Collect Responsibly But Deliver Arbitrarily? A Study on Cross-User Privacy Leakage in Mobile Apps

Shuai Li*
lis19@fudan.edu.cn
Fudan University

Zhemin Yang*
yangzhemin@fudan.edu.cn
Fudan University

Nan Hua
huan19@fudan.edu.cn
Fudan University

Peng Liu
pxl20@psu.edu
The Pennsylvania State University

Xiaohan Zhang
xh_zhang@fudan.edu.cn
Fudan University

Guangliang Yang
yanggl@fudan.edu.cn
Fudan University

Min Yang
m_yang@fudan.edu.cn
Fudan University

ABSTRACT

Recent years have witnessed the interesting trend that modern mobile apps perform more and more likely as user-to-user platforms, where app users can be freely and conveniently connected. Upon these platforms, rich and diverse data is often delivered across users, which brings users great conveniences and plentiful services, but also introduces privacy security concerns. While prior work has primarily studied illegitimate personal data collection problems in mobile apps, few paid little attention to the security of this emerging user-to-user platform feature, thus providing a rather limited understanding of the privacy risks in this aspect.

In this paper, we focus on the security of the user-to-user platform feature and shed light on its caused insufficiently-studied but critical privacy risk, which is brought forward by <u>c</u>ross-user <u>p</u>ersonal data <u>o</u>ver-delivery (denoted as XPO). For the first time, this paper reveals the landscape of such XPO risk in wild, along with prevalence and severity assessment. To achieve this, we design a novel automated risk detection framework, named `XPOChecker`, that leverages the advantages of machine learning and program analysis to extensively and precisely identify potential privacy risks during user-to-user connections, and regulate whether the delivered data is legitimate or not. By applying `XPOChecker` on 13,820 real-world popular Android apps, we find that XPO is prevalent in practice, with 1,902 apps (13.76%) being affected. In addition to the mere exposure of diverse private user data which causes serious and broad privacy infringement, we demonstrate that the XPO exploits can invalidate privacy preservation mechanisms, leak business secrets, and even restore the sensitive membership of victims which potentially poses personal safety threats. Furthermore, we also confirm the existence of XPO risks in iOS apps for the first time. Last, to help understand and prevent XPO, we have responsibly launched two notification campaigns to inform the developers of the affected apps, with the conclusion of five underlying lessons from developers' feedback. We hope our work can make up for the deficiency of the understandings of XPO, help developers avoid XPO, and motivate further researches.

*The first two authors contributed equally to this research.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3559371

CCS CONCEPTS

• **Security and privacy** → *Social network security and privacy.*

KEYWORDS

Mobile Privacy, Cross User, Privacy Leakage, Information Loss Analysis, Security Assessment

ACM Reference Format:
Shuai Li, Zhemin Yang, Nan Hua, Peng Liu, Xiaohan Zhang, Guangliang Yang, and Min Yang. 2022. Collect Responsibly But Deliver Arbitrarily? A Study on Cross-User Privacy Leakage in Mobile Apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3559371

1 INTRODUCTION

Nowadays, mobile technique has become an indispensable part of our daily life. It offers millions of mobile applications (apps for short) with rich functionalities, and significantly shortens the distance of communication between people. A recent interesting trend is that modern mobile apps perform more and more likely as *user-to-user platforms* (or social platforms) where app users can be freely and conveniently connected.

This user-to-user platform characteristic is becoming increasingly popular in practice. It is being involved and reflected in many *regular* mobile apps (not even mentioning real social apps). For example, the popular video app YouTube does not only allow mobile users to watch videos on their smartphones, but also let them conveniently touch each other and build connections, such as checking other users' information (e.g., user profile and public play-lists) and discussing video content with other users.

However, up to now the security of this user-to-user platform feature is still rarely concerned. In particular, for the purpose of

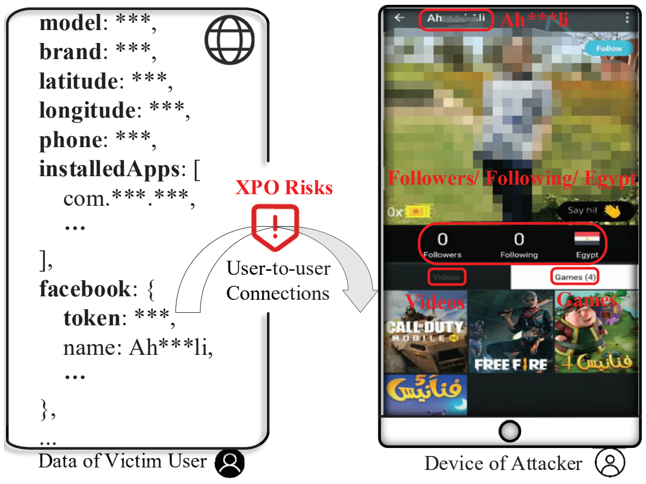


Figure 1: XPO Risk on the Vulnerable Platform of W.

helping users build strong connections between each other, mobile apps often design and provide a variety of user-to-user connection channels, such as messaging (often built in mobile apps), exhibited and public user profile, interested topics, social groups, and app-specific content. Considering the richness and diversities of the essential content these channels offer, an important security question naturally arising is whether sensitive user data is exposed during user-to-user connections or not.

To confirm this security concern, first of all, we conduct an empirical study on a set of real-world popular apps. Surprisingly, we find serious privacy risks commonly exist during user-to-user connections. By abusing and exploiting (vulnerable) user-connection channels, a remote attacker, pretending and attacking as a regular app user, can easily attack all available users on vulnerable user-to-user platforms, and cause severe privacy infringement. Consequently, a variety of personal data may be compromised by such remote attacks, including user real name, phone number, location, and even secret account data. Using leaked data, the attacker can even create and draw a detailed picture for a victim user.

We have noticed that app developers and the whole development and security communities barely pay attention to and even totally ignore the security of user-to-user connections. Thus, the data-delivery processes among users are often casually designed by app developers without security considerations, i.e., not following the '*data minimization*' security principle as regulated in GDPR [22]: "personal data shall be adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed". For convenience, we refer to this insufficiently studied but critical privacy risk in user-to-user connections as cross-user personal data over-delivery (XPO).

To demonstrate the XPO security issues, we take a real-world XPO vulnerability found in a high-profile vulnerable app (discovered by our above study) for example. As strongly requested by the app developers, we anonymize this app and name it W for convenience. W is an entertainment- and game-platform app with 10,000,000+ installs. In this app, a user can conveniently log in with SSO (Single Sign-on) using Facebook account, and easily touch other users in multiple channels, e.g., finding like-minded people who like the same games. As shown in Figure 1, an adversary can easily exploit these channels to access more and sensitive data[1] of any user than what is displayed in this app, including model and brand of user smartphone, precise geographical location, phone number, installed app list, and even app account token. Namely, what app W actually delivers from other users to the adversary is not consistent with what is shown on the app's UI of the adversary. What actually has been delivered contain more sensitive information about victim users but not shown on UI. And thus, the adversary can harvest the underlying (but hidden from UI) sensitive information of victim users. In our test targeting on our own account, via the stolen token, the adversary can log in to W with the victim user account and further perform malevolent behaviors, e.g., squandering the victims' money to reward his or her own game videos.

After understanding the serious security consequences of XPO issues, we intend to assess its security impacts in (a large number of) real-world apps. For this purpose, we need to design an automated vulnerability detection system against XPO issues. Nevertheless, we find this is not an easy task. On the one hand, it is difficult to precisely identify expected private user data (belonging to victim users) from various data delivered during user-to-user connections, including data belonging to things or places that should be insensitive (e.g., latitude and longitude of a shopping center), and personal data but not belonging to victim users. On the other hand, although the most convincing manifest of XPO risks is the inconsistency between the delivered and presented personal data of victim users (developers usually know what data should be presented; see details in §2.2), it is challenging to identify and determine the inconsistency. The reason is simply comparing the literal difference between the data shown to users and the delivered raw data may not work, as developers often adopt customized transformations or formats according to diverse service logics and development styles.

Existing work (e.g., [2, 12, 28, 29, 33, 54, 57]) fail to address above two challenges, and are hardly extended to achieve our goal, i.e., the prevalence and severity assessment on XPO in wild. Therefore, we design a novel automated detection approach, called `XPOChecker`, that can scalably and precisely vet real-world mobile apps against the XPO issues. For this purpose, `XPOChecker` leverages the advantages of program analysis and machine learning techniques. In particular, first `XPOChecker` designs and utilizes a bi-directional program slicing technique to analyze the typical process of user-to-user connections and initialize related potentially sensitive data. Then, `XPOChecker` applies correlation-based learning-assist privacy discovery to differentiate expected personal data from mixed data sets, e.g., potentially sensitive data but belonging to things, places, user own. Last, `XPOChecker` conducts an inconsistency-directed risk detection to regulate and determine whether delivered personal data violates the data minimization principle or not.

