Asking for (and about) Permissions Used by Android Apps

Ryan Stevens, Jonathan Ganz, Vladimir Filkov, Premkumar Devanbu, and Hao Chen
University of California, Davis
Davis, CA, USA
{rcstevens, jmganz, chen}@ucdavis.edu {devanbu, filkov}@cs.ucdavis.edu

***Abstract*—Security policies, which specify what applications are allowed to do, are notoriously difficult to specify correctly. Many applications were found to request over-liberal permissions. On mobile platforms, this might prevent a cautious user from installing an otherwise harmless application or, even worse, increase the attack surface in vulnerable applications. As a result of such difficulties, programmers frequently ask about them in on-line fora. Our goal is to gain some insight into both the misuse of permissions and the discussions of permissions in on-line fora. We analyze about 10,000 free apps from popular Android markets and found a significant *sub-linear* relationship between the popularity of a permission and the number of times when it is misused. We also study the relationship of permission use and the number of questions about the permission on StackOverflow. Finally, we study the effect of the *influence* of a permission (the functionality that it controls) and the *interference* of a permission (the number of other permissions that influence the same classes) on the occurrence of both permission misuse and permission discussions in StackOverflow.**

I. Introduction

Building and operating software securely is a major concern. A key aspect of achieving this goal is to set up appropriate *security policies*, which govern how applications are permitted to use the resources and services provided by the platform they operate on. For example, Java applets are typically prohibited from reading and writing files. The proper configuration of these security permissions provides a good operating point on the well-known trade-off between security and convenience. Correctly configured policies ensure that a) the applications are protected from each other, b) the platform is protected from the applications, c) the applications function correctly, and d) the users are not unduly inconvenienced. However, programmers often misuse these policy permissions, being either too conservative or too liberal in requesting permissions. Over-conservative security permissions can lead to inconvenient applications (*e.g.,* the user being frequently asked for various credentials) or even application failure (*e.g.,* when the application attempts an operation prohibited under the configured policy regime but necessary for its function). Over-liberal security permissions can be even worse: the application that is over-liberally permissioned can maliciously or erroneously cause harm at a scope larger than that what would be reasonably expected by users of the application. An increasingly common trend over the last decade or so is to specify these permissions in XML configuration files, which are distinct from the program source code. Separating these from the code provides design conveniences as well as operational conveniences. This is standard practice in container-based web service applications, for example on the .NET or J3EE platforms. The intent here is to change what applications are allowed to do at configuration time, and also allow security experts (who may not be same as the application developers) to configure the security permissions. Mobile platforms, notably Android, also use XML-configured security policies. However, the goal here is different: the permissions stated in the application security XML configuration files is essentially a *declaration* by applications of their intent to use certain platform-provided services; this type of intent declaration is called "provisioning". An application, for example, can ask for permission to use fine-grained geo-location, the network, or the device's microphone. When an application is installed, these provisions can be either checked interactively with the user (who may choose to deny them, which would abort the installation) or checked against some previously-specified defaults. If an application uses services beyond what is specified in the provisions, the application will throw an exception, usually resulting in application termination. On the other hand, if the application provisions for services beyond what it strictly needs, a user may unnecessarily deny installation; or, if s/he permits it, the application (because of error or malice) is capable of causing harm or inconvenience to the user.

Like other kinds of complex programming activity, security provisioning is fraught with human errors, and frequently a subject of discussion and concern in public forums, and in the media [11, 12, 7]. But how widespread is this problem? What are the factors that influence the occurrence of error in security permissioning? How much difficulty do programmers experience in learning about these permissions? These important questions are attractive ones for the software repository miner. Unfortunately, until very recently, it had been difficult to find large samples of applications that provision security permissions. Now, however, it is possible to find *thousands* of free applications in app marketplaces, such as Android marketplaces. These applications come each with their own security provisions. Furthermore, the use of high-level byte

978-1-4673-2936-1/13 © 2013 IEEE

code language allows the applications to be readily analyzed to determine the resources they actually use. Finally, it is now possible (thanks to StackOverflow) to obtain data relating to the demand for documentation and knowledge concerning permissions. With these observations in mind, we seek to answer the above questions.

In this paper, we make the following contributions.

