) to determine their perception of whether *SBT*, *SDI*, *SDE* have caused privacy leakage for them. From the results, it can be seen that users strongly agreed there are privacy risks during *Cracs*. For example, 94% of users think that *SBT* actually causes them privacy leakage (Q10), and 63.67% of users feel unacceptable about the case when the source app obtains their social relationships of the target app (Q11). Similarly, 81% of people believe that *SDI* has caused privacy leaks for them (Q7), and 64% of users feel unacceptable when adversaries obtain their shared content (Q8). For *SDE*, 73% of users think that such a pattern also caused privacy leakage for them and feel unacceptable (Q9).

• **Finding-4: Users' willingness to share is significantly dropped after understanding the actual privacy implications of their sharing activities.** According to Q3, more than half of the participants (55.33%) would rather give up sharing to preserve their privacy. For the rest who would still proceed to share, the vast majority of participants (i.e., 29.33%) expressed that they are forced to give up privacy for *Cracs* due to the lack of alternative options. In addition, in response to Q5, 40.33% (121/300) of participants opted to refrain from logging in if it is mandated for *Cracs*. Similarly, 34.67% (104/300) of users choose not to view shared content if the source app requires them to log in before viewing the content.

VI. Analyzing *Cracs* data practices in Mobile Apps

Goal and procedure. To better understand real-world apps' data practices of *Cracs* activities, it is necessary to perform a large-scale measurement study over the top-popular mobile apps collected from app markets. However, manually exploring and confirming the sharing functions of each app is extremely time-consuming and tedious. Therefore, we have designed and implemented *Shark*, an automated framework, which enables us to effectively identify the three privacy exposure patterns (i.e., *SBT*, *SDI* and *SDE*) from a given Android app. Finally, we manually inspect and confirm whether an iOS version of the identified app is also affected by such privacy risks.

A. Challenges

Shared content tracking and API labeling. Similar to prior works for large-scale privacy leaks detection in mobile apps [49], [53], [58], a naive solution here is directly performing data-flow analysis to inspect whether it is the app indeed contains the three data exposure pattern (*SBT*, *SDI*, *SDE*). However, due to the differences in attack vectors and data exposure patterns, existing tools, even those specifically tailored for inter-component communication (ICC) data flow analysis [49], [61], [36], can not detect our problems.

For instance, *SBT* involves inferring social relations through a side-channel approach, rather than directly sending the social relationship to the server, which is the attack vector of the ICC privacy leak. Besides, the process of implanting the user's identity into the content link is on the Source app server side, which can not be analyzed by prior tools since they are all based on static analysis. In contrast, detecting whether a certain field in the shared URL allows the source app to identify groups of users' relations requires a differential analysis (Section VI-D). In addition, due to the pervasive usage of code-obfuscation techniques [23], in many cases, the sharing API could be obfuscated due to various reasons, such as intellectual property protection. For these apps, it is possible that the method names of sharing APIs will no longer exist in their app code.

Sharing activity triggering and leakage detection. To find out whether the privacy leakage pattern indeed exists in the app, it is necessary to first dynamically trigger those sharing

activities. However, this task is non-trivial as it requires accurately modeling a feasible path across two apps. Particularly, the path refers to starting from the source app's entry point (i.e., landing page) to the corresponding sharing function (e.g., the share button) and further finishing the sharing process in the target app (e.g., Facebook).

In addition, we need to confirm whether the shared content indeed contains any user-identifiable information from the triggered traffic. Existing techniques such as Query Parameter Stripping [27] can only identify those explicit trackers with specific parameter names (e.g., `user ID`, `google ID`). Unfortunately, the trackers in sharing links could be any customized identifiers (i.e., any random string label), which can not be directly identified based on the parameter names.

B. Workflow of Shark

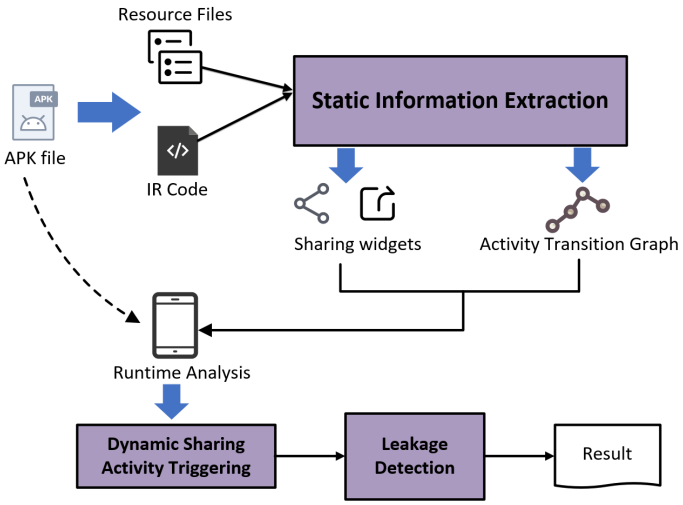

Fig. 6: The workflow of Shark.

To overcome the above challenges, we opt to use a mixed pipeline that integrates both static and dynamic analysis to detect the three identified patterns in *Cracs*. The workflow of *Shark* is shown in Figure 6, which mainly includes the following steps:

Static information extraction. *Shark* first uses static analysis to extract key information that is necessary for triggering sharing activities at runtime. More specifically, *Shark* parses both the decompiled app code and the UI resource files to locate the sharing buttons and their hosted activities (i.e., the UI for triggering sharing functions). Then, *Shark* constructs an activity transition graph, which contains a set of feasible path candidates to trigger and complete the sharing activity from the source app and the target app. In addition, *Shark* employs backward data-flow analysis to track variables related to the shared content. In this way, it can accurately build the connection between the share button of the source app and the target app (i.e., which button leads to WeChat, and which button leads to Facebook).

Dynamic sharing activity trigger. At runtime, *Shark* explores the app and triggers the sharing activities with information collected by static analysis. In the meantime, we intercept the concrete sharing related data, as well as its exposing contexts (e.g., sending destinations, parameters), and pass them into the leakage detection module to confirm the three exposure patterns.

Leakage detection. *Shark* has set up three different rules to confirm whether an app indeed contains privacy leakage based on the distinct features of the three data exposure patterns. Particularly, we have set up a specific testing scenario to detect *SBT*, *SDI*, and *SDE*. For example, we used differential analysis to identify whether a particular sharing link indeed contains user-identifiable information in detecting *SBT*.

C. Static Information Retrieving

Locating Sharing widgets. With a given source app, *Shark* traverses its UI resource files (i.e., the XML file describing app UIs) and extracts all the clickable widgets. Then, based on the widget id, *Shark* extracts the corresponding elements in the program code, i.e., the specific program variable whose value is assigned by the API `findViewById (element_id)`, as well as its corresponding event callbacks (e.g., the function `onClick`).