Relying on `XPOChecker`, we conduct our security assessment on 13,820 real-world popular apps collected from the official Android app marketplace (i.e., Google Play). Our assessment results show that XPO is unexpectedly prevalent, with 1,902 (13.76%) apps being

[1]Note that to protect user privacy, sensitive user data is replaced with fake values or masked before being shown, and so as the rest in this paper.

vulnerable. Both high-profile and long-tail apps are affected and the total installs of affected apps reach 16.89 billion, which suggests that numerous users are at risk and XPO has broad security impacts. In addition to exposing diverse private user data which cause serious and broad privacy infringement, the XPO exploits is demonstrated to be able to invalidate privacy preservation mechanisms, leak business secrets, and even restore the sensitive membership of victims which poses personal safety threats. As an example, we find an communication app C (anonymized as requested) leaked the sensitive cell location of more than 30,000 inmates held in 130 prisons or detentions in the US. By linking the leaked data, the adversary can infer whether the target inmate is a member of a specific cell in a particular prison, who are the cellmates of the target inmate, how many cells are in the given prison and even the lower bound of its cells' sizes.

Besides, by manually checking the iOS counterparts of the verified vulnerable apps during the validation of `XPOChecker`, the existence of XPO risks in real-world iOS apps is also confirmed for the first time. The identified XPO risks have received confirmations from app vendors, e.g., Strava, Opera and Smule. Last, to help understanding and preventing XPO, we have responsibly launched two notification campaigns to inform the developers of the affected apps, with the conclusion of five underlying lessons from developers' feedback. We hope our work can make up for the deficiency of the understandings of XPO, help developers avoid XPO, and motivate further researches.

In summary, our contributions are outlined as follows:

- In real-world apps, the under-studied but critical XPO risk is comprehensively and systematically analyzed, which facilitates the community's understandings in this regard.
- We design and implement `XPOChecker`, which overcomes several non-trivial technical challenges to automatically identify XPO risks.
- The landscape and severity of XPO in wild is unveiled with security assessment on 13,820 real-world apps. We responsibly inform the affected app developers, and conclude five helpful and meaningful lessons for mitigating XPO.

2 PROBLEM STATEMENT

2.1 Typical Process of XPO

By looking into deeply the app W introduced before, we present the typical process of XPO in Figure 2. During user-to-user connections, two different roles of users are involved, namely the **current user** and **other users** (i.e., the victim users). When the current user build connections with other users, e.g., viewing their user profiles, the personal data of other users is first delivered (①) to the mobile device of current user, then parsed (②) into specific data structure, e.g., a Java class named "`User`" in app W, next transformed (③) with various functions (e.g., "`String.replaceAll()`") and finally presented in UI or not.

During this process, the delivered personal data belonging to other users can be inconsistent with the presented ones. Typically, this kind of inconsistency has two forms. The first one is that the delivered personal data of other users is not presented in apps. The second one is that the delivered personal data of other users is presented in apps but being transformed to be coarse grained (e.g., the

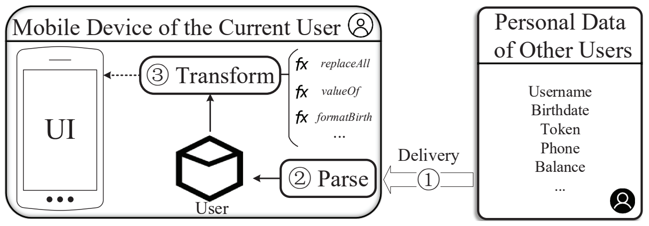


Figure 2: The typical process of XPO.

delivered email address of one other user is "`alice@gmail.com`" while the correspondingly presented one is "`a***e@gmail.com`"). Generally, both of these two forms of inconsistency will enable the attacker to access more information of the personal data belonging to other users. In this work, we take this inconsistency as the criterion of regulating XPO. More details are discussed below.

2.2 Vulnerability criterion & Definition

The inconsistency between the delivered and presented personal data of other users is chosen as the vulnerability criterion on the basis of two aspects. On the one hand, from the perspective of users, if they find inappropriately presented personal data, they can send feedback to app developers for asking adjustment. Besides, they can even uninstall the target mobile app with a request to delete all their personal data, which is protected by privacy-preservation laws[2]. Thus, the presented personal data within the target mobile app can be regarded as being acquiesced by the corresponding users.

On the other hand, from the perspective of app developers, they are typically the ones who are most familiar with the provided service and clear about what should be presented. Hence, it is believed that the presented user data can be regarded as in line with *"data minimization"* and if the delivered personal data of other users is inconsistent with it, then XPO risks are brought in.

We emphasize that whether certain personal data should be displayed and to what granularity should it be presented for offering specific services is out of the scope of this work. It should be noted that mobile users and app developers can have different metrics of judging privacy, and no relevant standards or regulations are available to refer to.

Based on the selected criterion, here we formally define XPO, which is inspired by information theory. E_x is denoted as the information entropy of x. Therefore, regarding the personal data of other users, we name the entropy of presented personal data as $E_{presented}$ and the entropy of delivered personal data as $E_{delivered}$. The formal equation of judging XPO risks is as follows:

$$E_{presented} \neq E_{delivered} \tag{1}$$

2.3 Threat Model

With the criterion and definition of XPO, this paper further introduces the following threat model of XPO.

Adversary. Regarding XPO, we consider the current user as the adversary who is curious about the personal data of other users

[2]For example, users are protected by "Right to be forgotten" as stipulated by GDPR [21] and "Right to delete" as requested in CCPA [9].

(i.e., the victim users) but not satisfied with only the presented information in-app.

Adversary goals. The goal of the adversary is to access as much personal data as possible. Generally, the more personal data of other users the adversary obtains, the greater privacy risks the other users will face.

Adversarial capabilities. We assume that the adversary has complete control of his or her own mobile device, which means the adversary can arbitrarily monitor cross-user mobile traffic, decompile the target mobile app or instrument mobile app code to access the delivered personal data of other users. Besides, the underlying operating system and Java runtime are considered as trusted and not compromised.

3 METHODOLOGY

3.1 Design Overview

According to Equation 1, the general idea to identify XPO risks is two-fold. First, we need to locate the delivered personal data of other users in mobile apps. Then we need to figure out whether they are consistent with what are presented. One naive way is that we can automatically exercise the target app with the dynamic app testing. Then, by extracting the network traffic along with relevant UI, we can compare their values to check if any concerned inconsistency exists. Nevertheless, the dynamic app testing is greatly limited in code coverage [11, 26, 38, 52]. Thus, applying dynamic analysis on a large-scale mobile apps to identify XPO risks would bring considerable false negatives and we adopt static analysis instead.

Following the general idea, this paper proposes a static tool named `XPOChecker` to automatically identify XPO risks. As shown in Figure 3, `XPOChecker` first performs the automated Pre-analysis which utilizes a bi-directional program slicing technique to analyze the typical process of XPO and initialize candidate sensitive data. Then, since the semantics of candidate data can be a reliable sign of user data and the data of other users normally deviates from the one of the current user, `XPOChecker` applies correlation-based learning-assist Privacy Discovery to automatically differentiate the personal data of other users from mixed data sets, e.g., potentially sensitive data but belonging to things, places or the current user. Last, from the insight that the mathematical essence of the concerned inconsistency is non-injective property, `XPOChecker` conducts an inconsistency-directed Risk Detection to automatically regulate and determine whether delivered personal data violates the data minimization principle or not.

Since our work mainly targets Android, `XPOChecker` is implemented for Android. Similarly, our methodology also works for other platforms such as iOS. The details of `XPOChecker` are presented in the following.

3.2 Pre-analysis

To identify the delivered personal data of other users, we first need to figure out how it is managed within mobile apps. Specifically, during the empirical study we performed, we randomly sampled 50 apps from the top 500 apps of each app category in Google Play store [14] and pick out 107 apps that have the personal data of other users in them by manual check. After an investigation into them, we have the following observation.

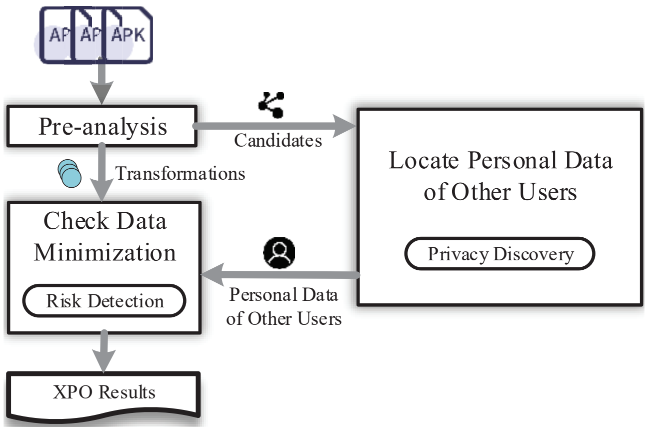


Figure 3: Basic workflow of `XPOChecker`.

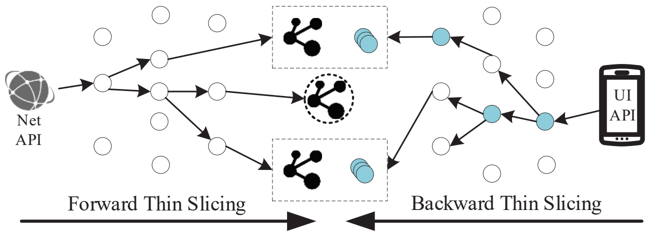


Figure 4: Bi-directional Thin Slicing.