1) We mine data from approximately 10,000 free applications from Android markets, and analyze which permissions are commonly used and which are commonly over provisioned. We relate the *use* of these permissions with discussions that *mention* these permissions in StackOverflow.
2) We examine the factors that influence permission *misuse*, including: the popularity of the permissions, the *influence* of the permissions (how much functionality they control) and the *interference* of the permission (how much the permission interacts with other permissions).
3) We examine the effects of the above factors on the demand for documentation about permissions, as measured by degree to which these permissions come up in on-line fora.

II. Background

This section briefly outlines important topics which we refer to in the paper. We also point out data sources available to developers when searching for information, notably the official Android documentation, and community help forums such as StackOverflow.

A. Android and Permissions

Android applications (commonly referred to as apps) are programmed in Java, and compiled into Dalvik bytecode; they are then packaged as a `.apk` file (APK), which is similar to Java's JAR file format. An APK consists of the app's class files, static assets such as image or video data, and a Manifest file, among other things. The Manifest is an XML file that specifies a number of properties about the app, such as what Android API level the app is targeted for, what screen (called an Activity) the app should start on, and what permissions the app requests. *Permissions* are the basis for the Android security model; they are granted to the app for its lifetime while installed on the device, with the exception of updates which may change the app's permissions. The appropriate permission must be requested in order for an app to access sensitive device functionality, such as the network or GPS manager. Typically, each permission controls some set of Android or Java API functions, and calling one of these privileged functions without the appropriate permission will cause a security exception (although note, it is not possible to bypass a permission simply by bypassing the API, for example through `exec` or native code). Developers who want more information about the API and permissions rely on the official Android documentation, which provides Android and Java API documentation, as well as tutorials and examples of Android development [1]. When a user installs an app, usually done through an Android app marketplace, all permissions requested by the app are presented to the user prior to installation, and the user is given an all-or-nothing choice of granting all the permissions and installing the app, or cancelling the install altogether. Thus, the more permissions an app requests, the more warnings users will be presented with, which may keep concerned users from buying or installing the app.

B. Overprivileged Apps

Considering that requesting too many permissions may decrease the popularity of an app, it is surprising that developers would request more permissions than they need; however previous work by Felt, et al. has shown that the problem of overpermissioning, when apps request more permissions than what they need, is prevalent among Android apps [6]. The authors found that one potential cause of overprivileged apps is unclear or incorrect documentation, where Android API documentation does not mention that a permission is required or the wrong permission is documented. Since permission errors result in exceptions during runtime, developers may feel compelled to include more permissions than they need so that users do not experience crashes after the app has been uploaded to a market. Additionally, the authors developed a tool called Stowaway that is able to detect when an app contains unnecessary permissions, which we used for this work. Stowaway provides a mapping of Android API functions to permissions that control that function, and can detect unnecessary permissions in an app by identifying all priviliged API calls in the app's bytecode and then mapping these back to the minimal set of permissions that the app requires. Any permissions in the app's Manifest that are not in the minimal set of permissions are then considered superfluous.

C. StackOverflow

StackOverflow is a software developement online forum where users can ask questions related to a variety of software development topics and receive answers from other users [19]. All questions asked in the open forums are publicly available for anyone to view, regardless if they are registered with the site. Questions are tagged with keywords that describe what topics the question relates to. Android is a popular topic on StackOverflow, being the fifth most used tag as of February 2013[1]. In addition to being a useful source of information for developers, StackOverflow provides researchers with insight into what topics are being discussed by developers.

III. Theory

Permissions are not misused equally. In our analysis, we found that some permissions are rarely misused, for example the WRITE_CONTACTS permission. On the other hand, some are almost always unnecessarily requested, such as the ACCESS_LOCATION_EXTRA_COMMANDS permission. Additionally, Felt et al. found that some permissions are more likely to be superfluous among overprivieged apps [6]. We would

[1] See `stackoverflow.com/tags`

like to gain insight into why developers use some permissions better than others using security-engineering related metrics capturing the popularity, complexity, and influence of different permissions. To that end, we model misuse as a function of these predictors.

A. Quantifying Permission Properties

One potential predictor that naturally arises when measuring premission misuse is the popularity of a permission, which we call *permission usage*. We measure use simply as the number of times the permission is requested in a manifest. We consider *permission misuse* as the number of instances a permission is requested by an app but not checked in any Android API call present in the app. We might expect that as a permission is used by more apps, it is also misused by more apps, and that the relationship between these variables is linear. However, it might also happen that developers do better at properly using permissions that are more popular, which would imply a sub-linear relationship between usage and overusage. This may arise in the case that popular permissions are better documented in the Android API or are discussed more often by the Android development community, as indicated by more demand for documentation.