To identify which UI element is indeed related to *Cracs*, *Shark* performs a backward analysis from the content-sharing API to the top-level UI element. Given the number of sharing APIs is limited to a relatively small number of social network platforms (SDKs), we manually review their documentation and collect these APIs. The full list of APIs is shown in Table I. The backward analysis starts from the sharing APIs and ends with the event callbacks (i.e., onClick) of an UI element. To this end, we can clearly extract the data flow path and build the connection: `sharing widget -> event callback -> target app SDK`. Note that due to the inherent inaccuracy of static analysis, this process may report some false positives. For example, a UI element is incorrectly associated with a sharing API. However, such false positives will not affect the overall effectiveness of *Shark*, as our later dynamic (Section VI-D) only triggers those valid UI elements for *Cracs*.

To identify sharing API invocation from the obfuscated app code, we extract a set of obfuscation-resilient sharing API signatures from those apps that are not obfuscated. More specifically, we first extract the method body of each sharing-related API from apps without code obfuscation. By inspecting their implementation details, we extract the Android system-level APIs, class types of the parameters, and data flow information of the method body as invariants. We use such invariants to generate the signatures of the sharing APIs. In this way, our approach can further eliminate the potential false negatives caused by code obfuscation.

Constructing Activity transition graph (ATG). The Activity transition graph in *Shark* contains two parts: (1) the path from the source app entry point (i.e., main activity) to the activity hosting the sharing widgets. (2) an additional path that triggers the sharing function and completes the sharing function by selecting the target app. For the first part, *Shark* implements similar approaches as in previous work [44] to construct the activity transition graph for the source app that hosts the sharing content. For the second part, since

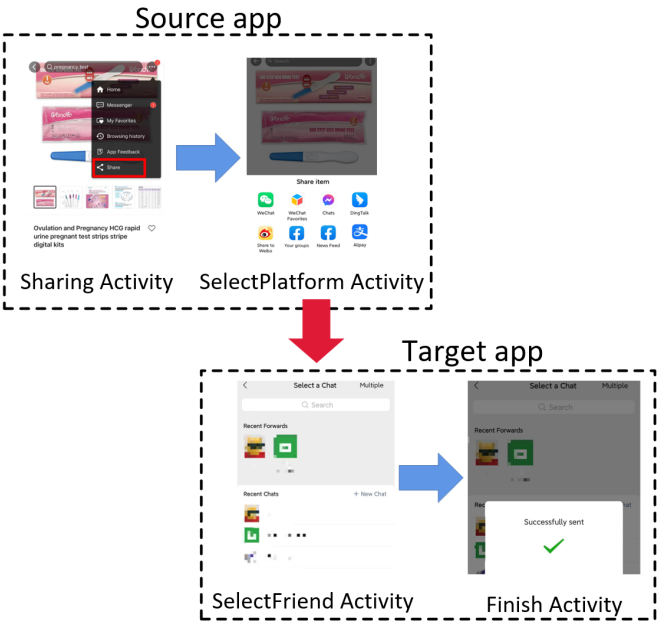

Fig. 7: An example of Activity transition between source app and target app.

the sharing destinations are specifically limited to a number of popular target apps (e.g., Facebook and WeChat), *Shark* explicitly adds an activity transition edge in the previously constructed graph between the source app and the target app. For example, in Figure 7, with the content shared to WeChat from the source app to the target app, *Shark* will add two edges into the ATG: the edge `SelectPlatform Activity -> SelectFriend Activity` and edge `SelectFriend Activity -> Finish Activity`. Based on this ATG, *Shark* will trigger the share action during the dynamic analysis process.

D. Dynamic Sharing Activity Triggering

At runtime, *Shark* takes the previously constructed Activity Transition Graph (ATG) to reach the widget that triggers the sharing function with the following two steps: Firstly, *Shark* traverses all available paths in the ATG until it reaches the desired activity, which hosts the sharing widget. Then, *Shark* simulates the click event to the sharing widget and completes the sharing process. Note that *Shark* utilizes the single-sign-on feature [29] (e.g., log in via Facebook) to perform an automated login attempt before sharing. If the automation fails (e.g., additional identity verification is required by the source app), we finish the rest login process manually. During the sharing process, the leakage detection module will help us to confirm all the patterns we discussed in Section III.

we use Xposed [7], a widely used dynamic app analysis framework, to hook the sharing APIs and extract the shared content (i.e., sharing link) from the specific parameters of the sharing API for future analysis. In the meantime, *Shark* sets up a network proxy, which automatically captures the network packets and data during the sharing process.

E. Leakage Detection

As mentioned earlier, *Shark* sets up a specific testing scenario to automate the sharing process to more effectively confirm whether the sharing activity indeed exposes user identity. During the automation, *Shark* will use different strategies to confirm all three data exposure patterns.

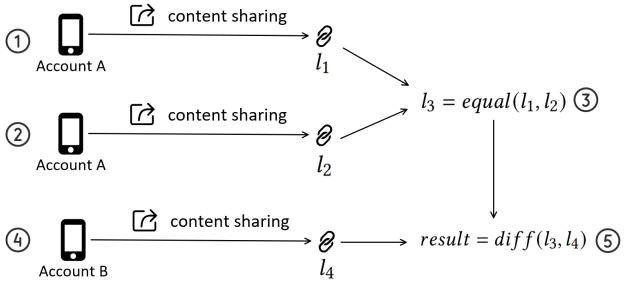

Fig. 8: Approach of *SBT* detection with runtime analysis.

***SBT* detection.** To detect *SBT*, we employ differential analysis to identify whether a shared link indeed contains user-identifiable information. we consider when different users share the same content, the link carries the shared content should remain consistent regardless of the user identity. In other words, if the same content shared by different users makes different sharing links, we suspect it is involved in *SBT*. To this end, our approach extends the idea from a previous work [54], which is specifically designed to detect unique tracking identifiers from the network traffic. Figure 8 shows the detailed process for this process, which consists of five steps:

1) We use account A to share specific content once and obtain the corresponding sharing link l_1.
2) We use account A to share specific content for a second time and obtain the corresponding sharing link l_2.
3) We identify parameters that have different values by comparing the two links l_1 and l_2. Such parameters are not user-identifiable data (e.g., timestamp information, sharing counters, etc.) because they are not consistent under the same user. We eliminate such parameters from the sharing link as a new link labeled as l_3.
4) We use account B to share the same content as we did in account A and obtain the sharing link as l_4.
5) We compare the parameter values between the two links l_3 and l_4. If there are still any parameters with different values, we consider them as potential user-identifiable information.

***SDI* detection.** Recall that the key pattern of *sharing data interception* (*SDI*) is the adversary unnecessarily sends out the shared content to the Internet. To this end, we check whether the sharing link indeed exists in our captured network traffic. To detail, we automatically extract the payload body of each network packet and extract their contained URLs. If the sharing link also exists in the captured network traffic, we consider the app exposed user's sharing activity via *SDI*.

***SDE* detection.** For *SDE*, we check whether the sharer's username or UID is presented in the shared content/link. We set up a specific testing scenario to detect *SDE*. More concretely,

TABLE I: API signatures used for sharing.