Instead of being scattered everywhere in mobile apps, the delivered user personal data is typically managed within centralized data structures (76.85% apps adopt Java class, 12.04% manage user data in webview, 7.41% use react javascript code, 2.78% employ specific frameworks, e.g., Cordova, and 0.93% adopt JSONObject), which is convenient for further usages. Based on this observation, we can spot the superset (i.e., candidates) of other users' personal data in mobile apps by focusing on the centralized data structures, and further investigate whether and how they are presented.

Therefore, the Pre-analysis utilizes a bi-directional thin slicing to identify the candidate user data structures and how they are presented in UI. As shown in Figure 4, the forward thin slicing is performed to locate all candidate data structures whose values are tainted through a taint propagation path that starts from given network request application programming interfaces (APIs) and ends at parsed data structures. Thus, if the delivered user personal data exists, it should be within these parsed data structures (i.e., superset). Compared to the traditional slicing including all statements that may affect a point of interest, e.g., data structures in the context of this paper, the performed forward thin slicing focuses on producer statements (i.e., those statements that help compute and copy the value to data structures) with the same principle as in [47], thus being lightweight.

During implementation, `XPOChecker` focuses on the main data structure for managing user personal data, i.e., Java class, and the top six popular third-party network libraries as ranked by AppBrain [3] (others have less than 2% adoption in apps) are supported.

Moreover, two typical ways of data parsing, including deserialization and method annotation, are considered. For deserialization, we configure XPOChecker with the top six popular data deserialization libraries [4] (others have less than 1.7% installs). Regarding method annotation, we simply extract data structures from the annotated network request methods. A real case is as follows, where getProfile() is an annotated method that is executed to request data of the current user and the annotation placed immediately before the method name is a data structure (i.e., ProfileModel) we look for. Besides, we extract all data structures nested in the identified data structures as candidates.

```
@GET("/api/mobile/v0.5/my_user_info")
Call<ProfileModel> getProfile();
```

Meanwhile, the backward thin slicing of Pre-analysis is performed with the same principle, which starts with UI APIs and ends at reached data structures. During the backward thin slicing, the applied data transformations and relevant UI components are also extracted. The observation here is that we do not need to track how data structures are propagated in apps before being presented. Instead, we directly focus on how data structures are presented in UI and extract the applied data transformations along with relevant UI components for presentation, thus being more precise and lightweight. We found that nested data structures could cause a data structure to be missed by the backward thin slicing. But, this can be fixed through the forward thin slicing where nested data structures are all extracted.

In addition to explicit data flow, implicit data flow is considered since personal data may implicitly flow to UI, e.g., a mobile app shows user gender with a static icon of female or male according to the delivered gender value instead of directly showing its value. To cover as many UI mechanisms applied in wild as possible, we have performed an experiment over the dataset of our security assessment, which checks the view components in each app and lists 21 types of view components that occur more than 1‰ times. We crawled Android development documentation [13] and manually extracted all standard APIs related to UI which covers all these 21 types of view components. Non-standard UI interfaces such as "notifyChange()" are also consciously included. Meanwhile, it should be noted that XPOChecker is highly configurable so that customized mechanisms can be flexibly incorporated. For instance, if a new API needs to be supported, it can be appended to the configuration file of XPOChecker and no other changes are required.

In general, with the bi-directional thin slicing, the Pre-analysis outputs the candidate user data structures and the relevant data transformations applied in an analyzed app.

3.3 Locate Personal Data of Other Users

Next, given all candidates, XPOChecker needs to differentiate the personal data of other users from irrelevant one. Specifically, this step is designed as two-fold, i.e., identifying the user personal data first and then locating the personal data of other users.

3.3.1 Identify Personal Data. Since not all of the candidate data structures are related to users, the challenge must be adequately addressed here is as following.

Challenge: User personal data is mixed up with data of things or places. Based on the semantics, previous works identify user personal data with either privacy-related keywords [27–29] or machine learning [34, 35, 43]. Nevertheless, these ways are normally limited by their comprehensiveness (e.g., the considered keywords or the training set) and not able to effectively identify user personal data in the context of XPO. For instance, during user-to-user connections, the keyword "email" could belong to a public institution, which is not sensitive at all. During the empirical study, we have an insight to overcome this challenge as follows.

Insight: The semantics of the data structures' names along with their fields could be a reliable sign of personal data. A real case is that a Java class named "UserProfile" in mobile apps directly tells that it belongs to an user. Besides, we can infer that if the target data structure contains private attributes fields (e.g., "age") indicating a natural person, it is also an user data structure.

Based on this insight, the correlation-based learning-assist Privacy Discovery is applied. Firstly, a user data classifier which takes as input the name of a data structure is employed to pick out the real user data structures from candidate ones. Next, since user personal data is typically managed within centralized data structures (as clarified in the previous observation), all fields within the identified user data structures can be personal data fields and thus being kept. The greatest advantage is that this way can break through the limitations of prior works and cover as much as possible personal data of users. Then, since fields of the same type of personal data should be related or similar in semantics, the Privacy Discovery performs a correlation analysis to correlate all kept fields and further picks out the sensitive personal data fields.

During implementation, we first manually collect privacy-related keywords from previous works [27–29, 34, 35] and categorize them into 52 types as keyword lists according to their semantics. Since there is no standard source available for identifying attributes indicating a person, we take as a criterion the given attribute is related to a person only when it normally cannot have other meanings in apps. Three analysts were assigned to extract such attributes from the keyword lists hoping that two or more analysts reach a consensus. Then, we modify ClueFinder [35] which is a state of the art sensitive code fragment detector, and employ it to check whether a candidate user data structure has fields matched with the extracted attributes. If matched, the candidate user data structure is automatically labelled as user data structure. Subsequently, the user data classifier based on CNN & RNN [59] is trained to automatically identify user data structures from candidate ones. Next, to perform correlation analysis on the kept fields of the identified user data structures, the widely used and state of the art lexical databases for calculating the similarity or relatedness between a pair of concepts, i.e., WordNet::similarity [40] and ConceptNet [15, 46], are adopted. Two fields are correlated only when they are related in WordNet::similarity or ConceptNet. Thus, fields correlated with the keyword lists are treated as user personal data. The remaining uncorrelated ones are then clustered with the same way and only take negligible manual effort to further judge whether they are personal data. With all identified personal data fields, a **privacy lexicon** is constructed, which has 64 types and is used to identify personal data fields within the identified user data structures thereafter.

Table 1 shows samples of private attributes indicating a person and names of user data structures. The keyword lists and privacy lexicon are published in https://github.com/xpochecker/XPOChecker.

Table 1: Samples of attributes indicating a person and names of user data structures.

Category	Sample
Attributes indicating a person	age, gender, religion, birthday, avatar ...
Names of user data structures	user, follower, member, myprofile, account ...

3.3.2 *Locate Other Users.*

Since not all of the identified user data structures belong to other users, we have the following challenge.

Challenge: Blurred boundary between personal data of the current user and other users. XPOChecker performs a static analysis, thus there is no idea of the real values of identified user data structures. As XPO risks are associated with the personal data of other users only, user data structures that belong to the current user need to be further filtered out.

Insight: Personal data of the current user and other users can deviate from each other in three key aspects of XPO.

- **Network request**. Personal data of the current user and other users are normally requested with different network requests. For instance, the url `"/api/mobile/users/me"` is designed to request the personal data of the current user while `"/api/v1/users/nearby"` is requested to obtain the personal data of other users nearby. Specifically, urls, names of methods handling network requests and interface names of network services are taken into consideration.
- **Data structure**. The names of data structures that belong to the current user and other users can be different with each other in some cases. For instance, a user data structure named as `"Follower"` belongs to other users while `"MyProfile"` refers to the data structure of the current user explicitly.
- **UI**. Personal data of the current user and other users are normally shown in different views. For example, personal data of other users typically would not flow to activities, fragments, or widgets where the current user edits his or her profile and settings.

These three key aspects are taken as features during Privacy Discovery and their string values extracted by the Pre-analysis are collected to train an other users classifier based on CNN & RNN. The other users classifier effectively infers the owner of a given user data structure and automatically filters out user data structures belonging to the current user.

3.4 Check Data Minimization

After identifying the data structures of other users along with their personal data fields, XPOChecker needs to further investigate how they are processed before being shown in UI and judge whether any concerned inconsistency exists. The key challenge is as follows:

Challenge: Highly customized personal data processing before presentation. In customized ways, the personal data of other users is transformed before being displayed in mobile apps, which depends on various service logics and different development

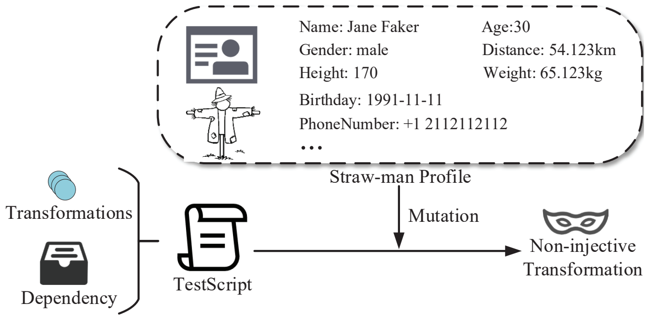


Figure 5: Design of information loss analysis.

styles of app developers. Thus, it is hard to tell whether the customized presentation forms of the delivered user data bring the concerned inconsistency and unknown how to achieve a general but effective identification of XPO risks under such circumstances. To overcome this challenge, the inconsistency-directed Risk Detection is proposed, which has the following insight.