Besides popularity, properties of the permission itself may affect how often it is misused by developers. The association between the permissions and their controlled API is a difficult security engineering problem. Groups of APIs that conceptually belong together, and often are used together, *should* be grouped together for developer convenience. On the other hand, if an API can at times be used by itself and is a critical API from user's point of view, then it would make sense to assign a permission specifically to that API. The trade-off between coarse- and fine-grained relationships between permissions and the APIs they control is a complex design problem. The Android platform includes both coarse- and fine-grained permissions; it also includes permissions that overlap with other permissions. In general, permission-API relationships can be modeled as a directed bi-partite graph, with edges between permissions and the resources they control.

The structure of this bi-partite graph might influence the ability of developers to understand them, and thus both the potential of permission abuse, and the demand for knowledge for them on fora like StackOverflow. For example, perhaps permissions that influence a small portion of the Android API are less likely to be misused, as apps which carelessly (*viz.,* virtually at random) request permissions are less likely to use an API function it controls; it also may be the case that the relationship between the permission and the controlled API are more coherent and easy to understand. Alternatively, the highly influential permissions may be more likely to be misused, as their precise purpose might be difficult for developers to divine. The *influence* of a permission is a measurement of how much of the Android API the permission controls. If the APIs are grouped under a permission very well (namely, all the APIs controlled by a permission are all used by an application, or none of them are used) then it is unlikely that the permission will ever be over-permissioned, that is, requested unnecessarily.

Lastly, our intuition as to why developers over-request permissions is based on the assumption that developers request permissions primarily to ensure their app executes properly. However, if many different permissions control the API they are interested in, the developers may be confused about precisely which permission is the right one to request, so they may have a tendency to *over*-provision. For example, a developer working with some specific feature in the Android API (like the GPS manager), may request all permissions that control any part of that feature. We call this intuition *interference* and expect that permissions with greater interference are more likely to be misused, as they may get "lumped in" with their interfering neighbors.

B. Research Questions

We have presented some intuitions concerning three potential mechanisms that might influence a developer's tendency to misuse a permission. With this in mind, we present our first research interest in this paper:

RQ1: *What is the effect of permission usage, influence, and interference on permission misuse?*

The precise metrics we use for measuring *influence* and *interference* are described in the following section.

In addition to developers misusing a permission more often when they do not understand it, they are also more likely to seek help regarding the permission. *Demand for documentation* captures how often developers are seeking documentation regarding the permission, which should be evident from Android developer community fora such as StackOverflow. Using this as our second dependent variable, we present our second research direction:

RQ2: *What is the effect of permission usage, influence, and interference on demand for documentation?*

Finally, we look at the relationship between permission misuse and the demand for documentation.

RQ3: *What is the a relationship between use (or misuse) and demand for documentation for the permissions?*

We both quantify this relationship statistically and illustrate it via a case study of example permissions which receive higher, medium, or lower levels of documentation demand for their corresponding levels of misuse.

IV. Dataset

In this section we describe our data sources and how we measured the various metrics for each permission presented in the previous section. We used two different data set for our work:

1) The authors of Stowaway have made their tool publicly available. We gathered a large collection (about 10,000) of apps from various Android markets. We ran these tools through stowaway, and determined a) how many times each permission was used (*ie.,* provisioned in the manifest) and b) how many times it was used, but the APIs it controls were not actually invoked in the application. Of the 130 permissions recognized by the Andriod documentation, we found 94 were actually used in the applications, after removing false positives that are not in Stowaway's mapping.
2) StackOverflow includes a large group of messages tagged with "Android". We used these for our analysis of the demand for information about different permissions.

A. Permission Properties

In order to determine properties of a permission like influence or interference, we need to know which API functions a permission controls. We chose to use Stowaway's publicly available permission mapping to do so [20]. This database provides a mapping from Android API functions to permissions that control the function. Using the mapping, we built a bipartite graph between permissions and API functions such that an edge represents that a function is controlled by a permission (similar to the graph in Figure 1, except the nodes on the right are *functions*, rather than *classes*). In this graph, the degree of a node from one part is a direct measure of it's relationships to nodes on the other part. We found the average function indegree to be 1.12, indicating that most API functions are controlled by exactly one permission (functions that are not controlled by any permission are not in the graph). We then measure the influence of a permission by its outdegree in the graph, which we will refer to by the variable $fout$.

