A Large-Scale Empirical Study on Android Runtime-Permission Rationale Messages

Xueqing Liu, Yue Leng
Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL, USA
{xliu93,yueleng2}@illinois.edu

Wei Yang
Department of Computer Science
University of Texas, Dallas
Richardson, TX, USA
weiyang.utd@gmail.com

Wenyu Wang, Chengxiang Zhai, Tao Xie
Department of Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL, USA
{wenyu2,czhai,taoxie}@illinois.edu

***Abstract*—After Android 6.0 introduces the runtime-permission system, many apps provide runtime-permission-group rationales for the users to better understand the permissions requested by the apps. To understand the patterns of rationales and to what extent the rationales can improve the users' understanding of the purposes of requesting permission groups, we conduct a large-scale measurement study on five aspects of runtime rationales. We have five main findings: (1) less than 25% apps under study provide rationales; (2) for permission-group purposes that are difficult to understand, the proportions of apps that provide rationales are even lower; (3) the purposes stated in a significant proportion of rationales are incorrect; (4) a large proportion of customized rationales do not provide more information than the default permission-requesting message of Android; (5) apps that provide rationales are more likely to explain the same permission group's purposes in their descriptions than apps that do not provide rationales. We further discuss important implications from these findings.**

***Index Terms*—Android Security, Runtime Permission, Rationale, Natural Language Processing**

I. Introduction

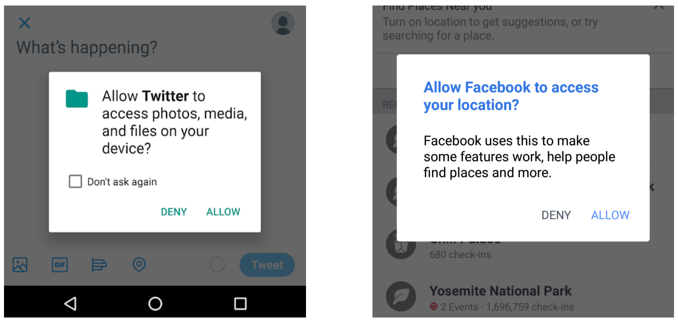
Mobile security and privacy are two challenging tasks [1]–[7]. Recently user privacy issues gather tremendous attention after the Facebook-Cambridge Analytica data scandal [8]. Android's current solution for protecting the users' private data resources mainly relies on its sandbox mechanism and the permission system. Android permissions control the users' private data resources, e.g., locations and contact lists. The permission system regulates an Android app to request permissions, and the app users must grant these permissions before the app can get access to the users' sensitive data.

In earlier versions of Android, permissions are requested at the installation time. However, studies [3], [5] show that the install-time requests cannot effectively warn the users about potential security risks. The users are often not aware of the fact that permissions are requested, and the users also have poor understandings on the meanings and purposes of using the permissions [3], [9]. It is a critical task to educate the users by explaining permission purposes so that the users can better understand the purposes [5], [10], [11].

Since Android 6.0 (Marshmallow), the permission system has been replaced by a new system that requests permission

(a) Default permission-requesting message for the permission group `STORAGE` in Android.

(b) A runtime-permission-group rationale provided by the app for the permission group `LOCATION`.



What's happening?

Allow **Twitter** to access photos, media, and files on your device?

Don't ask again

DENY ALLOW

Turn on location to get suggestions, or try searching for a place.

Allow Facebook to access your location?

Facebook uses this to make some features work, help people find places and more.

DENY ALLOW

680 check-ins

Yosemite National Park
2 Events · 1,696,759 check-ins

Fig. 1

groups [12] at runtime. An example of runtime-permission-group requests is in Figure 1a, where Android shows the default permission-requesting message for the permission group `STORAGE`[1]. The runtime model has three advantages over the old model. (1) It gives the users more warnings than the install-time model. (2) It allows the users to control an app's privileges at the permission-group level. (3) It gives apps the opportunity to embed their permission-group requests in contexts, so that the requests are self-explanatory. For example, in Figure 1a, a request for accessing the user's gallery is prompted when she is about to send a Tweet.

With the runtime-permission system, each Android app can leverage a dialog to provide a customized message for explaining its unique purpose of using the permission group. In Figure 1b, we show an example of such messages from the *Facebook* app for explaining the purpose of requesting the user's location: "*Facebook uses this to make some features work...*". Such customized messages are called *runtime-permission-group rationales*. Runtime-permission-group rationales are often displayed before or after the permission-requesting messages, or upon the starting of the app. For the rest of this paper, for simplicity, whenever the context refers to a runtime-permission-group rationale or a runtime-

[1]The permission-requesting message is the message displayed in the permission-requesting dialog (Figure 1a). For each permission group, this message is fixed across different apps. For example, the permission-requesting message for `STORAGE` is *Allow **appname** to access photos, media and files on your device?*

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

permission-group request, we use the term *rationale*, *runtime rationale*, and *permission-group rationale* in short for *runtime-permission-group rationale*; we use the term *permission request(-ing message)* in short for *runtime-permission-group request(-ing message)*.

There are three main reasons why runtime rationales are useful in the new permission system. (1) *Challenge in Explaining Background Purposes*. Although the runtime system allows permission-group requests to be self-explanatory in contexts, there exist cases where the permission groups are used in the background (e.g., read phone number, SMS) [13]. As a result, there does not exist a user-aware context for asking such permission groups. (2) *Challenge in Explaining non-Straightforward Purposes*. When the purpose of requesting a permission group is not straightforward, such as when the permission group is not for achieving a primary functionality, the context itself may not be clear enough to explain the purpose. For example, when the user is about to send a Tweet (Figure 1a), she may not notice that the location permission group is requested. (3) *Effectiveness of Natural Language Explanations*. Prior work [5] shows that the users find the usage of a permission better meets their expectation when the purpose of using such permission is explained with a natural language sentence. Furthermore, user studies [14] on Apple's iOS runtime-permission system also demonstrate that displaying runtime rationales can effectively increase the users' approval rates.