Insight: The mathematical essence of the inconsistency between delivered personal data and presented personal data of other users is non-injective. We find that no matter how customized is the user personal data processed, only information loss can result in the concerned inconsistency and the essence of information loss is non-injective property, i.e., whether two close but different delivered user personal data can result in the same presentation. In line with the criterion of XPO, this property is reflected in two aspects. On the one hand, if personal data of other users is not shown in UI (i.e., resulting the same presentation - null), it is obviously a concerned inconsistency. On the other hand, if personal data of other users is displayed in UI but being processed with non-injective transformations (aka, data masking transformations), the inconsistency is also brought in.

The Risk Detection inspects the first aspect by checking whether the personal data fields within data structures of other users are reached by the backward thin slicing (i.e., flow to UI). For the second aspect, the Risk Detection employs the information loss analysis based on a forged straw-man profile to automatically check the "non-injective" property of applied transformations.

```java
public void bind(VideoItem vi, int i){
    ...
    bindDistance(Float.valueOf(vi.metadata.distanceInKm));
    ...
    return;
}
//Bind transformed value of "distanceInKm"
private void bindDistance(Float f){
    ...
    int max=Math.max(1, Math.round(f.floatValue()));
    SnsDIstanceLabelView sdlv=this.mDistanceView;
    sdlv.setText(sdlv.getContext().getString(R.string.
          distance_km, new Object[]{Integer.valueOf(max)}));
}
```

Figure 6: A real case of non-injective transformations which round fine-grained "distanceInKm" to be a coarse-grained one and turn it to be "1 km" if it is less than 1 km.

Specifically, we constructed the forged straw-man profile with fake but general values for 40 types of personal data in the privacy lexicon. The left 24 types are ambiguous to assign concrete values, e.g., "user preference", thus not being considered. Most importantly, the fake value of each considered personal data type is mutated to be various values that are designed to be close to each other, which have the best chance to identify non-injective transformations. Then, as shown in Figure 5, the information loss analysis adopts an automated way to generate test scripts that invoke target transformation functions and take mutated personal data values as inputs. Finally, the outputs of executing test scripts efficiently and precisely tell whether given transformations are non-injective.

To ease the understanding of information loss analysis, a real case is presented in Figure 6. The personal data field "`distanceInKm`" of user data structure "`VideoItem.metadata`" is processed with transformation functions, e.g., "`Math.round()`" and "`Math.max()`". Then, a test script will be generated, which invokes these identified transformation functions and imports necessary dependencies to ensure that it can successfully run, e.g., importing the library "`java.lang.Math`" where two of the identified transformation functions are defined. Finally, according to the personal data type of "`distanceInKm`" (i.e., distance), the corresponding mutated values of distance (e.g., "54.122", "54.123" and "54.124") are taken as inputs. Then the generated test script is executed to check whether the outputs of the tested transformation function are the same or not. If the outputs are the same, the tested transformation is non-injective. Otherwise, it is not. Thus, "`Math.round()`" and "`Math.max()`" would be flagged as non-injective transformations.

Finally, based on the not presented personal data of other users and the identified non-injective transformations, the Risk Detection generates XPO results. To prevent `XPOChecker` from being abused, it will be gradually published in https://github.com/xpochecker/XPOChecker once this paper is accepted.

3.5 Performance Validation

It is critical to understand the performance of `XPOChecker` for ensuring the reliability of our security assessment. First, we evaluated each part of `XPOChecker`. Then, due to the missing of ground truth, we manually verify the reported XPO results. Studying the full set of results is infeasible. Hence, we adopt the sampling scheme.

Dataset of Training. From the dataset of our security assessment, we randomly selected 300 apps from 30 categories of Android apps (10 apps from each category) that contain user data structures. Specifically, four Android experts manually labeled and cross-validated 8,958 elements from network requests, data structures, and UI in these sampled apps, which took around 50 hours for each and span over 10 days. For data structures that have fields indicating a person, their names are automatically labelled as user data structures. In total, we collected 1,549 positive samples and 7,409 negative ones for classifying user data structures, and 1,039 positive samples together with 7,919 negative ones for filtering out data structures of the current user. To achieve better classifying results, we balance the labeled samples by keeping all positive ones and randomly sampling the same amount of negative ones as our training set. As a result, the total sizes of our training sets are 3,098 and 2,078 respectively.

Performance of Privacy Discovery. A ten-fold cross-validation is adopted where we randomly partition each dataset into ten subsets, train the relevant classifier on nine of them, and then test the remaining subset. The process is repeated on each subset ten times. Finally, we adopted CNN & RNN as the machine learning algorithm, which has the best performance (88.96% precision and 88.39% recall for the user data classifier, 95.70% precision and 89.90% recall for the other users classifier).

Performance of Risk Detection. For the first aspect of non-injective property, performance of the Risk Detection can be verified by the validation results of `XPOChecker`. For the second aspect, i.e., information loss analysis, the average analysis time of it for each analyzed app in our security assessment is 2.61s, which reflects its high efficiency. In total, 82 apps were reported to be affected by non-injective transformations. To verify the performance of information loss analysis, we randomly selected another 82 apps that were not reported by it and manually checked all of them (i.e., 164 apps). As shown in Table 3, the information loss analysis achieves high precision (82.93%) and high recall(100%). During the validation, we found that the non-injective data transformations can still be effectively identified even when they are obfuscated. The main reason for the 14 false positives is that two or more of the mutated values for certain personal data happen to trigger the exception of target transformation function, which outputs the same results and thus is wrongly flagged as being non-injective.

To validate the performance of `XPOChecker`, we randomly selected another batch of 300 apps (10 apps from each category) analyzed during our security assessment, which is a best-effort solution. Since this batch was randomly picked, it is firmly believed the validation results can well provide a reasonable performance estimation of `XPOChecker` during the security assessment.

Validation of `XPOChecker`. By automatically monitoring network traffic with MitmProxy [1] or instrumenting the identified data structures of other users and UI APIs in these apps with Frida [37] and Xposed [44], we check whether the delivered personal data is consistent with the presented one of other users. Each sampled app was manually explored for at least 20 minutes on average (to trigger as many functionalities as possible). As shown in Table 2, `XPOChecker` achieves 75% precision and 96.77% recall among 170 verifiable apps. The main reasons for not being able to verify all the sampled apps are due to the need of special credentials, payment, certain network or area restrictions, and so on. Besides, Mitmproxy did not work for all the verified apps. It failed to capture the traffic of 9 apps. We mitigated this problem by conducting instrumentation with Frida and Xposed. Among all verifiable apps, there is only one false-negative case whose code is obfuscated and thus thwarting Privacy Discovery which relies on code semantics. For the left verifiable apps, 30 are true positives, 10 are false positives and 129 are true negatives. In most cases, the false positives are due to the failure of distinguishing other users from the current user. We further manually checked the code of these false-positive apps and confirmed that most of the results of Privacy Discovery are right. But these false-positive apps actually have no other users or employ coding practices (used by most apps for requesting or processing other users' data) to only handle the information belonging to the current user. Since `XPOChecker` achieves practical precision and high recall, we consider them acceptable.

Table 2: Performance of XPOChecker among 30 app categories. "-" in Precision means there is no reported positive for given app category (i.e., TP+FP = 0); "-" in Recall means there is no positive in ground truth for given app category (i.e., TP+FN = 0).

ID	App Category	#Verifiable	Precision (%)	Recall (%)	ID	App Category	#Verifiable	Precision (%)	Recall (%)
01	Education	5	100.00	100.00	16	Food & Drink	6	66.67	100.00
02	Entertainment	3	-	-	17	Health & Fitness	7	-	-
03	House & Home	6	66.67	100.00	18	Art & Design	10	-	-
04	Lifestyle	5	50.00	100.00	19	Books & Reference	10	-	-
05	Maps & Navigation	1	0.00	-	20	Comics	6	-	-
06	Music & Audio	5	75.00	100.00	21	Communication	7	-	-
07	Parenting	5	100.00	100.00	22	Medical	6	-	-
08	Shopping	3	66.67	100.00	23	News & Magazines	6	100.00	100.00
09	Auto & Vehicles	2	-	-	24	Personalization	9	-	-
10	Beauty	2	100.00	100.00	25	Photography	10	-	-
11	Business	3	0.00	-	26	Productivity	10	-	-
12	Dating	3	100.00	100.00	27	Sports	8	100.00	100.00
13	Social	6	100.00	100.00	28	Video Players & Editors	10	-	-
14	Travel & Local	4	-	-	29	Weather	8	0.00	-
15	Finance	0	-	-	30	Events	7	75.00	100.00
Overall		170	75.00	96.77					

Table 3: Performance of information loss analysis.

	TP	FP	TN	FN	Precision	Recall
Reported Apps	68	14	-	-	82.93%	100.00%
Not-reported Apps	-	-	82	0		

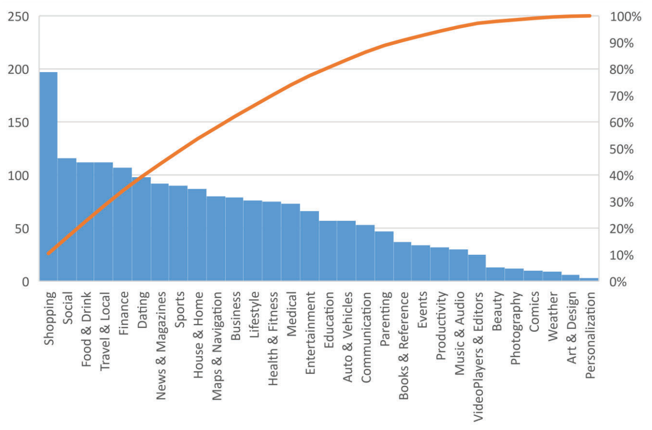


Figure 7: Distribution of app categories affected by XPO.