To measure the interference of a permission, we created a similar bipartite graph, precisely of the kind shown in Fig 1, between permissions and API classes, such that an edge represents that a permission controls at least one function in a class. Figure 1 shows an example graph with three permissions and 4 classes. We define the variable $twohop$ as the number of permissions which can be reached from a permission in two hops in the graph. Figure 1 shows an example graph with three permission and explains their $twohop$ values. We believe this metric captures our notion of interference, where permissions are requested simply because they control a similar part of the API as other permissions.

B. Permission Misuse

Our application dataset consists of 10,300 free Android apps that request at least one permission, downloaded from various Android markets[2]. Using Stowaway, we determined what permissions each app requests and which of these permissions are superfluous for each app. We found that 44% (4,565) of the apps contained at least one unnecessary permission according to Stowaway, which is higher than the 33% reported in Felt, et al [6]. This discrepancy is likely due Stowaway's behavior, which will report any permissions not in its mapping as superfluous, including any misspelled or made-up permissions. When only considering permissions in Stowaway's mapping, the number of apps in our dataset with at least one unnecessary permission decreases to 39% (4,055). The remaining discrepancy can be explained by changes to the Android API which have made Stowaway's mapping stale, or by the lower quality apps on third-party Android markets (Felt et al. only considered apps from the official Android market). It's also entirely possible that permission misuse has become more prevalent since the original study. Using our results, we define two variables for each permission: $popularity$ is the number of times the permission is requested by an app in our dataset, $misused$ is the number of times the permission is superfluously requested by an app in our dataset. Note that $misused \leq popularity$ for all permissions; a permission cannot be misused, unless it's actually used.

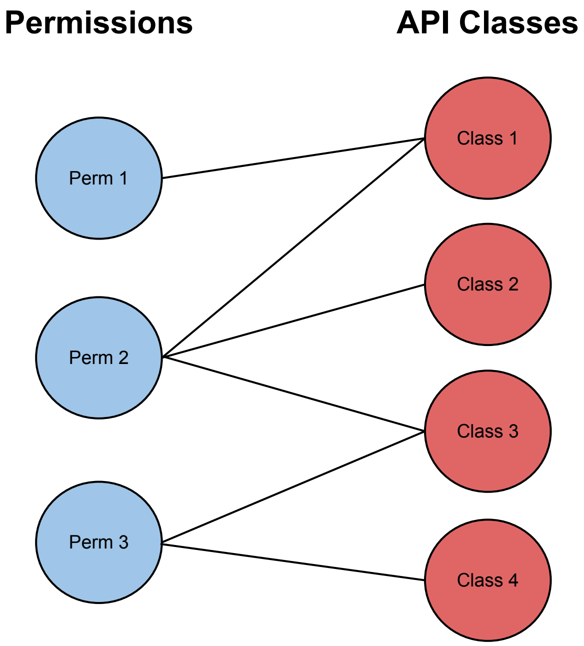


Fig. 1: Example bipartite graph between permissions and API classes. An edge indicates the permission controls at least one function in the class. The value of $twohop$ for permissions 1, 2, and 3 is 1, 2, and 1 respectively.

C. StackOverflow

Finally, we used StackOverflow to measure the demand for documentation. To do so, we used a publicly available database dump from StackOverflow from August 2012, which contains records of all questions and answers posted on StackOverflow [10]. We measured the demand for documentation for

[2] By market: Play (6, 978), SlideME (552), m360 (493), Brothersoft (464), androidonline.net (325), 1mobile (297), gfan.com (214), eoemarket.com (165), GoApk (97), Freeware Lovers (42), AndAppStore (44), softportal.com (37), androidsoft.org (12), Other (28)

each permission as the number of questions asked that contain the permission name in the post body or title. To avoid false positives, we ensured that the permission name was in all caps (to avoid confusing the INTERNET permission with the word “internet”, for example) and that all posts contain the keyword “android” in the title, body, or tags (case insensitive). Lastly, we found many posters will include their full Manifest file when asking an Android-related question. This is a potential confound. Permissions which occurred frequently would tend to frequently appear in questions, thus increasing the risk that we would see an artificial relationship between permission use in manifests and permission mention in the StackOverflow questions. So we removed full Manifest files from posts before searching for a permission name in the body.