The effectiveness of explaining permission purposes relies on the contents of the explanation sentences [5]. Because the rationale sentences are created by apps, the quality of such rationales depends on how individual apps (developers) make decisions for providing rationales. Three essential decisions are (1) which permission group(s) the app should explain the purposes for; (2) for each permission group, what words should be used for explaining the permission group's purpose; (3) how specific the explanation should be.

In this paper, we seek to answer the following questions: (1) what are the common decisions made by apps? (2) how are such decisions aligned with the goal of improving the users' understanding of permission-group purposes? To understand the general patterns of apps' permission-explaining behaviors, we conduct the first large-scale empirical study on runtime rationales. We collect an Android 6.0+ dataset consisting of 83,244 apps. From these apps, we obtain 115,558 rationale sentences. Our study focuses on the following five research questions.

RQ1: Overall Explanation Frequency. We investigate the overall frequency for apps to explain permission-group purposes with rationales. The result can help us understand whether the developers generally acknowledge the usefulness of runtime rationales, and whether the users are generally warned for the usages of different permission groups.

RQ2: Explanation Frequency for non-Straightforward vs. Straightforward Purposes. Prior work [5], [15] finds that the users have different expectations for different permission purposes. The Android official documentation [16] suggests that apps provide rationales when the permission group's purposes are not straightforward. Therefore, we investigate whether apps more frequently explain non-straightforward purposes than straightforward ones. The result can help us understand the helpfulness of rationales with the users' understandings of permission-group purposes.

RQ3: Incorrect Rationales. We study the population of rationales where the stated purpose is different from the true purpose, i.e., the rationales are incorrect. Such study is related to user expectation, because incorrect rationales may confuse the users and mislead them into making wrong security decisions.

RQ4: Rationale Specificity. How exactly do apps explain purposes of requesting permission groups? How much information do rationales carry? Do rationales provide more information than the permission-requesting message? Do apps provide more specific rationales for non-straightforward purposes than for straightforward purposes?

RQ5: Rationales vs. App Descriptions. Are apps that provide rationales more likely to explain the same permission group's purpose in the app description than apps that do not provide rationales? Are the behaviors of explaining a permission group's purposes consistent in the app description and in rationales? Do more apps explain their permission-group purposes in the app description than in rationales?

The rest of this paper is organized as follows. Section II introduces background and related work, Section III describes the data collection process. Sections IV- VIII answer RQ1-RQ5. Sections IX- XI discuss threats to validity, implications, and conclusion of our study.

II. Background and Related Work

Android Permissions and the Least-Privilege Principle. A previous study [2] shows that compared with attack-performing malware, a more prevalent problem in the Android platform is the *over-privilege* issue of Android permissions: apps often request more permissions than necessary. Felt *et al.* [3] evaluate 940 apps and find that one-third of them are over-privileged. Existing work leverages static-analysis techniques [2], [17] and dynamic-analysis techniques [1] to build tools for analyzing whether an app follows the *least-privilege principle*. The runtime-permission-group rationales we study are for helping the users make decisions on whether a permission-group request is over-privileged.

User Expectation. Over time, the research literature on Android privacy has focused on studying whether and how an app's permission usage meets the users' expectation [4], [5], [10], [18]–[23]. In particular, Lin *et al.* [5] find that the users' security concern for a permission depends on whether they can expect the permission usage. Jing *et al.* [15] further find that even in the same app, the users have different expectations for different permissions. For example, in the *Skype* app, the users find the microphone permission more straightforward than the location permission. The Android official documentation [16] also points out this difference and suggests that app developers provide more runtime-permission-group rationales for purposes that are not straightforward to expect.

The research literature on user expectation can be categorized into three lines of work. The first line of work is on detecting contradictions between the code behavior and the user interface [18], [24]. The second line of work is on improving existing interfaces to enhance the users' awareness of permission usages [4], [13], [20]–[22], [25]. This line of work includes privacy nudging [4], access control gadget [22], and mapping between permissions and UI components [25]. In particular, Nissenbaum *et al.* [20] first propose the concept of privacy as the *contextual integrity*; i.e., the users' decision-making process for privacy relies on the contexts [13], [21], [26], [27]. The runtime-permission system incorporates the contextual integrity by allowing apps to ask for permission groups within the context. The third line of work is on using natural language sentences to represent or enhance the users' expectation regarding the permission usages [5], [10], [19], [28]. For example, Lin *et al.* [5] find that the users of an app are more comfortable with using the app when the app provides clarifications for the permission purposes than they do not provide such clarifications. Pandita *et al.* [10] further extract permission explaining sentences from app descriptions. Our study results presented in Section VIII show that apps explain purposes of requesting permission groups more frequently in the rationales than in the description.

Runtime Permission Groups and Runtime Rationales. Since the launch of the runtime-permission system, another line of work [5], [14], [29] (including our work) focuses on the runtime-permission system and the users' decisions on such system. In particular, Bonne *et al.* [29] conduct a study similar to the study by Lin *et al.* [5] under the runtime-permission system, showing the users' security decisions in the runtime system also rely on their expectations of the permission usages. The closest to our work is the study by Tan *et al.* [14] on the effects of runtime rationales in the iOS system. Their user-study results show that rationales can improve the users' approval rates for permission requests and increase the comfortableness for the users to use the app. Although they have not observed a significant correlation between the rationale contents and the approval rates, such observations may be due to the fact that only one fake app is examined with limited user feedback. As a result, such unrelatedness cannot be trivially generalized to our case. Wijesekera et al. [30] redesigns the timing of runtime prompts to reduce the *satisficing* and *habituation* issues [31]–[34]. Both Wijesekera *et al.* [30] and Olejnik *et al.* [35] leverage machine learning techniques to reduce user efforts in making decisions for permission requests.