4 SECURITY ASSESSMENT

Our security assessment is performed on a large dataset of apps collected from Google Play during April 2021. These apps were selected with the top 500 apps in each category listed by AndroidRank [14] (30 categories were considered in total[3]). Since category "Events" only had the top 340 apps in AndroidRank, we crawled 14,840 app ids in total and successfully downloaded 13,906 apps while the remaining 934 apps were failed to download due to not being free, area restriction and so on.

Analysis Statistics. To perform the security assessment of XPO, XPOChecker analyzed these downloaded apps on a Ubuntu 18.04 LTS 64-bit server with 40 CPU cores (1.2GHz) and 128GB memory. The analysis is performed in parallel and has a timeout of 30 minutes to analyze each app. XPOChecker is implemented based on the Soot framework [30] and FlowDroid [5] with 4,270 lines of Java code and 3,180 lines of Python code. On average, the analysis took 52.65s to analyze each app and 13,820 (99.38%) apps were successfully analyzed in total. The rest apps either ran out of time or failed to be analyzed by Soot or FlowDroid. Finally, XPOChecker found that 1,902 apps were affected by XPO risks.

Research Questions. The following two research questions are vital in revealing the real situation of XPO in the wild, but have not yet been answered in any prior work:

- *RQ1: What is the landscape of XPO risks in the wild?*
- *RQ2: How severe are XPO risks?*

[3]Categories "Daydream", "Libraries & Demo", "Tools" and "Wear OS by Google" in Google Play [41] are filtered out since their apps are overlapped with other categories or irrelevant with XPO risk.

4.1 Ethics

Conducting the security assessment might raise ethical issues. Therefore, we carefully manage our research activities to ensure that they stay within the legal and ethical boundaries. This whole research has been approved by our institution's IRB. The approval process is similar to the exempt review in the US because this study is considered as "minimal risk" when consulting with IRB staff. Note that for performing necessary investigation, we only use our own accounts to verify XPO risks in most cases and immediately stop if real user data is encountered. We emphasize that real user data is not stored in any persistent storage and is also not disseminated in any way to anybody. Additionally, user data involved is all anonymized or masked to prevent their owners (ourselves in most cases) from being identifiable, and we diligently follow all requirements proposed by app developers and app service providers to protect real users from being affected.

4.2 Landscape of XPO

Prevalence of XPO. Out of the 13,820 successfully analyzed apps, we identified 1,902 (13.76%) apps having XPO risks in total. Especially, as shown in Figure 7, the affected number of apps in all app categories varies. We find that the distribution (of the apps having XPO risks) among the app categories is positively correlated with the amount of services involving user-to-user channels. For example, compared to other categories, more apps of Shopping and Social categories are affected while they naturally provide more services involving socializing with other users. Besides, this distribution conforms to the "Pareto principle" and can provide key areas for regulators to review and supervise.

Table 4: The top-10 of affected personal data types.

Personal Data Type	Num. of Affected Fields
Mail Address	1,890
Geographical Location	1,743
Phone Number	1,440
Birthday	1,429
Age	1,084
Personal Address	816
Password	363
Social Identifier	355
Account Balance	328
Job	266

Affected personal data. As shown in Table 4, we find that several types of sensitive personal data (e.g., mail address) are the mainly affected ones (note that the multiple affected personal data fields within one app may belong to the same type). During the validation, we noticed that the most affected personal data types are also the most frequently collected ones. This finding is not out of expectation but extremely helpful. On the one hand, it can help determine which types of user personal data should be reviewed as the highest priority for app developers. On the other hand, it can help mobile users keep focused and better protect their privacy with their own efforts. For example, on the premise that mobile service can be normally served, mobile users can register with disposable email addresses or phone numbers, fake geographical locations and forged birthdays to ensure the security of their privacy in most cases.

Moreover, by looking into the reported results, we found two special kinds of personal data that are rarely studied by prior works, i.e., activity privacy and passive privacy. Activity privacy includes sensitive personal data that users generate during interacting with mobile apps. For example, when viewing various goods in a shopping app, users can add anything they like into their wishlist/shopping cart, which is sensitive and can be used to learn their preferences. Passive privacy points to the sensitive personal data that is not proactively generated by the user. For instance, we found a popular communication app which leaked the cell location of inmates in the United States. Such passive privacy is not provided or generated by its owners but passively assigned by other parties (e.g., the prison). Both of these two special kinds of personal data have no specific system APIs to track [6, 42], no user input to monitor [27, 34] and are irrelevant with illegal personal data collection. Thus, they cannot be well handled or even found by most prior works.

Scale of affected users. The installations of identified apps in Google Play (if available) were collected and the results are out of expectation as shown in Table 5[4]. If taking ">5,000,000" as the boundary, we found that compared to long-tail apps, high-profile apps do not care more about the security of user-to-user connection channels. Regarding the distribution among the installation levels, we found that the installation level "1,000,000-5,000,000" had the most affected apps. Moreover, we estimated the scale of affected mobile users by counting the sum of all affected apps' installation, which reaches 16.89 billion in total. Noted that this number does not represent the real scale of affected users since a mobile user can use multiple apps simultaneously. But it actually can reflect how big the affected user scale is, and we would like to raise the awareness of the community by doing so.

Existence of XPO Risks on iOS Platform. For the validated vulnerable apps during performance validation, we try to look for their counterparts on iOS platform and successfully find 44 apps have iOS versions. Then by monitoring network traffic, we confirm that 19 apps still have the same privacy risks as their Android counterparts, which reveal the existence of XPO risks in iOS apps for the first time.

Answer I: *XPO is unexpectedly widespread in the wild and numerous mobile users are at risk.*

4.3 Severity of XPO Risks

Generally, XPO directly leads to the exposure of sensitive personal data, which already causes various privacy infringement. To further understand the severity of XPO, we randomly picked 100 apps from all detected ones to study. We find that the severity of XPO can be more than merely exposing certain user data and take exposure of location-related user data as illustrations in the next.

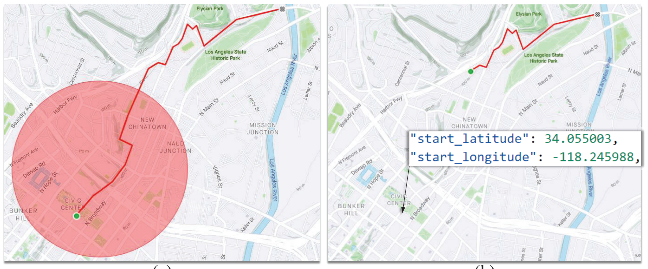
4.3.1 Invalidate Mechanisms of Privacy Preservation. An obvious case proving that XPO can invalidate mechanisms of preserving user privacy is *Strava*, a popular fitness app. `XPOChecker` detected that *Strava* exposed the start GPS point of users' running trajectory. To further study it, we created two *Strava* users, i.e., A and B, and set them as friends of each other. Then, we set the privacy zone[5] of A, started running which created a running trajectory as shown in Figure 8, and posted this running record. Nevertheless, as shown in Figure 8(b), while the segments of A's running trajectory within the privacy zone were correctly hidden, the real GPS location of the start point was delivered to B. Since user A intends to hide the start point, which is inside the privacy zone, such XPO may result in a leak of sensitive personal data, e.g., the home address and workplace where users may commonly take as start points to work out. By combining public information in *Strava*, e.g., photos, the adversary may even be able to locate a specific person in the real world, which further raises safety threats to users of *Strava*.

4.3.2 Leak Business Secret. XPO risks can even leak business secrets. A real case is that we found app *B* directly exposed users' geographical location. And the adversary can directly exploit this at scale by incrementally crawling as many as possible users' geographical location to leak business secrets, e.g., the geographical distribution of users in app *B*, as shown in Figure 9. There are many

[4]1,848 detected apps were still in Google Play on August 2021.
[5]"Privacy Zone" [16] is a privacy-preservation mechanism, which hides the portion of any activity within it from being seen by other users.

Table 5: Distribution of affected apps' installation.

Installation Level	Num. of Affected Apps
>50,000,000	105
10,000,000 - 50,000,000	413
5,000,000 - 10,000,000	316
1,000,000 - 5,000,000	691
500,000 - 1,000,000	155
100,000 - 500,000	136
<100,000	32



```
"start_latitude": 34.055003,
"start_longitude": -118.245988,
```

(a) (b)

Figure 8: (a) User A: Running trajectory and privacy zone; (b) User B: View and network traffic of A's running record.