SDK Provider	SDK Sharing API
QQ	com.tencent.tauth.Tencent: void shareToQQ(android.app.Activity,android.os.Bundle,com.tencent.tauth.IUiListener)
Qzone	com.tencent.tauth.Tencent: void shareToQzone(android.app.Activity,android.os.Bundle,com.tencent.tauth.IUiListener)
Weibo	com.sina.weibo.sdk.openapi.a: void shareMessage(android.app.Activity,com.sina.weibo.sdk.api.WeiboMultiMessage,boolean) com.sina.weibo.sdk.share.ShareResultActivity: void onCreate(android.os.Bundle)
WeChat	com.tencent.mm.opensdk.openapi.BaseWXApiImplV10: boolean sendReq(com.tencent.mm.opensdk.modelbase.BaseReq)
DingTalk	com.laiwang.ding.share.openapi.ddshareopenapi.MainActivity: void sendWebPageMessage(boolean) com.laiwang.ding.share.openapi.ddshareopenapi.MainActivity: void sendTextMessage(boolean)
Facebook	com.facebook.share.model.ShareLinkContent$Builder: com.facebook.share.model.ShareLinkContent$Builder setContentDescription(java.lang.String) com.facebook.share.model.ShareLinkContent$Builder: com.facebook.share.model.ShareLinkContent$Builder setContentTitle(java.lang.String) com.facebook.share.model.ShareLinkContent$Builder: com.facebook.share.model.ShareLinkContent$Builder setQuote(java.lang.String) com.facebook.share.ShareApi: void share(com.facebook.share.model.ShareContent,com.facebook.internal.CollectionMapper)
Messenger	com.facebook.share.model.ShareMessengerURLActionButton: void writeToParcel(android.os.Parcel,int) com.facebook.share.model.ShareMessengerURLActionButton$Builder: com.facebook.share.model.ShareMessengerURLActionButton build()
Twitter	com.twitter.sdk.android.tweetcomposer.TweetComposer$Builder: void show()
Android System	android.content.ClipboardManager: void setText(java.lang.CharSequence) android.content.ClipData: android.content.ClipData newPlainText(java.lang.CharSequence,java.lang.CharSequence) android.content.Intent: void setAction("android.intent.action.SEND"); android.content.Intent: void setType("text/plain"); android.content.Context: void startActivity(android.content.Intent) android.content.Intent: void setAction("android.intent.action.SEND"); android.content.Intent: void setType("image/*"); android.content.Context: void startActivity(android.content.Intent)

for a source app, we have a pre-registered account that contains all the necessary account credentials (e.g., username, user id). Usually, this information can be found in the settings options of the source app. Then we use the account to share content automatically and extract the shared content link, if the shared content link contains the account credential, we consider the app has *SDE*. Note that if the UID is included in the shared link, *Shark* will report it as with *SBT* and *SDE* both. As discussed earlier in Section III, the UID not only can be used by the Source App to associate Sharer and Sharee, but it can also be utilized by malicious Sharees to infer Sharer's personal information.

In addition, for the source app that explicitly displays the username in the shared content (i.e., discussed in Section III-C), we use another account of the target app to automatically view the content shared from the source app. Then, we inspect whether the sharer's username is presented on the app by checking all texts shown on the UI.

F. Effectiveness of Shark

The effectiveness of *Shark* lies in whether it can trigger the sharing widgets in the given app. To this end, we randomly select ten apps from our dataset (see Section VII) as the evaluation dataset. We manually explored these apps and established the ground truth data – the available sharing functions and their corresponding UI widgets. The coverage rate of *Shark* is shown in Table II. Note that the design of *Shark* does not provide any coverage guarantee in detecting sharing leaks. Instead, we aim to significantly reduce the manual effort needed for such analysis.

In this experiment, we set up a time-out threshold of five minutes for each app. Among the 368 manually identified sharing widgets, our analysis showed that *Shark* can effectively trigger 306 of them, reaching a coverage rate of 83.15%. The average time taken for the whole exploration is 75.94s, with a maximum of 289.42s and a minimum of 10.36s. To this end, we consider *Shark* can effectively support our analysis for a larger number of apps.

In addition, recall that *Shark* employs an obfuscation-resilient mechanism to label the sharing APIs and improve its coverage (Section VI-C). For the 300 apps we measured in this research, we extracted 587 sharing APIs provided by different target app SDKs. Among them, 144/443 (32.50%) of them are with code obfuscation. We manually verify the obfuscated APIs we extracted and the precision is 98.2%, which proves our mechanism is effective in eliminating false negatives in detecting sharing APIs.

TABLE II: Effectiveness of *Shark* in triggering sharing activities.

App	# Trigger Share Action	# Total Share Action	Coverage
TestApp-1	32	32	100%
TestApp-2	40	48	83.33%
TestApp-3	4	4	100%
TestApp-4	76	96	79.17%
TestApp-5	16	16	100%
TestApp-6	16	25	64%
TestApp-7	48	64	75%
TestApp-8	24	28	85.71%
TestApp-9	30	35	85.71%
TestApp-10	20	20	100%
Total	306	368	83.15%

VII. Understanding Sharing Leaks in the Wild

Dataset. In our study, we collected 300 top apps from two representative countries (e.g., China and US) to observe the potential impact of privacy risks in the real world. Specifically, based on the number of downloads, we collected the top 150 apps from the Xiaomi app store [6], one of the largest app markets in China. In the meantime, we collected the top 150 apps from Google Play [4] under the region of the US. We chose such a data set because both China and the US are the two dominant mobile markets. Therefore, our analysis results can provide generic insights into the mobile ecosystem.

Measurement Setup. Our static analysis is implemented on an Ubuntu server with two Intel Gold 5218R CPUs (2.10GHz, 20 cores) and 512GB of physical memory; We run the experiment with 25 concurrently running instances of Soot with its latest version [24] and set 20 minutes timeout for each app. In dynamic app analysis, we use uiautomator2 [26] to explore the app with two Google Pixel 4 devices running on Android 10. Each app is given a five-minute timeout for triggering sharing activities.

In addition to analyzing Android apps, we also detected the corresponding iOS version of the apps in our dataset. Specifically, based on the identified Android apps, we confirmed whether the patterns also exist in their iOS version by analyzing the intercepted network traffic. The iOS apps are tested with iPhone 6 running on iOS 8.

As mentioned earlier in Section I (responsible disclosure), to avoid potential legal issues, we anonymized the names of apps and SDKs when presenting our results.

A. Measurement results

TABLE III: Identified apps with the three data exposure patterns (*SBT*, *SDI*, *SDE*).

Region	Type	# Apps with share function	# Confirmed pattern	Percentage
China	*SBT*	120	52	43.33%
	SDI		19	15.83%
	SDE		26	21.67%
	Total	120	67	55.83%
US	*SBT*	66	5	7.57%
	SDI		3	4.54%
	SDE		1	1.5%
	Total	66	7	10.60%

Total affected apps. In total, our analysis confirmed that among the 300 top apps we investigated, 186 apps have sharing functions, and 74 of them have at least one pattern. Table III presents the overall statistics regarding our analysis results in detail. Specifically, our static analysis module extracts a total number of 186 apps that have share functions, 120 apps from China and 66 apps from the US. Then, we manually confirmed that the candidate apps were indeed embedded and used one or more target app SDKs (i.e., exclude dead code). Later, the dynamic analysis confirmed that 57 apps indeed contain the pattern of *SBT*, 22 apps contain *SDI*, and 27 apps contain *SDE*. Our identified apps cover various app categories such as short video, shopping, news, health, and social contact. Besides, most of them are with more than 100M+ downloads. In addition, the pervasiveness of such patterns in iOS apps is consistent with those in Android, we will give more details later.