III. Data Collection

A. Crawling Apps

Since the launch of Android 6.0, many apps have migrated to support the newer versions of Android. To obtain as many Android 6.0+ apps as possible, we crawl apps from the following two sources: (1) we crawl the top-500 apps in each category from the Google Play store, obtaining 23,779 apps in total; (2) we crawl 482,591 apps from APKPure [36], which is another app store with copied apps (same ID, same category, same description, etc.) from the Google Play store[2]. From the two sources, we collect 494,758 apps. Among these apps, we find 83,244 apps that (1) contain version(s) under Android 6.0+; (2) request at least 1 out of the 9 dangerous permission groups (Table I). We use these 83,244 apps as the dataset in this paper[3].

B. Annotating Permission-group Rationales

For each app found in the preceding step, we annotate and extract runtime rationales from the app. Same as other static user interface texts, runtime rationales are stored in an app's `./res/values/strings.xml` file. Each line of this file contains a rationale's name and the content of the rationale.

The size of our dataset dictates that it is intractable to manually annotate all the string variables. As a result, we leverage two automatic sentence-annotating techniques: (1) keyword matching; (2) CNN sentence classifier. The automatic annotation is a two-step process.

Annotating Rationales for All Permission Groups. For the first step, we design a keyword matching technique to annotate whether a string variable contains mentions of a permission group. More specifically, we assign a binary label to each string variable by matching the variable's name or content against 18 keywords referring to permission groups, including "*permission*", "*rationale*", and "*toast*"[4]. To estimate the recall of keyword matching, we randomly sample 10 apps and inspect their string resource files. The result of our inspection shows that such keyword matching found all the rationales in the 10 apps.

Annotating Rationales for the 8 Dangerous Permission Groups[5]. For the second step, we use the CNN sentence classifier [38], [39] to annotate the outputs from the first step. The annotations indicate whether each rationale describes 1 of the 9 dangerous permission groups [12]. The 9 permission groups contain 26 permissions. These permission groups' protection levels are dangerous and the purposes of requesting these permission groups are relatively straightforward for the users to understand.

For each permission group, we train a different CNN sentence classifier. We manually annotate 200$\sim$700 rationales as the training examples for each classifier. After applying CNN, we estimate the classifier's false positive rate (FP) and false negative rate (FN) by inspecting 100 output examples in each permission group. The average FP (FN) over the 8 permission groups is 5.1% (6.8%) and the maximum FP (FN) is 13% (16%).

In total, CNN annotates 115,558 rationales, which can be found on our project's website [37].

[2]We are not able to collect all these apps from the Google Play store, due to its anti-theft protection that limits the downloading scale.

[3]To the best of our knowledge, this dataset is the largest app collection on runtime rationales; it is orders of magnitude larger than other runtime-rationale collections in existing work [13], [14].

[4]The complete list of the 18 keywords can be found on our project website [37].

[5]In this paper, we skip the `BODY_SENSORS` permission group because it contains too few rationales.

TABLE I: The number of the used apps (the #used apps column), the explained apps (the #explained apps column), and the proportion of explained apps in the used apps (the %exp column). We sort the permission groups by #used apps.

permgroup	#used apps	#explain -ed apps	%exp	%exp (top)
STORAGE	73,031	14,668	20.2%	**28.3%**
LOCATION	32,648	7,088	**21.6%**	**30.7%**
PHONE	31,198	2,070	6.7%	11.0%
CONTACTS	23,492	2,607	11.1%	17.7%
CAMERA	16,557	4,235	**25.6%**	**37.7%**
MICROPHONE	9,130	2,152	**23.5%**	28.0%
SMS	4,589	589	12.8%	16.0%
CALENDAR	2,492	357	14.2%	22.6%
BODY_SENSORS	122	16	13.1%	15.4%
overall	83,244	19,879	23.8%	33.9%

Discussion. One caveat of our data collection process is that the rationales in string resource files are only *candidates* for runtime prompts. That is, they may not be displayed to the users. The reason why we do not study only the actually-displayed rationales is that such study relies on dynamic-analysis techniques, which limit the scale of our study subjects.

IV. RQ1: Overall Explanation Frequency

In the first step of our study, we investigate the proportion of apps that provide permission-group rationales to answer RQ1: how often do apps provide permission-group rationales? For each of the 9 permission groups, we count how many apps in our dataset request the permission group; we denote this value as #used apps. Among these apps, we further count how many of them explain the requested permission group's purposes with rationales; we denote this value as #explained apps. Given the two values, we measure the *explanation proportion* of a group of apps:

Definition 1 (Explanation proportion)**.** *Given a group of apps, its explanation proportion of a permission group is the proportion of apps in that group to explain the purposes of requesting the permission group, i.e., #explained apps / #used apps. We denote the explanation proportion as %exp.*

In Table I, we show the values of #used apps, #explained apps, and %exp for each permission group. In addition, we compute the %exp value for only the categorical top-500 apps; we denote this value as %exp (top).

Result Analysis. From Table I we can observe three findings. (1) Overall, 23.8% apps provide runtime rationale. (2) The top-500 apps more frequently explain the purposes of using permission groups than the overall apps do. (3) The purposes of the four permission groups STORAGE, LOCATION, CAMERA, and MICROPHONE are more frequently explained than the other five permission groups.

Finding Summary for RQ1. 23.8% apps provide runtime rationales for their permission-group requests. Among all the permission groups, four groups' purposes are explained more often than the other permission groups. This result may imply that app developers are less familiar with the purposes of PHONE and CONTACTS.