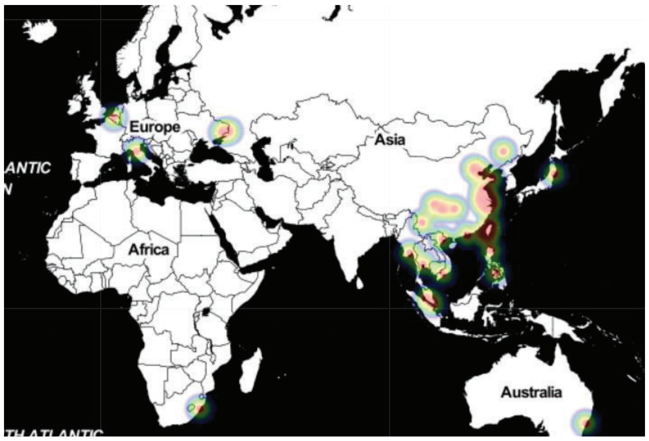


Figure 9: Revealing the geographical distribution of users.

similar situations where business secrets can be leaked due to XPO risks. For instance, we found that some apps leaked the registration and last active date of users, which the adversary can exploit to understand the daily registered users, daily active users (DAU) and so on of a mobile app. Such business secrets can be key factors for seeking funding and can even be analyzed by competitors to build targeted marketing strategies.

4.3.3 Enable Link Attack and Pose Safety Threats. Among the affected personal data, many types are linkable (e.g., phone number), which have been studied by LinkDroid [18]. LinkDroid focused on the risk that **malicious apps** can link and aggregate usage behaviors of the same user across different apps. In contrast, we found that such linking/aggregation attack is possible for the malicious current user to conduct. Moreover, the leaked linkable personal data can even be used to aggregate users with the same sensitive attributes, which can further leak the sensitive membership of users.

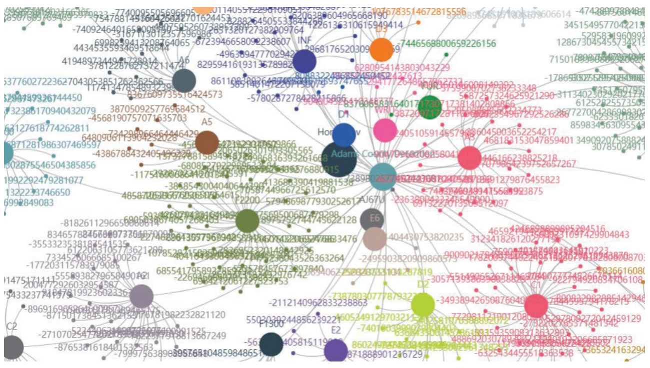


Figure 10: Leakage of inmate membership in app *C*

App *C* is a communication app designed for allowing inmates in the United States to be visited by their families and friends through video calls. As identified by `XPOChecker`, app *C* did deliver more than necessary personal data of inmates to the attacker. `"inmate_location"` is the linkable personal data involved, which represents the cell location of an inmate. As presented in Figure 10, such excessive exposure can benefit the adversary to restore the cellmate membership among more than 30,000 inmates held in 130 prisons or detentions in the US. Namely, the adversary can infer whether the target inmate is a member of a specific cell in a particular prison, who are the cellmates of the target inmate, even how many cells are in the given prison and the lower bound of its cells' sizes. Combined with the public personal information of inmates shown in app *C*, such XPO risk can seriously threaten the personal safety of inmates in addition to just privacy.

Answer II: *XPO can leak various sensitive personal data, make the privacy preservation mechanisms ineffective, leak business secrets and even pose real safety threats to users.*

5 LESSONS LEARNED FROM XPO

We responsibly notified the affected developers, which has two main goals: first, we need to inform them of the identified XPO risks. Second, we want to gain insights into the underlying lessons that can be learnt in the first place.

5.1 Notification Campaign

To notify the affected app developers, we extracted the email addresses they submitted to Google Play. To ease the overhead of handling our reports, we briefly explained XPO risks and put details in an attached report. Besides, three questions were asked, including if they are aware of preserving user privacy while delivering personal data of other users, if they are conscious of XPO risks as well as why they exist, and what are their plans to remedy XPO or any proposal for support. Furthermore, we followed best practices established by prior works [31, 48, 49] allowing developers to opt-out.

We performed two notification campaigns to inform the affected app developers. During this process, we merged the emails sending to the same developer and finished the first notification campaign before August 15, 2021. Next, for developers that did not response to us or only provide automated responses, we launched the second notification campaign which was finished before December 1, 2021. Overall, 1,748 unique report emails were sent and 1,641 of them were successfully sent. The fail ones are due to invalid email addresses, inboxes of the recipients are full and so on. In total, 386 unique responses were received, where 355 of them are automated replies. Among the left 31 manual responses, 1 denied our report, 12 confirmed and 7 confirmed our report along with feedback towards our questions. Most of the manual responses that neither confirm nor deny our report said that *"We'll forward this report to the appropriate team"* and no further responses are received any more. Note that there may be multiple back and forth processes during the notification campaigns since app developers may query about more details of XPO and ask for supports.

5.2 Developer Response

For the app developer that denied our report, we confirmed that it is the false positive of `XPOChecker`. Referring to the confirmation responses but without feedback towards our questions, we have observed that many of them just agree with or acknowledge our reports and briefly explain how they are going to fix the reported privacy issues. We notice that the possible reason for not answering our questions can be the consideration of protecting certain business secrets, which may be needed to provide reasonable feedback. For instance, some app developers thank us and *"acknowledge about the report email"* but say that they cannot *"disclose any information with regards to our queries"*.

Most importantly, by looking into those confirmations with feedback and relevant apps, multiple lessons are learnt as follows:

- App developers subconsciously thought that their users can only obtain information from what is shown in UI. This is hardly new since app developers nowadays still hard code revertible credentials in mobile apps [23, 60, 62]. However, as shown in the threat model of XPO, the scope of the mobile client can be fully under the adversary's control.

  Lesson I: *App developers should correct their wrong understanding of the privacy security boundary.*

- App developers stated that *"they are always doing their best to take user privacy seriously and preserve user privacy while transmitting or using user personal data"*. However, three of them admit that XPO exists because their current development status is concentrated on the implementation of functionality to occupy the market while carelessly resulting in such privacy risk.

  Lesson II: *The regulators should notice that developers can be inconsistent between their sayings and actions, and app developers should recheck their service.*

Then, we found that some issues commonly seen in software development also contribute to the existence of XPO and can be referred to avoid it.

- It is mentioned that during the fast iterative development of mobile apps, there might appear inconsistent demands of the user data among different app versions. One of these respondents directly checked with his team once he confirmed our report and said that *"there were three features that required longitude and latitude from other users but two out of three were removed in the current version, the last one can be fixed by returning a different type of data 'boolean'"*. As a result, during the frequent iteration of mobile apps, app developers may forget to stop delivering personal data that is no longer needed in the current version of their apps.

  Lesson III: *When iterating app versions, app developers should check whether user data used in the previous app version is still necessary for the current version.*

- During product management, app developers may indeed considered what user data should be delivered for building user-to-user connections. But, as argued by one typical feedback, *"during development, it is a common issue where developers' work and product management cannot always be in sync"*, and thus bringing about the XPO risks.

  Lesson IV: *App developers should ensure the synchronization between app development and product management to avoid XPO risks.*

With further investigation, we found that app developers were even forced to intentionally allow such privacy risks to exist in some cases. Such special situations are as follows:

- **Limited budget to perform privacy-preservation computation.** There is indeed a balance between protecting user privacy and providing satisfying service, as some respondents stated: *"The product was built as a startup with a small team and limited budget so the overhead processing time on the server is not possible, that is why we had to push some logic to clients like a distributed way to handle the logic"*.
- **Consideration of service performance.** Another factor worried by the app developers is the performance of their service. If all privacy-related computations are moved to the server-side, the performance of their service would be affected, e.g., bringing high response delay. Better service performance means better user experience, which is positively correlated to the market share the app can occupy. Thus, app developers have to consider the performance cost they have to pay for preserving user privacy.

Lesson V: *App developers should ensure the security of user privacy, instead of intentionally allowing the existence of XPO.*

During the notification campaigns, most of the app developers that confirmed our report have adjusted their apps or services to mitigate XPO risks with our help. More importantly, we want to point out that app developers should and must reconsider the balance between providing their service and preserving user privacy, and put the security of user privacy in the first place.

6 DISCUSSION & LIMITATION

Our study brings to light an insufficiently-studied but critical privacy risk (i.e., XPO) existed within the data delivery across users. Our security assessment reveals the landscape and severity of XPO, which suggest that the community and app developers lack a sufficient awareness of it, and numerous mobile users are at risk. By notifying the affected app developers, our work further presents five underlying lessons which can be referred to mitigate XPO risk.

We admit that our approach naturally suffers from certain limitations. For instance, code obfuscation can bring false negatives since XPOChecker relies on code semantics to identify the personal data of other users. However, as shown by Wermke et al. [55] (less than 25% of apps were obfuscated in Google Play) and Nan et al. [35], app developers tend not to obfuscate data-related code within their apps to avoid disrupting the apps' normal executions (e.g., causing a crash). Besides, during our validation, there is only one false negative that is due to obfuscation. Admittedly, XPOChecker cannot prevent determined app developers from evading our analysis such as adopting code obfuscation to erase the semantics of app code. To handle this, XPOChecker can adopt more sophisticated techniques [7, 56] which is an orthogonal research direction. In this paper, XPOChecker is designed to conduct security assessment of XPO in the wild and thus we do not consider a future app developer who tries to evade XPOChecker. Besides, our tool validation and security assessment show that XPOChecker is effective to reveal the landscape and severity of XPO risks in the wild.