However, if only a fragment of a of a Manifest is mentioned in a body, this indicates the poster’s belief that the part of the Manifest is related to the question in some way. The variable $questions$ represents the number of questions asked about a permission, such that the question body or title contains the permission’s name after the above data grooming has been done. Summary statistics of these variables is presented in Table I.

TABLE I: Summary of the variables which we measured from our datasets. N is the number of permissions which we included in our study.

Statistic	N	Mean	St. Dev.	Min	Max
$fout$	94	14.723	19.902	1	124
$twohop$	94	7.819	7.334	0	34
$popularity$	94	811.617	1,792.216	1	9,789
$misused$	94	84.064	115.006	1	547
$questions$	94	55.638	142.638	0	1,015

V. ANALYSIS

We present our analysis of the Stowaway and StackOverflow data based on the measurements presented in the previous section and our research questions presented in Section III. We first present a model for predicting permission misuse based on our variables, then we do the same for demand for documentation. Lastly, we examine the relationship between permission misuse and the demand for documentation.

A. Permission Misuse

We first build a linear regression model for predicting $misuse$ based on our independent variables $popularity$, $fout$, and $twohop$. We would expect there to be a strong relationship between $popularity$ and $misused$, regardless of our other variables, however the model only produces a modest fit for our data, with a Pearson correlation of 0.3. Looking at the effects of each independent variable on its own, we find that $popularity$ explains most of the variance, while the effects of our other variables are minimal. Interestingly, the Spearman rank coefficient between $popularity$ and $misused$ is 0.89. The significantly higher rank correlation between $popularity$ and $misused$ indicates the presence of a strong, but positive monotonic relationship; when $popularity$ increases, so does $misuse$; however, the smaller Pearson correlation indicates that the relationship is not linear.

We adjusted our model by taking the logarithm of $misused$ and $popularity$. We didn’t log scale the other variables. since they had much lower variances (indeed log-scaling them did not provide a significantly better model fit as indicated by the non-nested Cox test for linear models). This yields a model that fits our data much better, with an R^2 of 0.73. and a p-value < 0.01. A summary of this model is presented in Table II. We find that the effect of the influence measure $fout$ and the interference measure $twohop$ are insignificant. Thus, we conclude that our data *does not support the hypothesis that our measures of interference and influence affect the degree of misuse*.

TABLE II: Summary of our model of permission misuse.

	Dependent variable:
	$log(misused)$
Constant	0.916^{***}
	(0.210)
$log(popularity)$	0.579^{***}
	(0.040)
$fout$	-0.003
	(0.005)
$twohop$	-0.006
	(0.011)
Observations	94
R^2	0.729
Adjusted R^2	0.720
Residual Std. Error	0.790(df = 90)
F statistic	80.549^{***}(df = 3; 90)
Note:	*p<0.1; **p<0.05; ***p<0.01

Regardless, the strong non-linear relationship between $popularity$ and $misused$ warrants further investigation. Figure 2 presents a log-log plot of these two variables, which shows the relationship between $popularity$ and $misused$ is sub-linear (the log-log plot has a linear trend with slope, $s = 0.588$). This indicates that as a permission gets more popular, developers get better at using it in the correct context, and not over provisioning the permission. One factor that contributes to correct usage of security policies in general is a useful body of documentation. As the Android documentation does not describe correct usage of any specific permission, we speculate that the primary source of examples and use cases for each permission comes from community fora such as StackOverflow. If popular permissions were more thoroughly documented on such fora, we should be able to observe these trends by analyzing permissions’ demand for documentation.