TABLE II: The app sets for measuring the correlation between the usage proportion and the explanation proportion. The apps in each set share the same purpose (the purpose column) to use the primary permission group (the permgroup column) with the usage proportion (the %use column).

appset	permgroup	purpose	%use	#apps
file mgr	STORAGE	file managing	95.4%	499
video players	STORAGE	store video	96.6%	1,306
photography	STORAGE	store photos	99.7%	3,534
maps&navi	LOCATION	GPS navigation	92.6%	1,541
weather	LOCATION	local weather	95.4%	908
travel&local	LOCATION	local search	87.8%	2,647
lockscreen	PHONE	answer call wh -en screen locked	82.6%	425
voip call	PHONE	make calls	84.9%	847
caller id	PHONE	caller id	92.0%	175
caller id	CONTACTS	caller id	86.7%	196
mail	CONTACTS	auto complete	77.1%	140
contacts	CONTACTS	contacts backup	85.8%	259
flashlight	CAMERA	flashlight	96.6%	298
qrscan	CAMERA	qr scanner	88.4%	155
camera	CAMERA	selfie&camera	71.4%	749
recorder	MIC	voice recorder	75.7%	559
video chat	MIC	video chat	77.0%	139
sms	SMS	sms	60.4%	379
calendar	CALEND	calendar	36.0%	300

V. RQ2: Explanation Frequency for Non-straightforward vs. Straightforward Purposes

In the second part of our study, we seek to *quantitatively* answer RQ2: do apps provide more rationales for non-straightforward permission-group purposes than for straightforward permission-group purposes?

It is challenging to *precisely* measure the straightforwardness for understanding the purpose of requesting a permission group. The reason for such challenge is that such straightforwardness relies on each user's existing knowledge, which varies from user to user. Therefore, we propose to *approximate* the straightforwardness by measuring the *usage proportion* of a permission group in *a set of apps*:

Definition 2 (Usage proportion)**.** *Given a set of apps, its usage proportion (denoted as %use) of a permission group is the proportion of the apps (in this set) that request the permission group.*

Our approximation is based on the observation that if a permission group is frequently used by a set of apps, the permission-group purpose in that app set is often also straightforward to understand. For example, in a camera app, the users are more likely to understand the purpose of the camera permission group than the location permission group [16]; meanwhile, our statistics show that camera apps also more frequently request the camera permission group (71.4%) than the location permission group (27.0%).

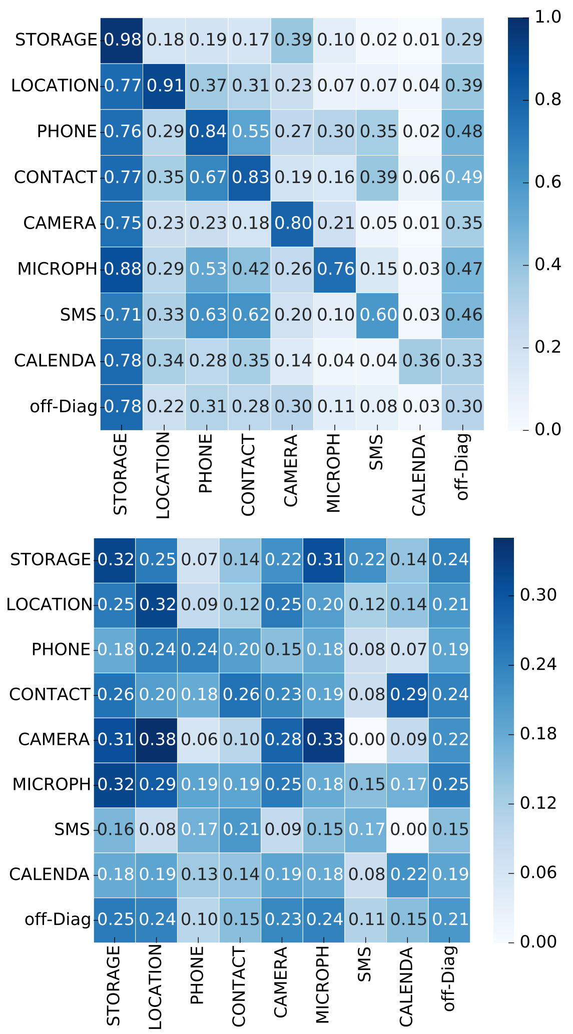


Fig. 2: The usage proportion (top) and the explanation proportion (bottom) of the app sets in Table II. Each element at (Q, P) shows the proportion of apps in set Q to use/explain the purpose of permission group P.

To answer RQ2, we first introduce the definitions of the primary permission group.

Definition 3 (Primary Permission Group)**.** *Given a set of apps that share the same primary functionality, if any app relies on (does not rely on) requesting a permission group to achieve that primary functionality, we say that this permission group is a primary (non-primary) permission group to this app set, and this app set is a primary (non-primary) app set to this permission group. An example of such primary (non-primary) pairs is GPS navigation apps and* `LOCATION` *(*`CAMERA`*) permission group.*

To study the relation between the straightforwardness of permission-group purposes and explanation proportions, we leverage the following three-step process. (1) For each permission group P, we use keyword matching to identify 1$\sim$3 app sets such that P is a primary permission group to these app sets. (2) For each permission group Q, we merge its primary app sets to obtain a larger primary app set for Q. (3) For each permission group P and the merged app sets for each permission group Q, we compute the proportion for app set Q to use/explain P, obtaining two 8×8 matrices. We show all the app sets in Table II, and the two matrices in Figure 2. In each matrix in Figure 2, each row corresponds to a merged app set Q and each column corresponds to a permission group P. For each row/column, we also compute the average over its off-diagonal elements and show these values in an additional column/row named off-Diag. That is, elements in off-Diag show the average over non-primary permission groups/app sets.