TABLE IV: Top 15 apps with *SBT* in China.

App Name	Package Name	Version	Downloads	Category
App-CN-1	c**.s*.a******.u**.a****	25.9.0	13.4B+	Short video
App-CN-2	c**.x******.p********	6.64.0	11.6B+	Shopping
App-CN-3	c**.s****.g*******	11.5.20.31491	7.6B+	Short video
App-CN-4	c**.t*****.t*****	10.25.10	6.3B+	Shopping
App-CN-5	c**.s***.w****	13.6.1	5.4B+	Social Contact
App-CN-6	c**.s******.m******	12.10.406	4.4B+	Life Style
App-CN-7	t*.d******.b***	7.35.0	3.6B+	Video social
App-CN-8	c**.s*.a******.a******.v****	7.6.4	3.2B+	Video social
App-CN-11	c**.t******.n***	7.1.60	2.8B+	News
App-CN-12	c**.x*****.x**	7.91.0	2.5B+	Social Contact
App-CN-13	c**.a******.v******	9.2.8	2.2B+	Shopping
App-CN-14	c**.n******.c*********	8.10.10	1.8B+	Music
App-CN-18	c**.m*.m***.m***	9.9.0.0	1.1B+	Photography
App-CN-19	c**.d*****.k***	10.23.23	990M+	Live
App-CN-20	c**.t****.w*******	15.0.1	980M+	Shopping

TABLE V: Apps with *SBT* in US.

App Name	Package Name	Version	Downloads	Category
App-US-1	s*.b***.l***	5.21.3	500M+	Live
App-US-2	c**.c****.e*****	2.165.0	100M+	Art & Design
App-US-5	c**.m***.r******	10.33.1	10M+	Home Renovation
App-US-6	c**.s********.p********	90.9.0	1M+	Beauty Fashion
App-US-7	o******.h***	20.43.0	1M+	Sports

Apps with *SBT*. Among the 186 apps that have sharing-related functions, 57 of them in total are with the pattern of *SBT*. Besides, we list the top affected apps in both regions in Table IV and Table V respectively. It can be observed that apps with huge downloads contain the *SBT* pattern, which means there are huge victims that leak their social relationships during *Cracs*. In addition, among all the apps affected by *SBT* that we detected, there are also apps specifically designed for pregnant people and some related to physical health. For these types of apps, social relationships are often even more important, as we previously mentioned in Section IV-B. Besides, the leakage rate in China is higher than US (43.33% vs. 7.57%), indicating the privacy threats in China could be more severe than US.

TABLE VI: Identified trackers in apps with *SBT*.

Type	# Total	Example
UserID	28	uuid=d5c8005151c4c1bae61c752c15832aad
Other Identifiers	31	lch=71c4705793bc7b94b uni=119200070407 trackid=5331964ef4hal6e38
IMEI/DeviceID	3	imei=ad7943e9-cdbc-4973-989e-e6d0e29b0639 device_id=1997182679927454

In Table VI, we list all user identifiers (trackers) extracted from the dynamic differential analysis. As can be seen, other than those well-known privacy-sensitive data such as `IMEI`, `DeviceID`, a large portion (50%) of user identifiers are actually not labeled with explicit parameter names, even higher than the proportion of `UserID` (45.16%). For example, `App-CN-6`, a daily life app including shopping and eating, with more than 4.4 billion downloads in China, uses the parameter name **`lch`** to identify a unique user. Such a data practice indicates a large number of apps leak user identifiers in a more stealthy manner.

• **Login requirement for sharing.** Another important observation in our study is that among the identified apps, we manually checked all the *SBT* apps and found that 19 out of 57 (33.33%) mandate the user to login before processing the sharing action or view the shared content. In this way, the app is actually forcing app users to make the decision between utility and privacy compromise. However, this requirement may not be sufficiently noticed by app users, as our user study indicates that most users prefer to log in before they share content or view the shared content. As a result, we consider the privacy leakage pattern of *SBT* to be more stealthy.

TABLE VII: Top 15 apps with *SDI* in China.

App Name	Package Name	Version	Downloads	Category
App-CN-15	c**.z****.a******	7.36.1	1.8B+	Social Contact
App-CN-21	c**.d*****.r***	5.1.3.32	950M+	Books & Reference
App-CN-23	c**.m***.m********	9.0200.02	710M+	Weather
App-CN-27	c**.s*.a******.a***	6.6.3	420M+	Car
App-CN-28	c**.y*****.d***	9.1.10	410M+	Education
App-CN-29	c**.d*****.c****.s****	5.4.3.1	310M+	Education
App-CN-30	c**.i****.n****	7.36.1	290M+	News
App-CN-31	c*.s******.a******	4.8.1	270M+	Social Contact
App-CN-32	c**.s**.a******.s*****	3.7.7	250M+	Social Contact
App-CN-33	c**.b*****.d****	7.5.7	190M+	Health & Fitness
App-CN-34	c**.l******.b****	2.68.0	160M+	Home Renovation
App-CN-35	c**.y****.a*******	10.62.0	150M+	Car
App-CN-36	c**.b****.t******	7.1.1	150M+	Books & Reference
App-CN-37	c**.h***.s*****	6.79.0	150M+	Shopping
App-CN-38	c**.d*****.f****	7.18.0	140M+	Social Contact

Apps with *SDI*. Based on the hostnames of the intercepted *SDI* traffics, we found that 22 apps collect users' sharing activities to their own server. Table VII and VIII shows the top apps with *SDI* in China and US respectively. While there are fewer apps with *SDI* in the US than in China, the number of its affected users is still significant.

TABLE VIII: Identified apps with *SDI* in US.

App Name	Package Name	Version	Downloads	Category
App-US-1	s*.b***.l***	5.21.3	500M+	Social Contact
App-US-3	w*.w******	9.78.1	100M+	Books & Reference
App-US-4	n**.z****.a******	7.38.7	100M+	Personalization

TABLE IX: Apps with *SDI* caused by third-party sharing sdk.

Package Name	Version	Downloads	Category	Third-party sharing SDK
c**.m***.m********	9.0200.02	710M+	Weather	SDK-1
c**.d*****.c****.s****	5.4.3.1	310M+	Education	SDK-1
c**.h***.s*****	6.79.0	150M+	Shopping	SDK-1
c**.k***.v**********	5.47.0.547002	140M+	VideoPlayer	SDK-2
c**.v******	6.1.10.211025	88M+	VideoPlayer	SDK-2

In the meantime, we identified two third-party sharing SDKs that also intentionally harvested users' sharing data. While it is reasonable for such SDK to collect users' sharing events for statistical analysis, such shared content collection practices are never motioned in their related official documents (e.g., privacy policy and terms-of-service). Besides, their data collection also includes users' unique identifiers such as `UserID` and `IMEI`. In total, 5 of our investigated apps are affected by these two SDKs and more than 1.3 billion downloads, we show the result in Table IX.