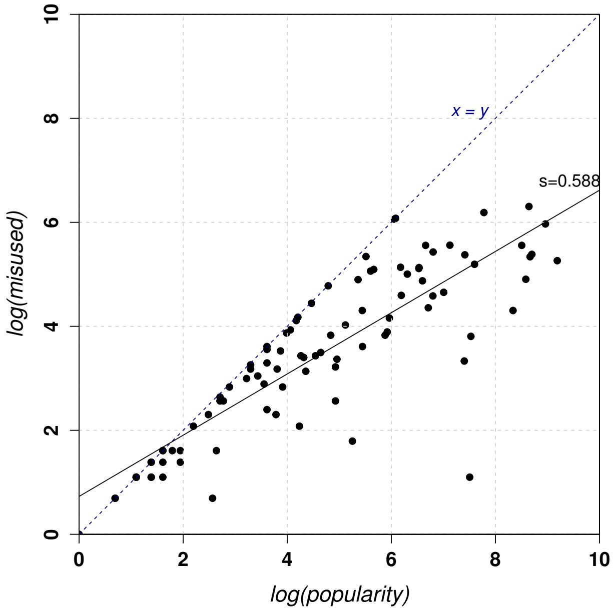


Fig. 2: Log-log plot of *popularity* versus *misuse* for each permission. The black line represents the best fit regression line for the points, while the dashed blue line is simply $x = y$, included for reference. The points that lie near $x = y$ are permissions which are almost always misused.

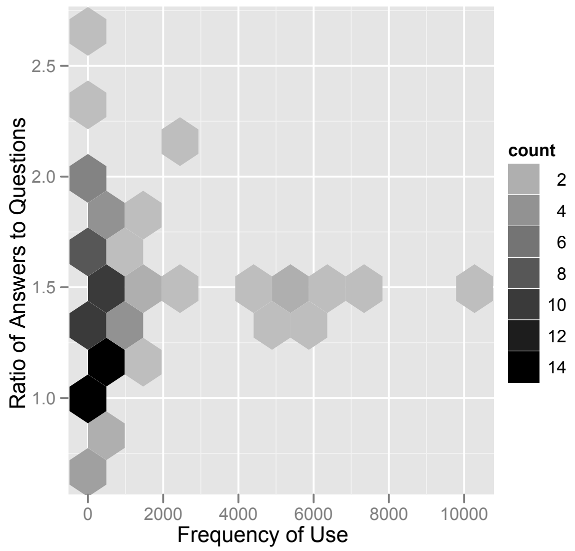


Fig. 3: Hexbin plot showing the variation of the answer/question ration with the frquency of use. While there is more variance in the number of answers per question for less popular permissions, questions about permissions used more than a few hundred times never go unanswered.

B. Demand for Documentation

There are around 5,300 questions (after data grooming to remove full Manifests, and non-capitalized permissions) that explicitly mention an Android permission. In this section, we consider the demand for documentation about permissions in this clearly important on-line forum, and examine the influence on this demand, of *popuarlity* and *misuse*, as well as the influence and interference measure we discussed earlier.

As before, we build a linear model that describes demand for documentation by using $questions$, the questions asked on StackOverflow, as our dependent variable and $popularity$, $fout$, and $twohop$ as our independent variables. We find our model fits our data reasonably well, with an R^2 of 0.58, however most of the variance is explained by $popularity$, with the other variables contributing very little. In order to account for nonlinear relationships, we build a model using the logarithm of $questions$ and $popularity$, while leaving the other variables unchanged, as we did with permission misuse. A summary is presented in Table III. This model fits our data better than the previous model (Cox's non-nested model comparison test indicates a much better fit after log-scaling). Interestingly, now we find the variables $fout$ and $twohop$ provide significant explanatory power, with $popularity$ still the most influential. Looking at the log-log plot of $popularity$ and $questions$ in Figure 4a, we see that their relationship is sub-linear, indicating that as a permission gets more popular, its ratio of questions to popularity goes down. This "leveling off" is likely due to etiquette on StackOverflow, where users are discouraged from asking questions that have been already adequately answered on the forum; the popular permissions have had the common questions asked about them already, and only get new questions when new or uncommon problems are experienced by developers. However, clearly, popular permissions *do* see a lot of questions that mention them on StackOverflow. We note here (See figure 3) that even extremely popular permissions which already have a large corpus of answers don't get neglected by the StackOverlow community. In fact, while some questions about less popular permissions get many answers, and others get none, the most popular permissions never get ignored, receiving an average of about 1.5 answers per question. This supports our previous claim that popular permissions are less likely to be misused because they have a more complete set of documentation from the community.