Additionally, it should be pointed out that XPOChecker may have certain amount of false positives. Although dynamic Android app testing may still be the most convincing way to confirm XPO risks, we cannot apply it to automatically verify the results of XPOChecker, since it is limited by low code coverage [26, 38, 52]. Namely, if a truly risky app reported by XPOChecker is dynamically tested by an automated exerciser, e.g., Monkey [24], the user interactions to manifest the risk can be too specific for the exerciser to trigger. Further, a semi-automated dynamic analysis is also considered to verify the results of XPOChecker. Nevertheless, we find that it is obviously unsuitable for a large scale of apps, i.e., manually examine, register, login and explore all 1,902 apps reported by XPOChecker to verify the XPO risks. Hence, we take the sampling scheme instead to perform the performance validation of XPOChecker, which adopts a similar semi-automated way to verify the randomly selected results of XPOChecker.

Besides, two procedures, i.e., extracting UI APIs from the official documents of Android development and re-training Privacy Discovery of XPOChecker, may need to be repeated regularly. On the one hand, when Android updates and adds new UI APIs, we need to re-extract these APIs. However, by analyzing their update dates, we find that UI APIs are updated every 444.33 days on average. The time window for updating XPOChecker is even broader, since it often takes time to adopt new APIs in practice. On the other hand, we collect the previous versions (one year ago) of the apps that were used for training in this paper. Based on these collected apps, we then re-train the two classifiers of Privacy Discovery, and check their performance changes on the training sets labelled in this paper. The results show their performance is slightly affected and degraded (i.e., model aging issue), which can be mitigated by employing relevant SOTA but orthogonal approaches.

Finally, although our security assessment is performed on Android apps, the ideas proposed in this paper also work on iOS platform. For the first time, the existence of XPO risks in iOS apps is confirmed. Besides, based on XPOChecker, further researches can study whether the personal data delivery across users complies with app descriptions, privacy policies and many other possible sources, which we leave as a future work. Moreover, during the process of notifying the affected app vendors, we strongly feel the positive attitude of app developers (e.g., Opera) who have provided feedback for working together to mitigate XPO risks and it is firmly believed that mobile privacy will be substantially enhanced with the joint efforts of the whole community.

7 RELATED WORK

7.1 Personal Data Collection

Static & Dynamic Analysis. To detect privacy leakage associated with personal data collection, researchers have designed various schemes with both static [5, 8, 25, 32, 36, 54] and dynamic taint analysis [17, 33, 50, 51]. Moreover, Recon [43] and AGRIGENTO [12] detected privacy leakage based on network traffic which is generated by testing mobile apps. Different from them, this paper focuses on cross-user personal data over-delivery risk instead of privacy risks within the personal data collection.
Privacy Types & Legality of Collection. Some prior works [27, 28, 34, 35] considered leakage of fixed types of private data during personal data collection, while other works focused on the detection of real private data leakage, which is not in compliance with user intention [10, 19, 20, 39, 57, 58] or privacy policy [2, 45, 53, 61]. Unlike these works, this paper identifies as much as possible the personal data of other users with the Privacy Discovery and focuses on privacy leakage within the user-to-user connection channels.

7.2 Cross-User Personal Data Delivery

Few work has paid attention to personal data delivery across users. The most related work was conducted by Kock et al. [29], which revealed Server-based InFormation OvershariNg vulnerabilities (denoted as SIFON in the paper). By using both static analysis and dynamic analysis, Kock et al. proposed S-HUSH/D-HUSH to semi-automatically detect this vulnerability. S-HUSH/D-HUSH focuses on sensitive information that was sent from the server but hidden. With a large-scale analysis of 31,559 apps in social category, eight of these apps were identified to be vulnerable, which confirmed the existence of SIFON vulnerability in Android social apps.

Nevertheless, different from S-HUSH/D-HUSH, this paper focuses on cross-user personal data over-delivery and overcomes three new challenges to identify XPO risks in real-world apps.

Firstly, S-HUSH/D-HUSH focuses on sensitive information that is shared from the server side and identifies it based on manually collected keywords. The keywords-based scheme is naturally limited by its comprehensiveness. Most importantly, as pointed out by the key challenges in this paper, not all data shared by the server will bring privacy risks even when it is matched with specific keywords (e.g., an email address belonging to a public institution or the current user). From its evaluation results, S-HUSH/D-HUSH is also corroborated to have both considerable false negatives and high false positives, where only 126 out of 31,559 social apps were reported by S-HUSH and merely eight out of 126 social apps were confirmed by D-HUSH to be truly vulnerable. Thus, their findings can greatly mislead the community's understanding and not able to reveal the real situation in the wild. Contrary to S-HUSH/D-HUSH, XPOChecker focuses on the over-delivered personal data of other users during user-to-user connections. Based on the proposed insights, XPOChecker employs the Privacy Discovery to effectively identify the personal data of other users in mobile apps.

Secondly, S-HUSH/D-HUSH only cares about hidden sensitive information. However, it is demonstrated that the presented sensitive information can also bring about privacy risks when it is transformed with non-injective data transformations. Compared with S-HUSH/D-HUSH, this paper focuses on the inconsistency between the delivered personal data and presented personal data of other users and proposes the Risk Detection which effectively identifies the concerned inconsistency based on its mathematical essence, i.e., "non-injective" property. Therefore, S-HUSH/D-HUSH is naturally limited in this aspect.

Finally, D-HUSH was adopted to verify whether the social apps reported by S-HUSH were indeed vulnerable or not. However, as discussed before, both semi-automated and automated dynamic analysis are not suitable for verifying the results of XPOChecker. Consequently, we adopt the sampling scheme to conduct the performance validation of XPOChecker, which is believed to be able to reflect the real performance of XPOChecker among the whole dataset (i.e., 13,820 apps of 30 categories analyzed in our security assessment). Compared to the performance of S-HUSH, XPOChecker achieves 7x precision improvement (75.00% precision). Although the precision of semi-automated D-HUSH can be regarded as 100%, only eight social apps were confirmed to be vulnerable by it, which misses a significant number of true positives in the wild and lead to a low recall when compared to XPOChecker (96.77%).

Generally, without overcoming the key challenges solved by XPOChecker, S-HUSH/D-HUSH is greatly limited to perform the large-scale security assessment as conducted by XPOChecker. Furthermore, S-HUSH/D-HUSH cannot unveil the landscape and severity of XPO risks due to its limitations, not to mention drawing the five underlying lessons as learnt in this paper.

8 CONCLUSION

In this paper, we bring to light an insufficiently studied but critical privacy risk (i.e., XPO) in user-to-user connections. To better understand this issue in the wild, we have systematically analyzed it and designed XPOChecker, which overcomes several non-trivial challenges to automatically identify it. With a security assessment of 13,820 real-world Android apps, we revealed the severity of XPO in addition to its landscape in the wild, which puts numerous mobile users at serious privacy risk and even pose safety threats. The existence of XPO risks in iOS apps is confirmed for the first time, which shows that XPO risks can be more prevalent than thought. Moreover, by launching the notification campaigns, we revealed five underlying lessons learnt from XPO. It is believed that our work can make up for the deficiency of the community's understandings and awareness of XPO risks and inspire future researches.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. This work was supported in part by the National Key Research and Development Program (2021YFB3101200), National Natural Science Foundation of China (U1836210, U1836213, 62172104, 62172105, 61972099, 61902374, 62102093, 62102091), Natural Science Foundation of Shanghai (19ZR1404800), China Postdoctoral Science Foundation (BX2021079, 2021M690706). Min Yang is the corresponding author, and a faculty of Shanghai Institute of Intelligent Electronics Systems, and Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China.