TABLE X: Top 15 apps with *SDE* in China.

App Name	Package Name	Version	Downloads	Category
App-CN-1	c**.s*.a******.u**.a****	25.9.0	13.4B+	Short video
App-CN-3	c**.s****.g*******	11.5.20.31491	7.6B+	Short video
App-CN-5	c**.s***.w****	13.6.1	5.4B+	Social Contact
App-CN-8	c**.s*.a******.a******.v****	7.6.4	3.2B+	Video social
App-CN-9	c**.k*****.a******	11.6.6	3.1B+	Music
App-CN-10	c**.t******.q******	12.3.5.8	2.8B+	Music
App-CN-12	c**.x*****.x**	7.91.0	2.5B+	Social Contact
App-CN-14	c**.n******.c*********	8.10.10	1.8B+	Music
App-CN-16	c*.k***.p*****	10.5.2.2	1.2B+	Music
App-CN-17	c**.t******.k******	8.10.150.278	1.1B+	Live
App-CN-18	c**.m*.m***.m***	9.9.0.0	1.1B+	Photography
App-CN-22	c**.b****.t****	12.42.5.0	780M+	Social Contact
App-CN-24	c**.s********.d****	5.19.0	640M+	Shopping
App-CN-25	c**.t******.w*****	8.98.0.588	610M+	Live
App-CN-26	c**.g*******.k***	7.52.1	590M+	Shopping

Apps with *SDE*. Table X list the top 15 affected apps in China. For the US apps, there is only one app named `App-US-1` that contains *SDE* pattern, which also contains *SBT* and *SDI*. Due to the huge downloads of the affected apps, lots of sharers will be monitored by the malicious sharees, including browsing history records/posted content, viewing the likes, and checking the following/followers list. As our user study indicates that such privacy exposure is unacceptable for the sharer (RQ3 Answer), we manually check that all the *SDE* apps will set the user's personal information (e.g., browsing history records, posted content) publicly by default. That means the malicious sharee can monitor the sharer more easily.

Prevalence of privacy exposure between the two countries. By comparing the number of pattern instances between apps in two different countries, we observe that apps in China are relatively higher. In other words, apps in the US market seem less privacy intrusive. Specifically, apps with *SDI* in China are more than six times as in the US (i.e., 19 v.s. 3). Our further investigation showed that one of the reasons is because the two third-party SDK (i.e, SDK-1, or SDK-2) is not integrated by US apps. Previous research has shown that Google Play usually enforces more strict app vetting policies than other third-party markets [50], [46]. We consider such enforcement may effectively block apps that integrate such SDK with aggressive data collection behaviors.

Privacy risks in iOS apps. As mentioned earlier in Section VI, we also inspect the data exposure in iOS apps. Due to the well-known challenges [2] in reverse-engineering and analyzing iOS apps, we opt to inspect a subset of iOS apps through the intercepted network traffic to inspect the pervasiveness of privacy leaks in *Cracs*. To detail, based on those identified Android apps with at least one of the three privacy leakage patterns, we obtained their corresponding iOS version if available. Among all the 74 problematic Android apps we identified, 71 apps have the corresponding iOS version.

By manually intercepting and inspecting the collected network traffic, we were able to confirm that at least 58/71 (81.69%) of the apps also with those privacy leakage patterns as their Android version. Such a pervasiveness of privacy leakage in iOS apps is not surprising, due to the fundamental technical similarity in implementing *Cracs* between the two platforms. Besides, to enable better analytics, the app server tends to collect data from both of the two platforms. Since apps may introduce additional data protection mechanisms (e.g., encryption), our result only shows the lower bound of the sharing data exposure.

B. Case Studies

Here, we highlight three representative cases from our identified apps, each of them covering one of the privacy leakage patterns as we discussed in Section III.

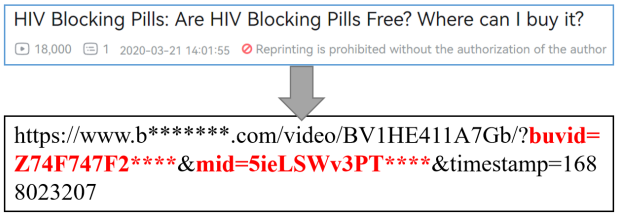

Fig. 9: Sharing URL extracted from the famous video app (`App-CN-7`) in China.

App with *SBT*. `App-CN-7` is a popular video app with more than 3.6 billion downloads (as of June, 2023) on our dataset. In this app, we found that the shared content (e.g., an HIV-related video) is always associated with a user tracker (i.e., `buvid` and `mid`). As illustrated in Section III-A, given the high popularity of this app, its privacy harm could be significant. For example, the video app can continuously track groups of people who are interested in a specific subject (HIV-blocking pills in this case).

SDK with *SDI*. `SDK-1` provides *Cracs* functions for the source app to deliver shared content to the target app. While it is merely designed to support implementing *Cracs* by integrating SDKs of different social network platforms, it aggressively collects and sends the shared content, together with the sharer identity to its own server. Figure 10 shows the detailed information collected by `SDK-1` once the user triggers

```
{
    "imgs": [
        "https://****.com/suo/image/Cw.webp?urlModule=templateCover"
    ],
    "text": "share a fun template for you, come to experience!",
    "attch": {
        "musicFileUrl": "https://****.com/share/template/index.html?id=8****1&trace_id=D******73090",
        "image": [
            "https://****.com/suo/image/Cw.webp?urlModule=templateCover"
        ],
        "title": "[template]",
        "url": "https://****.com/share/template/index.html?id=8****1&trace_id=D******73090",
        "content": "share a fun template for you, come to experience!"
    },
    "url": [
        "https://****.com/share/template/index.html?id=8****1&trace_id=D******73090"
    ]
}
```


Fig. 10: Shared data collected by SDK-1 via *SDI*.

Cracs. `SDK-1` is integrated by apps with a total number of 2.2 billion downloads in our dataset.

App with *SDE*. The short video app `App-CN-3` shares more than 7.6 billion downloads in China (as of June, 2023). The app is with *SDE* in both its sharing link and UI appearance. More specifically, the sharer's UID is explicitly attached to the shared video link. In addition, the user name of the sharer is shown on top of the shared content when viewed by the sharee. As a result, the sharee can easily identify the sharer's identity, and further access her profile information on `App-CN-3`. After 6 months after our first investigation, we find that in the updated version of `App-CN-3`, it no longer presents sharer's UI on its screen. While it is not clear whether the app developers change their design based on our email notification campaign, we believe the updated setup is a good data practice for *Cracs*.

VIII. Discussion and Countermeasures

Suggestions to different parties. Below we discuss how the mobile ecosystem can better protect user privacy against the potential privacy risks in *Cracs* activities.

- **App users.** Instead of using app-provided sharing mechanisms, a privacy-sensitive user can take screenshots and directly share the content image. In this way, since the shared content is not delivered via the adversary-controlled share link, the sharing activity becomes no longer traceable. However, as a trade-off, it may bring a negative impact on usability with additional user operations. For example, the content receiver might need to manually open the app hosting the shared content to access necessary information such as video clips and audio.