We now breifly look at the effects of influence and interference on demand for documentation. Since $fout$ is weakly positively correlated with $questions$, this implies that more influential permissions have more questions asked about them. Again considering that StackOverflow discourages repeat questions, we can see that influential permissions will apply in more contexts than uninfluential ones, and thus need more questions to saturate their demand for documentation. Strangely, $twohop$ is negatively correlated with $questions$, meaning permissions with greater interference have less demand for documentation, although the effect is weak. We speculate this is due to a type of suppressive dyadic effect that is common in network settings. Consider a particular permission p, controlling a class c, and another permission p' which is two hops from p. It may be that a pre-existing discussion about the influence of p' on c helps clarify a question that someone may have about the $p - c$ relationship. Thus, the more p's (two-hop connections) a particular permission p has, the more opportunity there is for this type of suppression. Further study is required to show this conclusively.

TABLE III: Summary of our model of demand for documentation.

	Dependent variable:
	$log(questions)$
$log(popularity)$	0.448^{***}
	(0.043)
$fout$	0.014^{***}
	(0.005)
$twohop$	-0.038^{***}
	(0.012)
Constant	0.805^{***}
	(0.225)
Observations	94
R^2	0.664
Adjusted R^2	0.653
Residual Std. Error	0.847(*df* = 90)
F statistic	59.277^{***}(*df* = 3; 90)
Note:	*p<0.1; **p<0.05; ***p<0.01

C. *Misuse and Documentation*

In the previous sections, we found as popularity increases, permissions become less and less likely to be misused. Our analysis of StackOverflow indicates that more widely used permissions get mentioned more often in questions, and furthermore, that these questions do get answered on StackOverflow. Next, we investigate the reason that some permissions are misused more heavily. To do so, we examine the relationship between permission misuse and demand for documentation. We observe that the Pearson correlation coefficient between $misuse$ and $questions$ is 0.43, while the Spearman rank is 0.72, indicating a non-linear relationship as in our previous cases. Figure 4b shows a log-log plot of $misused$ versus $questions$, where the relationship is once again sub-linear. The "levelling off" of questions is less severe than when observing popularity, indicating that developers' demand for documentation does not decrease with misuse as it does for popularity. This indicates that the demand for documentations for the misused permissions is not satisfied, or rather that the documentation for these permissions is still incomplete. We can see from these trends that StackOverflow provides measurable benefit to developers in terms of permission usage, and having many posts that reference permissions satisfies developers' demand for documentation regarding these permissions. In the next section, we observe the content of these posts in more detail to better understand how the permissions are documented.

VI. Case Studies

In this section, we present case studies of a few permissions that we manually analyzed in more detail. We first investigate the content of questions from StackOverflow to get an idea of how permissions are documented on the forum. Then, we present few permissions which have unusually high rates of misuse among our apps, and finally look at a permission with unusually low rates of misuse.

To get a better idea of how permissions are documented on StackOverflow, we manually analyzed few permissions in detail. The ACCESS_FINE_LOCATION permission is used to access the GPS coordinates of the device, and is one of the most popular permissions both in our app dataset, and on StackOverflow. We find that many of the posts that contain this permission name are not specifically asking about the permission, but about part of the API that is controlled by the permission. Often times, the question and top answer taken together form an example of how to use the API in question, and include the permissions needed to make the example work. Developers who rely on these examples to help them use the Android API will likely also use the suggested permissions, so it is important that the permissions they contain are correct and minimal. We manually verified a number of examples and found that they almost always contain the minimal set of permissions required to make the example execute properly. For example, Figure 5 shows a question where the poster is confused about why an API function is not working in their app, and mentions the permissions s/he includes. The top answer responds to the question, and points out that one of the permissions is actually unnecessary. We found similar results for other permissions as well, including the WAKE_LOCK permission, which is used for keeping the device from going to sleep, as well as for the less popular (and less asked about) CHANGE_WIFI_STATE permission, used for configuring the device's wireless LAN connection.