Why Using Primary Permission Groups? By introducing primary permission groups, we are able to identify permission-group purposes that are clearly straightforward (Table II), so that the boundaries between straightforward purposes and non-straightforward purposes are relatively well defined. We can observe such boundaries from the usage proportion matrix (Figure 2, top).

Result Analysis. We can observe the following findings from the explanation matrix in Figure 2 (bottom). (1) By comparing every diagonal element with its two off-Diag counterparts, we can observe that the diagonal elements are usually larger, indicating that straightforward permission-group purposes are explained more frequently than non-straightforward ones. On the other hand, there exist a few exceptional cases in `LOCATION`, `MICROPHONE`, `SMS`, and `CALENDAR` where at least one off-diagonal element is larger than the diagonal element, indicating that non-straightforward permission-group purposes are explained more frequently in these cases. (2) By comparing the elements in the off-Diag row, we find that the permission groups for which non-straightforward purposes are most explained are `STORAGE`, `LOCATION`, `CAMERA`, and `MICROPHONE`. Such result is consistent with the overall explanation proportions in Table I.

Measuring Correlation Over All Apps. Because the app sets in Table II cover only a subset of apps, we further design the second measurement study to capture all apps in our dataset. The second study includes the following two-step process. (1) Based on the app categories in the Google Play store, we partition all apps into 35 sets. After the partition, the two permission groups `SMS` and `CALENDAR` contain too few rationales in each app set, and therefore we discard these two permission groups. (2) For each permission group, we compute all its usage proportions and explanation proportions in the 35 app sets, and test the Pearson correlation coefficient [40] between the usage proportions and explanation proportions. In Table III, we show the results of the Pearson tests. We can observe that 4 out of the 6 tests show significantly positive correlation, i.e., straightforward purposes are usually more frequently explained. Such results are generally consistent with

TABLE III: The Pearson correlation tests of each permission group, between the usage proportion and the explanation proportion on the 35 Play-store app sets.

STORAGE		LOC		PHONE		CONTACT		CAMERA		MIC	
r	p	r	p	r	p	r	p	r	p	r	p
.4	8e-3	.6	1e-3	.5	6e-2	.8	1e-3	-.5	2e-2	.2	.5

the results in Figure 2.

Finding Summary for RQ2. Overall, apps *have not* provided more runtime rationales for non-straightforward permission-group purposes than for straightforward ones except for a few cases. This result implies that the majority of apps *have not* followed the suggestion from the Android official documentation [16] to provide rationales for non-straightforward permission-group purposes.

VI. RQ3: Incorrect Rationales

In the third part of our study, we investigate the correctness of permission-group rationales. We seek to answer RQ3: does there exist a significant proportion of runtime rationales where the stated purposes do not match the true purposes?

It is challenging to derive an app's true purpose for requesting a permission group. However, we can coarsely differentiate between purposes by checking the permissions under a permission group. Among the 9 permission groups in Android 6.0 and higher versions, 6 permission groups each contain more than one permission [12]. For example, the PHONE permission group controls the access to phone-call-related sensitive resources, and this permission group contains 9 phone-call-related permissions: CALL_PHONE, READ_CALL_LOG, READ_PHONE_STATE, etc. By examining whether the app requests READ_CALL_LOG or READ_PHONE_STATE, we can differentiate between the purposes of reading the user's call logs and accessing the user's phone number.

In order to easily identify the mismatches between the stated purpose and the true purpose, we study 3 permission groups consisting of relatively diverse permissions: PHONE, CONTACTS, and LOCATION. In particular, each of the 3 groups contains 1 permission such that 90% apps requesting the group have requested that permission (whereas other permissions in the same group are requested less frequently); therefore, we name such permission a *basic permission*. The basic permissions of PHONE, CONTACTS, and LOCATION are READ_PHONE_STATE, GET_ACCOUNTS, and ACCESS_COARSE_LOCATION, respectively.

Definition 4 (Apps with Incorrect Rationales)**.** *We identify two cases for an app to contain incorrect rationale(s): (1) all the rationales state that the app requests only the basic permission, but in fact, the app has requested other permissions (in the same permission group); (2) the app requests only the basic permission, but it contains some rationales stating that it has requested other permissions (in the same permission group).*

How many apps does each of the two incorrect cases contains? Both cases can mislead the user to make wrong decisions. For case (1), the user may grant the permission-group request with the belief that she has granted only the basic permission, but in fact she has granted other permissions. For case (2), the user may deny the permission-group request, because the stated purpose of such permission group seems to be unrelated to the app's functionality, e.g., when a music player app requests the READ_PHONE_STATE permission only to pause the music when receiving phone calls, the rationale can raise the user's security concern by stating that the music app needs to make a phone call. After the user denies the phone permission group, the app also loses the access to pausing the music.

TABLE IV: The upper table shows the criteria for annotating the basic permission and other permissions in the same permission group. The lower table shows the estimated lower bounds on the numbers of apps containing incorrectly stated rationales.

		CONTACTS	PHONE	LOCATION
annotate criterion	basic per-mission class (a)	google account/ sign in/ email address	pause incoming call/ imei/ identity/ number/ cellular	coarse loc /area/region /approximate /beacon /country
	other per-missions class (b)	contacts/ friends/ phonebook	make call/ call phone/ call logs	driving/ fine loc/ coordinate

incorrect apps	case (1)	#err	%err	#err	%err	#err	%err
		93	4.6	139	11.3	9	0.1
	case (2)	#err	%err	#err	%err	#err	%err
		76	13.2	37	4.2	3	0.6

To study the populations of the two preceding incorrect cases, we again leverage the aforementioned CNN sentence classifier [38]. We classify each runtime rationale into one of the following three classes: (a) the rationale states the purpose of requesting a basic permission; (b) the rationale states the purpose of requesting a permission other than the basic permission; (c) neither (a) nor (b). For each of the three permission groups, we manually annotate 600∼900 rationales as the training data. After we obtain the predicted labels, we manually judge the resulting rationales that are predicted as (a) or (b) to make sure that there do not exist false positive annotations for incorrect case (1) or (2). In Table IV, we show the lower-bound estimations (#err and %err) of the two incorrect cases' populations. We also show the detailed criteria of our annotations for (a) and (b). The list of incorrect rationales and their apps can be found on our project website [37].