- **System vendors.** Since the sharing mechanism relied on framework-level support (e.g., `Intent` in Android) to deliver the sharing link across apps, it is possible for system vendors to intercept the sharing link and filter out the parameters which expose user identity. However, this mechanism is not a silver bullet, as an adversary can adopt extra effort to hide user identities in the URL created by herself. For example, embeds tracker as a fully randomized URL with the actual shared content. Also, this might not work well for apps requiring login status to access shared content.

- **Regulators and app markets.** More feasible and practical countermeasures can be adopted by regulators and App markets. Specifically, both authorities (e.g., FTC [19]) and app markets can explicitly state how app developers should comply with the privacy-by-design practices regarding users sharing data. In addition, app markets such as Google Play can adopt additional vetting mechanisms to identify such data patterns.

Limitations of our research. Our research only provides a lower bound to evaluate the severity of exposure patterns. Specifically, since the user study is conducted in China, it may not represent app users in other countries. However, previous research showed that people in developed countries such as the US tend to be more privacy-sensitive [1]. Due to the non-trivial challenges in app analysis, our pipeline has a few limitations in terms of effectiveness. For example, for the static information retrieving module (Section VI-C), *Shark* can not handle the UI elements that are loaded or created at runtime. Besides, *Shark* can not deal with apps that are protected by specific anti-debugging techniques (e.g., packers [20]), or the payload of network traffic is encrypted. We believe more advanced reverse engineering techniques could further improve the effectiveness of *Shark*.

IX. Related Work

Our research is closely related to the topics of privacy leakage analysis and privacy enhancement in mobile apps. The approach used for identifying privacy exposure is closely relevant to techniques for privacy leakage detection, runtime app analysis, and app testing.

Privacy leakage in mobile platforms. Privacy leakage has been a long-term issue in modern systems. Following this line of research, for mobile platforms, prior studies have identified various channels and attack vectors that expose user's sensitive data [57], [67], [45], such as privilege escalation [37], sensors on the mobile device [67], [40], [60], and third-party libraries [53] which are not well-regulated by the mobile system. Besides, while regulations such as GDPR provide better privacy enforcement, it takes time and incurs technical gaps for app compliance. To this end, a lot of researchers focused on detecting GDPR violations [54], [41], as well as inconsistencies between app behaviors and their privacy policies [30], [59], [62]. Compared to these works, privacy leakage patterns and privacy implications of content-sharing activities have been overlooked by prior research.

Techniques for leakage analysis and detection. Previous works have proposed various techniques to better analyze privacy leakage in the mobile ecosystem. For the Android platform, static analysis [55], [56], [42], dynamic analysis [66], [51], and their mixed approaches [68], [57] are extensively used for detecting privacy leakage in a more efficient way. For example, FlowDroid [31] is one of the widely used tools for detecting privacy leakage via static analysis. To overcome the limitations of static analysis such as high false positives, more recent research adopts runtime analysis by intercepting network traffic to identify privacy leaks [51], [58]. These approaches provide more accurate results as they can effectively trigger the actual program behaviors. To confirm the privacy

leakage pattern in a given mobile app, our approach adopts a similar traffic analysis mechanism as in [54].

There are a number of research and frameworks focusing on developing automated app exploration techniques for app testing [12], [17], [26], [32]. For example, recent work [44] put efforts into constructing a more refined activity transition graph, allowing an in-depth exploration of the mobile apps. Our research utilizes such mechanisms to establish the runtime testing path from the entry point to the sharing widget. In addition, existing works are mostly focusing on effectively exploring a single app. Instead, to effectively trigger and complete the sharing process, *Shark* establishes the activity transition across multiple apps by parsing and adding additional edges to the activity transition graph.

X. Conclusion

In this paper, we performed an in-depth analysis of the privacy implications of cross-app content sharing (*Cracs*) in mobile apps. We identified three types of privacy risks in which an adversary could collect and infer users' private information without her awareness. We conducted a comprehensive user study to understand users' perceptions regarding such privacy-intrusive data practices. The results showed that app users are indeed concerned about such data exposure patterns. To show the pervasiveness of the identified issues in real world, we designed a pipeline named *Shark*, combined with both static and dynamic analysis techniques, and analyzed the top popular mobile apps. Finally, we discuss the possible countermeasures which can be feasibly adopted by different parties. Our research highlights new types of privacy threats which have been ignored for a long time in the mobile app ecosystem.

Acknowledgement

We thank all anonymous reviewers for their valuable comments and feedback to improve this paper. Yuhong Nan is supported in part by the National Natural Science Foundation of China (62202510), Guangdong Basic and Applied Basic Research Foundation (2023A1515012971), and the Fundamental Research Funds for the Central Universities, Sun Yat-sen University (No.22lgqb26), and the Alibaba Innovative Research Program (AIR) of Alibaba Group.

References

[1] Americans consider certain kinds of data to be more sensitive than others. https://www.pewresearch.org/internet/2014/11/12/americans-consider-certain-kinds-of-data-to-be-more-sensitive-than-others/, 2014.

[2] Android vs ios – a comparison of open and closed source. https://blogs.pennmanor.net/1to1/2016/02/02/android-vs-ios-a-comparison-of-open-and-closed-source/, 2016.

[3] Ccpa: California consumer privacy act. https://oag.ca.gov/privacy/ccpa, 2021.

[4] Google play app market. https://play.google.com/store, 2021.

[5] How tracking social media shares can strengthen your campaigns. https://sproutsocial.com/insights/tracking-social-media-shares/, 2021.

[6] Xiaomi app store. https://m.app.mi.com/, 2021.

[7] Xposed:dynamic instrumentation toolkit for developers. https://repo.xposed.info/, 2021.

[8] Adblock. https://getadblock.com/, 2022.

[9] Adguard ad blocker. https://adguard.com/en/welcome.html, 2022.

[10] Adlock. your life without ads. https://adlock.com/, 2022.

[11] Android documentation: Intent object. https://developer.android.com/reference/android/content/Intent, 2022.

[12] Appium: Mobile app automation made awesome. https://appium.io/, 2022.

[13] Art. 9 gdpr: Processing of special categories of personal data. https://gdpr-info.eu/art-9-gdpr/, 2022.

[14] Contact list permission in android. https://developer.android.com/reference/android/Manifest.permission?authuser=1#READ_CONTACTS, 2022.

[15] Data minimization. https://edps.europa.eu/data-protection/data-protection/glossary/d_en#data_minimization, 2022.

[16] Education gps: China overview of the education system (eag 2022). https://gpseducation.oecd.org/CountryProfile?primaryCountry=CHN&treshold=10&topic=EO, 2022.

[17] Espresso: Android ui test. https://developer.android.com/training/testing/espresso, 2022.

[18] Facebook share sdk. https://developers.facebook.com/docs/sharing/android, 2022.

[19] Federal trade commission. https://www.ftc.gov/, 2022.

[20] Ijiami inc. http://www.ijiami.cn/, 2022.

[21] Personal information protection law of the people's republic of china (pipl). http://www.npc.gov.cn/npc/c30834/202108/a8c4e3672c74491a80b53a172bb753fe.shtml, 2022.