REFERENCES

[1] Cortesi Aldo, Hils Maximilian, and Raumfresser. 2021. Mitmproxy - an interactive HTTPS proxy. https://mitmproxy.org/.
[2] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. 2020. Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with PoliCheck. In *Usenix Security Symposium (USENIX Security)*.
[3] AppBrain. 2021. Android network libraries. https://www.appbrain.com/stats/libraries/tag/network/android-network-libraries?sort=apps.
[4] AppBrain. 2021. Data serialization libraries. https://www.appbrain.com/stats/libraries/tag/data-serialization/data-serialization-libraries.
[5] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* (2014).
[6] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. Pscout: analyzing the android permission specification. In *ACM Conference on Computer and Communications Security (CCS)*.
[7] Benjamin Bichsel, Veselin Raychev, Petar Tsankov, and Martin Vechev. 2016. Statistical deobfuscation of android applications. In *ACM Conference on Computer and Communications Security (CCS)*.
[8] Yinzhi Cao, Yanick Fratantonio, Antonio Bianchi, Manuel Egele, Christopher Kruegel, Giovanni Vigna, and Yan Chen. 2015. EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework.. In *ISOC Network and Distributed System Security Symposium (NDSS)*.
[9] CCPA. 2021. CCPA. REQUESTS TO DELETE PERSONAL INFORMATION. https://oag.ca.gov/privacy/ccpa#sectione.
[10] Xin Chen and Sencun Zhu. 2015. DroidJust: Automated functionality-aware privacy leakage analysis for Android applications. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*.
[11] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated test input generation for android: Are we there yet?(e). In *International Conference on Automated Software Engineering (ASE)*.
[12] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, and Giovanni Vigna. 2017. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis.. In *ISOC Network and Distributed System Security Symposium (NDSS)*.
[13] Android Developer. 2021. Android API reference. https://developer.android.com/reference.
[14] AndroidRank Developer. 2021. List of Android Most Popular Google Play Apps. https://www.androidrank.org/..
[15] ConceptNet Developer. 2021. ConceptNet API. https://github.com/commonsense/conceptnet5/wiki/API.
[16] Strava Developer. 2021. Privacy Zones. Strava Support. https://support.strava.com/hc/en-us/articles/115000173384-Privacy-Zones.
[17] William ENCK. 2010. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.
[18] Huan Feng, Kassem Fawaz, and Kang G Shin. 2015. LinkDroid: Reducing unregulated aggregation of app usage behaviors. In *Usenix Security Symposium (USENIX Security)*.
[19] Hao Fu, Zizhan Zheng, Somdutta Bose, Matt Bishop, and Prasant Mohapatra. 2017. Leaksemantic: Identifying abnormal sensitive network transmissions in mobile applications. In *IEEE International Conference on Computer Communications (INFOCOM)*.
[20] Hao Fu, Zizhan Zheng, Aveek K Das, Parth H Pathak, Pengfei Hu, and Prasant Mohapatra. 2016. Flowintent: Detecting privacy leakage from user intention to network traffic mapping. In *International Conference on Sensing, Communication and Networking (SECON)*.
[21] GDPR. 2021. GDPR. Art. 17 Right to erasure ('right to be forgotten'). https://gdpr.eu/article-17-right-to-be-forgotten/.
[22] GDPR. 2021. GDPR. Art. 5 Principles relating to processing of personal data. https://gdpr.eu/article-5-how-to-process-personal-data/.
[23] Leonid Glanz, Patrick Müller, Lars Baumgärtner, Michael Reif, Sven Amann, Pauline Anthonysamy, and Mira Mezini. 2020. Hidden in plain sight: Obfuscated strings threatening your privacy. In *ACM Conference on Computer and Communications Security (CCS)*.

[24] Google. 2021. Google. Android Monkey. http://developer.android.com/tools/help/monkey.html.

[25] Michael I Gordon, Deokhwan Kim, Jeff H Perkins, Limei Gilham, Nguyen Nguyen, and Martin C Rinard. 2015. Information flow analysis of android applications in droidsafe.. In *ISOC Network and Distributed System Security Symposium (NDSS).*

[26] Yuyu He, Lei Zhang, Zhemin Yang, Yinzhi Cao, Keke Lian, Shuai Li, Wei Yang, Zhibo Zhang, Min Yang, Yuan Zhang, et al. 2020. TextExerciser: Feedback-driven text input exercising for Android applications. In *IEEE Symposium on Security and Privacy (S&P).*

[27] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. 2015. {SUPOR}: Precise and scalable sensitive user input detection for android apps. In *Usenix Security Symposium (USENIX Security).*

[28] Jianjun Huang, Xiangyu Zhang, and Lin Tan. 2016. Detecting sensitive data disclosure via bi-directional text correlation analysis. In *International Symposium on Foundations of Software Engineering (FSE).*

[29] William Koch, Abdelberi Chaabane, Manuel Egele, William Robertson, and Engin Kirda. 2017. Semi-automated discovery of server-based information oversharing vulnerabilities in android applications. In *International Symposium on Software Testing and Analysis (ISSTA).*

[30] Patrick Lam, Eric Bodden, Ondrej Lhoták, and Laurie Hendren. 2011. The Soot framework for Java program analysis: a retrospective. In *Cetus Users and Compiler Infastructure Workshop (Cetus).*

[31] Frank Li, Zakir Durumeric, Jakub Czyz, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson. 2016. You've got vulnerability: Exploring effective vulnerability notifications. In *Usenix Security Symposium (USENIX Security).*

[32] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. 2015. Iccta: Detecting inter-component privacy leaks in android apps. In *International Conference on Software Engineering (ICSE).*

[33] Björn Mathis, Vitalii Avdiienko, Ezekiel O Soremekun, Marcel Böhme, and Andreas Zeller. 2017. Detecting information flow by mutating input data. In *32nd International Conference on Automated Software Engineering (ASE).*

[34] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. 2015. Uipicker: User-input privacy identification in mobile applications. In *Usenix Security Symposium (USENIX Security).*

[35] Yuhong Nan, Zhemin Yang, Xiaofeng Wang, Yuan Zhang, Donglai Zhu, and Min Yang. 2018. Finding Clues for Your Secrets: Semantics-Driven, Learning-Based Privacy Discovery in Mobile Apps.. In *ISOC Network and Distributed System Security Symposium (NDSS).*

[36] Damien Octeau, Daniel Luchaup, Matthew Dering, Somesh Jha, and Patrick McDaniel. 2015. Composite constant propagation: Application to android inter-component communication analysis. In *IEEE International Conference on Software Engineering (ICSE).*

[37] Oleavr. 2021. Frida binary instrumentation toolkit. https://frida.re.

[38] Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement learning based curiosity-driven testing of android applications. In *International Symposium on Software Testing and Analysis (ISSTA).*

[39] Xiang Pan, Yinzhi Cao, Xuechao Du, Boyuan He, Gan Fang, Rui Shao, and Yan Chen. 2018. Flowcog: context-aware semantics extraction and analysis of information flow leaks in android apps. In *Usenix Security Symposium (USENIX Security).*

[40] Ted Pedersen, Siddharth Patwardhan, Jason Michelizzi, et al. 2004. WordNet:: Similarity-Measuring the Relatedness of Concepts.. In *AAAI*, Vol. 4. 25–29.

[41] Google Play. 2021. Android Apps on Google Play. https://play.google.com/store/apps.

[42] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. 2014. A machine-learning approach for classifying and categorizing android sources and sinks.. In *ISOC Network and Distributed System Security Symposium (NDSS).*

[43] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. 2016. Recon: Revealing and controlling pii leaks in mobile network traffic. In *International Conference on Mobile Systems, Applications, and Services (MobiSys).*

[44] Rovo89. 2021. Xposed Framework. https://repo.xposed.info/.

[45] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. 2016. Toward a framework for detecting privacy policy violations in android application code. In *International Conference on Software Engineering (ICSE).*

[46] Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *31st AAAI Conference on Artificial Intelligence (AAAI).*

[47] Manu Sridharan, Stephen J Fink, and Rastislav Bodik. 2007. Thin slicing. In *ACM SIGPLAN Conference on Programming Language Design Implementation (PLDI).*

[48] Ben Stock, Giancarlo Pellegrino, Frank Li, Michael Backes, and Christian Rossow. 2018. Didn't you hear me?—Towards more successful Web vulnerability notifications. In *ISOC Network and Distributed System Security Symposium (NDSS).*

[49] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. 2016. Hey, you have a problem: On the feasibility of large-scale web vulnerability notification. In *Usenix Security Symposium (USENIX Security).*

[50] Mingshen Sun, Tao Wei, and John CS Lui. 2016. Taintart: A practical multi-level information-flow tracking system for android runtime. In *ACM Conference on Computer and Communications Security (CCS).*

[51] Omer Tripp and Julia Rubin. 2014. A bayesian approach to privacy enforcement in smartphones. In *23rd Usenix Security Symposium (USENIX Security).*

[52] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie. 2018. An empirical study of android test generation tools in industrial cases. In *33rd International Conference on Automated Software Engineering (ASE).*

[53] Xiaoyin Wang, Xue Qin, Mitra Bokaei Hosseini, Rocky Slavin, Travis D Breaux, and Jianwei Niu. 2018. Guileak: Tracing privacy policy claims on user input data for android applications. In *International Conference on Software Engineering (ICSE).*

[54] Fengguo Wei, Sankardas Roy, and Xinming Ou. 2018. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. *ACM Transactions on Privacy and Security (TOPS)* (2018).

[55] Dominik Wermke, Nicolas Huaman, Yasemin Acar, Bradley Reaves, Patrick Traynor, and Sascha Fahl. 2018. A large scale investigation of obfuscation use in google play. In *Annual Computer Security Applications Conference (ACSAC).*

[56] Michelle Y Wong and David Lie. 2018. Tackling runtime-based obfuscation in Android with {TIRO}. In *Usenix Security Symposium (USENIX Security).*

[57] Shengqu Xi, Shao Yang, Xusheng Xiao, Yuan Yao, Yayuan Xiong, Fengyuan Xu, Haoyu Wang, Peng Gao, Zhuotao Liu, Feng Xu, et al. 2019. DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps. In *ACM Conference on Computer and Communications Security (CCS).*

[58] Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X Sean Wang. 2013. Appintent: Analyzing sensitive data transmission in android for privacy leakage detection. In *ACM Conference on Computer and Communications Security (CCS).*

[59] Jie Zhang. 2021. Multi Class Text Classification CNN RNN. https://github.com/jiegzhan/multi-class-text-classification-cnn-rnn.

[60] Yajin Zhou, Lei Wu, Zhi Wang, and Xuxian Jiang. 2015. Harvesting developer credentials in android apps. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec).*

[61] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven M Bellovin, and Joel Reidenberg. 2017. Automated Analysis of Privacy Requirements for Mobile Apps. In *ISOC Network and Distributed System Security Symposium (NDSS).*

[62] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. 2019. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *IEEE Symposium on Security and Privacy (S&P).*