Result Analysis. From Table IV we can observe that there exist a significant proportion of incorrectly stated runtime rationales, especially in the incorrect case (1) of the phone permission group and the incorrect case (2) of the contacts permission group. In contrast, there exist fewer incorrect cases in the location permission group. The reason for the location permission group to contain fewer incorrect cases may be that the majority of apps claim only the usage of location, without specifying whether the requested location is fine or coarse. The contacts and phone permission groups contain more diverse purposes than the location group does, and our study results show that a significant proportion of apps requesting the two groups state the wrong purposes. For example, a significant number of FM radio apps state in the rationales that these apps *only* need to use the phone state to pause the radio when receiving incoming calls; however, these apps have also requested the CALL_PHONE permission, indicating that if the

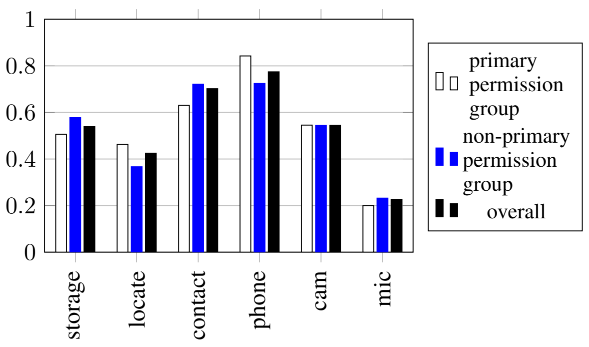


Fig. 3: The proportions of non-redundant rationales in each permission group.

user grants the permission group, these apps also gain the access to *making phone calls* within the app.

Finding Summary for RQ3. There exist a significant proportion of incorrect runtime rationales for the CONTACTS and the PHONE permission groups. This result implies that apps may have confused the users by stating the incorrect permission-group purposes for PHONE and CONTACTS.

VII. RQ4: Rationale Specificity

In the fourth part of our study, we look into the informativeness of runtime rationales. In particular, we seek to answer RQ4: do rationales (e.g., the rationale in Figure 1b) provide more specific information than the system-provided permission-requesting messages (e.g., the message in Figure 1a)?

Definition 5 (Redundant Rationales). *If a runtime rationale states only the fact that the app is requesting the permission group, i.e., it does not provide more information than the permission-requesting message, we say that the rationale is redundant, and otherwise non-redundant.*

Among all the runtime rationales, how many are non-redundant ones? How much do the proportions of non-redundant rationales in each permission group vary across permission groups?

To study the population of non-redundant rationales, we leverage the named entity tagging (NER) technique [41]. The reason for us to leverage the NER technique is our observation that non-redundant rationales usually use some words to state the more specific purposes than the fact of using the permission group. Moreover, these purpose-stating words usually appear in textual patterns. As a result, we can leverage such textual patterns to detect non-redundant rationales. For example, in the following rationale, the words tagged with "*S*" explain the *specific* purpose of using the permission group PHONE, and the words tagged with *_O* are other words: "*this_O radio_O application_O would_O like_O to_O use_O the_O phone_O permission_O to_S pause_S the_S radio_S when_S receiving_S incoming_S calls_S*". We train a different NER tagger for each of the top-6 permission groups in Table I[6]. For each permission group, we manually annotate 200∼1,000 training examples. To evaluate the performance of our NER tagger, we randomly sample 100 rationales from NER's output for each permission group, and manually judge these sampled rationales. Our judgment results show that NER's prediction accuracy ranges from 85% to 94%. The lists of redundant and non-redundant rationales tagged by NER can be found on our project website [37]. Next, we obtain the proportions of non-redundant rationales in each permission group. We plot these proportions in Figure 3.

[6]We skip SMS and CALENDAR, because they both contain too few rationales for estimating the proportions of non-redundant rationales.

Result Analysis. We can observe three findings from Figure 3 and additional experiments. (1) The proportions of redundant runtime rationales range from 23% to 77%. (2) While the two permission groups PHONE and CONTACTS have the lowest explanation proportions (Figure 2), they have the highest non-redundant proportions. The reason why most phone and contacts rationales are non-redundant is that they usually specify whether the permission group is used for the basic permission or other permissions. (3) We also study the proportions of non-redundant rationales in the app sets defined in Table II, but we have not observed a significant correlation between the usage proportions and the non-redundant proportions.

Finding Summary for RQ4. A large proportion of the runtime rationales have not provided more specific information than the permission-requesting messages. The rationales in PHONE and CONTACTS are most likely to explain more specific purposes than the permission-requesting messages. This result implies that a large proportion of the rationales are either unnecessary or should be more specifically explained.

VIII. RQ5: Rationales vs. App Descriptions

In the fifth part of our study, we look into the correlation between the runtime rationales and the app description. We seek to answer RQ5: how does explaining a permission group's purposes in the runtime rationales relate to explaining the same permission group's purposes in the app description? Are apps that provide rationales more likely to explain the same permission group's purposes in the app description than apps that do not provide rationales?

To identify apps that explain the permission-group purposes in the description, we leverage the WHYPER tool and the keyword matching technique [10]. WHYPER is a state-of-the-art tool for identifying permission-explaining sentences. We apply WHYPER on the CONTACTS and the MICROPHONE permission groups. Because WHYPER [42] does not provide the entire pipeline solution for other frequent permission groups, we use the keyword matching technique to match sentences for another permission group LOCATION. Prior work [11] also leverages keyword matching for efficient processing. We show the results in Table V.