[22] Principle of least privilege. https://csrc.nist.gov/glossary/term/least_privilege, 2022.

[23] Proguard: Java obfuscator and android app optimizer. https://www.guardsquare.com/proguard, 2022.

[24] Soot - a framework for analyzing and transforming java and android applications. https://github.com/soot-oss/soot, 2022.

[25] Turing test. https://en.wikipedia.org/wiki/Turing_test, 2022.

[26] uiautomator2: dynamic exploration tool for android. https://appium.io/docs/en/drivers/android-uiautomator2/, 2022.

[27] url-tracking-stripper: an open-source chrome extension that will remove the tracking parameters from urls. https://github.com/newhouse/url-tracking-stripper, 2022.

[28] Wenjuanxing user study. https://www.wjx.cn/, 2022.

[29] What is single sign-on (sso) and how does it work? https://www.techtarget.com/searchsecurity/definition/single-sign-on, 2022.

[30] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words:{Entity-Sensitive} privacy policy and data flow analysis with {PoliCheck}. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 985–1002, 2020.

[31] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[32] Tanzirul Azim and Iulian Neamtiu. Targeted and depth-first exploration for systematic testing of android apps. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, pages 641–660, 2013.

[33] Michael Backes, Mathias Humbert, Jun Pang, and Yang Zhang. walk2friends: Inferring social links from mobility profile. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1943–1957, 2017.

[34] Trung Tin Nguyen Ben Stock, Michael Backes. Freely given consent? studying consent notice of third-party tracking and its violations of gdpr in android apps. In *In Proc. CCS*, 2022.

[35] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, and Brian Ziebart. Leveraging machine learning to improve unwanted resource filtering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pages 95–102, 2014.

[36] Amiangshu Bosu, Fang Liu, Danfeng (Daphne) Yao, and Gang Wang. Collusive data leak and more: Large-scale threat analysis of inter-app

communications. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 71–85, New York, NY, USA, 2017. Association for Computing Machinery.

[37] Sven Bugiel, Lucas Davi, Alexandra Dmitrienko, Thomas Fischer, Ahmad-Reza Sadeghi, and Bhargava Shastry. Towards taming privilege-escalation attacks on android. In *NDSS*, volume 17, page 19. Citeseer, 2012.

[38] Jonah Burgess, Domhnall Carlin, Philip O'Kane, and Sakir Sezer. Redirekt: Extracting malicious redirections from exploit kit traffic. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2020.

[39] Soteris Demetriou, Whitney Merrill, Wei Yang, Aston Zhang, and Carl A Gunter. Free for all! assessing user data exposure to advertising libraries on android. In *NDSS*, 2016.

[40] Michalis Diamantaris, Serafeim Moustakas, Lichao Sun, Sotiris Ioannidis, and Jason Polakis. This sneaky piggy went to the android ad market: Misusing mobile sensors for stealthy data exfiltration. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1065–1081, 2021.

[41] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. An empirical evaluation of gdpr compliance violations in android mhealth apps. In *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*, pages 253–264. IEEE, 2020.

[42] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pages 576–587, 2014.

[43] Haiqian Gu, Jie Wang, Ziwen Wang, Bojin Zhuang, and Fei Su. Modeling of user portrait through social media. In *2018 IEEE international conference on multimedia and expo (ICME)*, pages 1–6. IEEE, 2018.

[44] Wunan Guo, Zhen Dong, Liwei Shen, Wei Tian, Ting Su, and Xin Peng. ifixdataloss: a tool for detecting and fixing data loss issues in android apps. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 785–788, 2022.

[45] Julie Haney, Yasemin Acar, and Susanne Furman. " it's the company, the government, you and i": User perceptions of responsibility for smart home privacy and security. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 411–428, 2021.

[46] Yangyu Hu, Haoyu Wang, Tiantong Ji, Xusheng Xiao, Xiapu Luo, Peng Gao, and Yao Guo. Champ: Characterizing undesired app behaviors from user comments based on market policies. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 933–945. IEEE, 2021.

[47] Umar Iqbal, Charlie Wolfe, Charles Nguyen, Steven Englehardt, and Zubair Shafiq. Khaleesi: Breaker of advertising and tracking request chains. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2911–2928, 2022.

[48] Murat Kezer, Barış Sevi, Zeynep Cemalcilar, and Lemi Baruh. Age differences in privacy attitudes, literacy and privacy management on facebook. *Cyberpsychology: Journal of Psychosocial Research on Cyberspace*, 10(1):Article 2, May 2016.

[49] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 280–291. IEEE, 2015.

[50] Fuqi Lin. Demystifying removed apps in ios app store. *arXiv preprint arXiv:2101.05100*, 2021.

[51] Yabing Liu, Han Hee Song, Ignacio Bermudez, Alan Mislove, Mario Baldi, and Alok Tongaonkar. Identifying personal information in internet traffic. In *Proceedings of the 2015 ACM on Conference on Online Social Networks*, pages 59–70, 2015.

[52] Francesca Malloggi. The value of privacy for social relationships. *Social Epistemology Review and Reply Collective 6, no. 2 (2017): 68-77*, 2017.

[53] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. Finding clues for your secrets: Semantics-driven, learning-based privacy discovery in mobile apps. In *NDSS*, 2018.

[54] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share first, ask later (or never?) studying violations of {GDPR's} explicit consent in android apps. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3667–3684, 2021.

[55] Trung Tin Nguyen, Duc Cuong Nguyen, Michael Schilling, Gang Wang, and Michael Backes. Measuring user perception for detecting unexpected access to sensitive resource in mobile apps. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 578–592, 2021.

[56] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, Rui Shao, and Yan Chen. {FlowCog}: Context-aware semantics extraction and analysis of information flow leaks in android apps. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1669–1685, 2018.

[57] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 603–620, 2019.

[58] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374, 2016.

[59] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, Serge Egelman, et al. "won't somebody think of the children?" examining coppa compliance at scale. In *The 18th Privacy Enhancing Technologies Symposium (PETS 2018)*, 2018.

[60] Nazir Saleheen, Supriyo Chakraborty, Nasir Ali, Md Mahbubur Rahman, Syed Monowar Hossain, Rummana Bari, Eugene Buder, Mani Srivastava, and Santosh Kumar. msieve: differential behavioral privacy in time series of mobile sensor data. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 706–717, 2016.

[61] Jordan Samhi, Alexandre Bartel, Tegawendé F. Bissyandé, and Jacques Klein. Raicc: Revealing atypical inter-component communication in android apps. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1398–1409, 2021.

[62] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*, pages 25–36, 2016.

[63] Mudhakar Srivatsa and Mike Hicks. Deanonymizing mobility traces: Using social network as a side-channel. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 628–637, 2012.

[64] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[65] Svitlana Volkova, Glen Coppersmith, and Benjamin Van Durme. Inferring user political preferences from streaming communications. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 186–196, 2014.

[66] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1043–1054, 2013.

[67] Lichen Zhang, Zhipeng Cai, and Xiaoming Wang. Fakemask: A novel privacy preserving approach for smartphones. *IEEE Transactions on Network and Service Management*, 13(2):335–348, 2016.