Result Analysis. From Table V, we can observe two findings. (1) In two out of the three cases, the correlations are significantly positive. Therefore, an app that provides runtime rationales is also more likely to explain the same permission group's purpose in the description. (2) There exist

TABLE V: The number of apps that explain a permission group's purposes in the app description (the #apps descript column), in the rationales (the #apps rationales column), in both (the #apps both column), and the Pearson correlation coefficients between whether an app explains a permission group's purpose in the description vs. rationales (the Pearson column).

	#apps descript	#apps rationales	#apps both	Pearson
LOCATION	5,747	7,088	2,028	(0.15, 1.86e-168)
CONTACTS	1,542	2,607	394	(0.12, 1.5e-78)
MICROPH	957	2,152	245	(0.02, 0.12)

more apps using runtime rationales to explain the permission-group purposes than apps that use the descriptions.

Finding Summary for RQ5. The explanation behaviors in the description and in the runtime rationales are often positively correlated. Moreover, more apps use runtime rationales to explain purposes of requesting permission groups than using the descriptions. This result implies that apps' behaviors of explaining permission-group purposes are generally consistent across the descriptions and the rationales.

IX. Threats to Validity

The threats to external validity primarily include the degree to which the studied Android apps or their runtime rationales are representative of true practice. We collect the Android apps from two major sources, one of which is the Google Play store, the most popular Android app store. Such threats could be reduced by more studies on more Android app stores in future work. The threats to internal validity are instrumentation effects that can bias our results. Faults in the used third-party tools or libraries might cause such effects. To reduce these threats, we manually double check the results on dozens of Android apps under analysis. Human errors during the inspection of data annotations might also cause such effects. To reduce these threats, at least two authors of this paper independently conduct the inspection, and then compare the inspection results and discuss to reach a consensus if there is any result discrepancy.

X. Implications

In this paper, we attain multiple findings for Android runtime rationales. These findings imply that developers may be less familiar with the purposes of the PHONE and CONTACTS permission groups and some rationales in these groups may be misleading (RQ1 and RQ3); the majority of apps have not followed the suggestion for explaining non-straightforward purposes [16] (RQ2); a large proportion of rationales may either be unnecessary or need further details (RQ4); and apps' explanation behaviors are generally consistent across the descriptions and the rationales (RQ5). Such findings suggest that the rationales in existing apps may not be optimized for the goal of improving the users' understanding of permission-group purposes. Based on these implications, we propose two suggestions on the system design of the Android platform.

Official Guidelines or Recommender Systems. It is desirable to offer an official guideline or a recommender system for suggesting which permission-group purposes to explain [11], e.g., on the official Android documentation or embedded in the IDE. For example, such recommender system can provide a list of functionalities, so that the developer can select which functionalities are used by the app. Based on the developer's selections, the system scans the permission-group requests by the app, and lets the developer know which permission group(s)'s purposes may look non-straightforward to the users. In addition, the system can suggest rationales for the developers to adapt or to adopt [11].

Controls over Permissions for the Users. When a permission group contains multiple permissions, such design increases the challenges and errors in explaining the purposes of requesting such permission group. It is interesting to study whether a user actually knows which permission she has granted, e.g., does a weather app use her precise location or not? One potential approach to improve the users' understanding of permission-group purposes is to further scale down the permission-control granularity from the user's end. For example, the "permission setting" in the Android system can display a list showing whether each of the user's *permissions* (instead of permission groups) has been granted; and doing so also gives the users the right to revoke each permission individually.

XI. Conclusion

In this paper, we have conducted the first large-scale empirical study on runtime-permission-group rationales. We have leveraged statistical analysis for producing five new findings. (1) Less than one-fourth of the apps provide rationales; the purposes of using PHONE and CONTACTS are the least explained. (2) In most cases, apps explain straightforward permission-group purposes more than non-straightforward ones. (3) Two permission groups PHONE and CONTACTS contain significant proportions of incorrect rationales. (4) A large proportion of the rationales do not provide more information than the permission-requesting messages. (5) Apps' explanation behaviors in the rationales and in the descriptions are positively correlated. Our findings indicate that developers may need further guidance on which permission groups to explain the purposes and how to explain the purposes. It may also be helpful to grant the users controls over each permission.

Our study focuses on analyzing natural language rationales. Besides the rationales, other UI components (e.g., layout, images/icons, font size) can also affect the users' decision making. In future work, we plan to study the effects of runtime-permission-group requests when considering these factors, and study ways to encourage the developers to provide higher-quality warnings than the current ones.

Acknowledgment. We thank the anonymous reviewers and Xiaofeng Wang for their useful suggestions. This work was supported in part by NSF CNS-1513939, CNS-1408944, CCF-1409423, and CNS-1564274.

References

[1] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. D. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*. ACM, 2014, pp. 393–407.

[2] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the ACM Conference on Computer and Communications security*. ACM, 2011, pp. 627–638.

[3] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Symposium on Usable Privacy and Security*. USENIX Association, 2012, pp. 3:1–3:14.

[4] H. Almuhimedi, F. Schaub, N. M. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal, "Your location has been shared 5,398 times! A field study on mobile app privacy nudging," in *Proceedings of the Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 787–796.

[5] J. Lin, N. M. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang, "Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing," in *Proceedings of the ACM Conference on Ubiquitous Computing*. ACM, 2012, pp. 501–510.

[6] J. Lin, B. Liu, N. M. Sadeh, and J. I. Hong, "Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings," in *Proceedings of the Symposium on Usable Privacy and Security*. USENIX Association, 2014, pp. 199–212.

[7] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society, 2015, pp. 303–313.