[68] Xueling Zhang, Xiaoyin Wang, Rocky Slavin, and Jianwei Niu. Condysta: Context-aware dynamic supplement to static taint analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 796–812. IEEE, 2021.

[69] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1017–1028, 2013.

APPENDIX

A. User Study

Q1. Do you often use the *cross-app content sharing* feature provided by apps?

(A) Often (225/300, **75%**)
(B) Occasionally (75/300, **25%**)
(C) Never (0/300, **0%**)

Q2. How often do you share content using the *cross-app content sharing* feature on apps?

(A) Almost every day (35/300, **11.67%**)
(B) Once a week (74/300, **24.67%**)
(C) Once a month (175/300, **58.33%**)
(D) Almost never (16/300, **5.3%**)

Q3. If you were informed that using the *cross-app content sharing* feature on an app would leak privacy data, including shared content and personal identity information, would you still proceed?

(A) Yes, I don't care about privacy leaks (20/300, **6.67%**)
(B) Yes, because I have no other options (88/300, **29.33%**)
(C) No, I am willing to give up sharing for personal privacy (166/300, **55.33%**)
(D) No, I have other ways to protect my privacy (26/300, **8.67%**)

Q4. When using the sharing function within an app (such as TikTok or Taobao), do you log in to your personal account? For example, when sharing a TikTok video, have you already logged into your TikTok account?

(A) Yes (259/300, **86.33%**)
(B) No (8/300, **2.67%**)
(C) Not sure, I didn't notice (33/300, **11%**)

Q5. If an app's sharing function requires you to be logged in (i.e., if you are not logged into the app before sharing, a pop-up will ask you to log in), what would you do in this scenario?

(A) Refuse to log in and give up sharing (121/300, **40.33%**)
(B) Login to complete the sharing function (179/300, **59.67%**)

Q6. When receiving interesting content shared by your friends from other apps (e.g., TikTok) on social media applications (e.g., WeChat, Messenger), but the app (e.g., TikTok) requires you to log in before you can view the shared content, what would you do in this scenario?

(A) Refuse to log in and give up viewing the shared content (104/300, **34.67%**)
(B) Log in as required to view the shared content (196/300, **65.33%**)

Q7. When you do content sharing, the third-parties hosted in this APP (such as advertising, payment, and analytics services) also obtain your identity and shared content, Do you think this behavior constitutes a privacy leakage?

(A) Yes (243/300, **81%**)
(B) No (28/300, **9.33%**)
(C) I don't know (29/300, **9.67%**)

Q8. What do you think when third-parties obtain your identity and shared content?

(A) Acceptable (108/300, **36%**)
(B) Unacceptable, third-parties should not have access to my content sharing privacy (192/300, **64%**)

Q9. If you share content from a source app (e.g., TikTok, WeiBo, Instagram) with your friends on social media apps (e.g., WeChat, Messenger, WhatsApp), but your friends on social media apps (e.g., WeChat, Messenger, WhatsApp) obtain your account information from the source app (e.g., TikTok, WeiBo, Instagram), and can further view your history, followers, fans, and other information, would you find this acceptable?

(A) Unacceptable, as this is my private information (219/300, **73%**)
(B) It doesn't matter, I don't care (81/300, **27%**)

Q10. Do you feel violated if the APP can use your sharing activity to obtain your social relationships on WeChat (Social APP)?

(A) Yes (282/300, **94%**)
(B) No (18/300, **6%**)

Q11. In the following scenario, if a video app (such as Bilibili or Youtube) could use your video-sharing behavior to obtain your friend relationships on social media apps (such as WeChat or Messenger), what is your opinion?

(A) Based on the sharing function, the video app should obtain my friend relationships on other social applications (109/300, **36.33%**)
(B) The video app has violated my privacy because the sharing function does not require obtaining WeChat friend relationships (191/300, **63.67%**)

Q12. Attention Test (to exclude robots/random clicks): What should we say if we accidentally touch someone else in public?

(A) Thank you (0/300, **0%**)
(B) I'm sorry (300/300, **100%**)
(C) Hello (0/300, **0%**)
(D) You are welcome (0/300, **0%**)

Q13. What is your age?

(A) 18-30 (156/300, **52%**)
(B) 31-45 (140/300, **46.67%**)
(C) 46-60 (2/300, **0.67%**)
(D) Other (2/300, **0.67%**)

Q14. What is the highest level of education you have completed?

(A) Primary school (0/300, **0%**)
(B) Junior high school (4/300, **1.33%**)
(C) High school (11/300, **3.67%**)

(D) College (265/300, **88.33%**)
(E) Master or Ph.D. (20/300, **6.67%**)

Q15. What is your gender?

(A) Male (118/300, **39.33%**)
(B) Female (182/300, **60.67%**)
(C) Decline to answer (0/300, **0%**)

Q16. What is your professional background?

(A) STEM (Science, Technology, Engineer, Mathematics) (167/300, **55.67%**)
(B) Liberal arts (119/300, **39.67%**)
(C) Other (14/300, **4.67%**)

B. Email Template used for our notification campaign

To whom it may concern,

We are a security research group from Sun Yat-sen University, Fudan University, and Indiana University Bloomington. We would like to let you know about a potential privacy risk identified in the app {`Package`}.

Background: Cross-app content sharing is a basic functionality in modern apps and this feature is frequently used by mobile users. Specifically, like your app, the in-app content (e.g., a video post, product detail) can be shared to other social apps (e.g., WeChat, QQ) and further opened by other recipients of the social app user.

Unfortunately, we found that {`Package`} seems to unnecessarily implant user-associated unique identifier(s) in the shared content during the content sharing process. The collection of user identifiers here is not necessary from the perspective of sharing functionalities.

In the meantime, we found such data exposures (i.e., collecting the sharing-related data) are not explicitly mentioned in your app's privacy policies ({`Policy URL`}), which may lead to potential privacy incompliance required by regulators and bring app user confusion.

Note that we fully understand the observed data collection practice could be reasonable for marketing purposes, or it may caused by un-caution code implementation by app developers. What we want to point out is that, from the perspective of a privacy-sensitive user, it is recommended to keep as much data transparency as possible, and minimize the unnecessary privacy exposure.

With the above information, we would like to give the following recommendations:

(1) To better protect user privacy, you may perform additional, necessary code reviews on the app data practice in content sharing.

(2) To minimize compliance risks, you may better clarify the data usage purpose of sharing data/identifiers in the provided privacy policy.

Lastly, as a research project, part of our findings will be presented in The Network and Distributed System Security Symposium (NDSS) – a top-tier conference that fosters information exchange among researchers and practitioners of network and distributed system security. Information about this conference held last year can be accessed at https://www.ndss-symposium.org/ndss2023/. Note that we will not disclose any of your identity (such as app name, or affiliated company), we would kindly ask you to provide the following feedback if possible.

(1) Would you consider the above issue as a privacy risk?

(2) Could you provide us additional information on the actual purpose of such data collection, particularly, user identifiers?

(3) Would you take any action to alleviate this risk, such as updating the app code, or privacy policy statement?

We look forward to your reply, and please feel free to reach out for more details.

Sincerely,
Privacy Compliance Research Group