[8] "Facebook and cambridge analytical data breach," https://en.wikipedia.org/wiki/Facebook_and_Cambridge_Analytica_data_breach, accessed: 2018-07-27.

[9] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. M. Sadeh, and D. Wetherall, "A conundrum of permissions: Installing applications on an Android smartphone," in *Financial Cryptography Workshops*. Springer, 2012, pp. 68–79.

[10] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WHYPER: Towards automating risk assessment of mobile applications," in *Proceedings of the USENIX Security Symposium*. USENIX Association, 2013, pp. 527–542.

[11] X. Liu, Y. Leng, W. Yang, C. Zhai, and T. Xie, "Mining Android app descriptions for permission requirements recommendation," in *Proceedings of the International Requirements Engineering Conference*. IEEE Computer Society, 2018.

[12] "Android permission groups," https://developer.android.com/guide/topics/permissions/requesting.html\#perm-groups, 2018, accessed: 2018-07-27.

[13] K. K. Micinski, D. Votipka, R. Stevens, N. Kofinas, M. L. Mazurek, and J. S. Foster, "User interactions and permission use on Android," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2017, pp. 362–373.

[14] J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. A. Wagner, "The effect of developer-specified explanations for permission requests on smartphone user behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 91–100.

[15] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "RiskMon: Continuous and automated risk assessment of mobile applications," in *Proceedings of the ACM Conference on Data and Application Security and Privacy*. ACM, 2014, pp. 99–110.

[16] "Should show request permission rationale API," https://developer.android.com/reference/android/support/v4/app/ActivityCompat#shouldShowRequestPermissionRationale(android.app.Activity,java.lang.String), 2018, accessed: 2018-07-27.

[17] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, 2012, pp. 217–228.

[18] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "AsDroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction," in *Proceedings of the International Conference on Software Engineering*. ACM, 2014, pp. 1036–1046.

[19] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the International Conference on Software Engineering*. ACM, 2014, pp. 1025–1035.

[20] H. Nissenbaum, "Privacy as contextual integrity." Washington University School of Law, 2004, pp. 101–139.

[21] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. A. Wagner, and K. Beznosov, "Android permissions remystified: A field study on contextual integrity," in *Proceedings of the USENIX Security Symposium*. USENIX Association, 2015, pp. 499–514.

[22] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2012, pp. 224–238.

[23] P. G. Kelley, L. F. Cranor, and N. M. Sadeh, "Privacy as part of the app decision-making process," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 3393–3402.

[24] B. Andow, A. Acharya, D. Li, W. Enck, K. Singh, and T. Xie, "UiRef: Analysis of sensitive user inputs in Android applications," in *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 23–34.

[25] Y. Li, Y. Guo, and X. Chen, "PERUIM: Understanding mobile application privacy with permission-UI mapping," in *Proceedings of the ACM Conference on Ubiquitous Computing*. ACM, 2016, pp. 682–693.

[26] K. Z. Chen, N. M. Johnson, V. D'Silva, S. Dai, K. MacNamara, T. R. Magrino, E. X. Wu, M. Rinard, and D. X. Song, "Contextual policy enforcement in Android applications with permission event graphs," in *Proceedings of the Network & Distributed System Security Symposium*. The Internet Society, 2013.

[27] D. Votipka, K. Micinski, S. M. Rabin, T. Gilray, M. M. Mazurek, and J. S. Foster, "User comfort with Android background resource accesses in different contexts," in *Proceedings of the Symposium on Usable Privacy and Security*. USENIX Association, 2018.

[28] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "AutoCog: Measuring the description-to-permission fidelity in Android applications," in *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, 2014, pp. 1354–1365.

[29] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft, "Exploring decision-making with Android's runtime permission dialogs using in-context surveys," in *Proceedings of the Symposium on Usable Privacy and Security*. USENIX Association, 2017, pp. 195–210.

[30] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov, "The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017, pp. 1077–1093.

[31] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer, "Here's my cert, so trust me, maybe?: Understanding TLS errors on the web," in *Proceedings of the International Conference on World Wide Web*. ACM, 2013, pp. 59–70.

[32] M. S. Wogalter, V. C. Conzola, and T. L. Smith-Jackson, "Research-based guidelines for warning design and evaluation," vol. 33, no. 3. Elsevier, 2002, pp. 219–230.

[33] M. Harbach, S. Fahl, P. Yakovleva, and M. Smith, "Sorry, I don't get it: An analysis of warning message texts," in *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 94–111.

[34] F. Schaub, R. Balebako, A. L. Durity, and L. F. Cranor, "A design space for effective privacy notices," in *Proceedings of the Symposium On Usable Privacy and Security*. USENIX Association, 2015, pp. 1–17.

[35] K. Olejnik, I. Dacosta, J. S. Machado, K. Huguenin, M. E. Khan, and J.-P. Hubaux, "SmarPer: Context-aware and automatic runtime-permissions for mobile devices," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017, pp. 1058–1076.

[36] "APKPure website," https://www.apkpure.com, 2018, accessed: 2018-07-27.

[37] "Runtime permission rationale project website," https://sites.google.com/view/runtimepermissionproject/, accessed: 2018-07-27.

[38] "A tensorflow implementation of CNN text classification," https://github.com/dennybritz/cnn-text-classification-tf, 2018, accessed: 2018-07-27.

[39] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2014, pp. 1746–1751.

[40] "Pearson correlation coefficient," https://en.wikipedia.org/wiki/Pearson_correlation_coefficient, 2018, accessed: 2018-07-27.

[41] J. R. Finkel, T. Grenager, and C. D. Manning, "Incorporating non-local information into information extraction systems by Gibbs sampling," in *Proceedings of the Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[42] "WHYPER tool," https://github.com/rahulpandita/Whyper, accessed: 2018-07-27.