Two permissions, MOUNT_UNMOUNT_FILESYSTEMS and ACCESS_LOCATION_EXTRA_COMMANDS are almost always misused in our dataset ($>$ 99% of the time), despite being requested by 438 and 429 apps, respectively. The MOUNT_UNMOUNT_FILESYSTEMS allows developers to mount filesystems on the device, which could be used for managing the SD card, for example. This permission is likely a false positive from Stowaway, as developers may choose to execute Linux commands directly through `exec` instead of using the API, which Stowaway would not be able to detect as proper usage of the permission. The ACCESS_LOCATION_EXTRA_COMMANDS, on the other hand, is used for adding special behavior to Android's location manager, and developers would most likely not bypass the API as in the previous case, as the permission controls Android-specific behavior. One reason for this is interference from the more popular and similarly named ACCESS_COARSE_LOCATION and ACCESS_FINE_LOCATION permissions, which are only misused 3% of the time. If developers want to ensure that code that interfaces with the

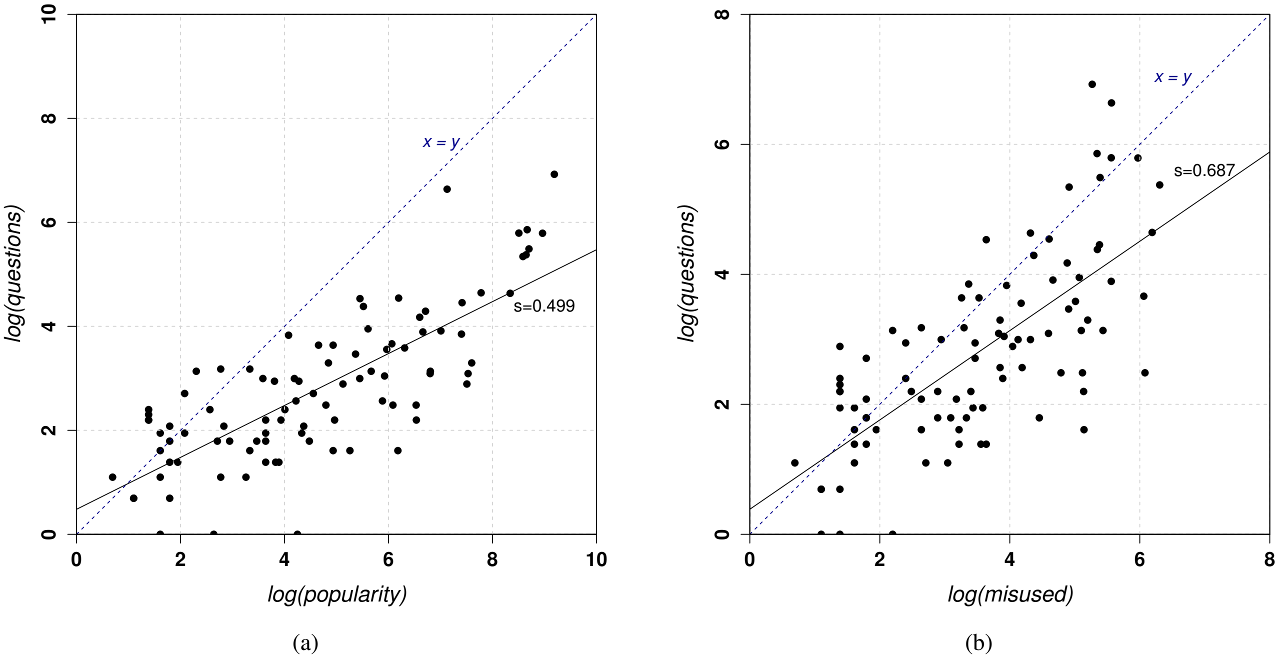


Fig. 4: Log-log plots of permission $popularity$ and $misuse$ versus $questions$ for the permission in our data set. The black line represents a best fit regression line for the points, while the dashed blue line represents $y = x$, included for reference.

location manager works properly, they may feel compelled to request all permissions that control some part of the location manager.

The DUMP permission, which is used for reading state information of system services, is the least often misused permission in our dataset, and is only overprovisioned 0.1% of the time despite being requested by $1,820$ apps. This case is interesting as DUMP is only granted to system applications or apps signed with the platform key. Developer apps will still run if they request this permission, however they will not actually be granted the permission and will crash if they attempt to use a privileged function that requires it. Since it is rarely misused, developers must also have code in their app that executes these privileged functions. It may be that this is dead code, that the security exceptions are properly handled, or that these apps are buggy and prone to crash. However, a more likely explanation is that these apps are designed to run on rooted devices (where the user has unlocked the root user account), allowing them to hold system permissions. Although the official Android market does not allow such apps, many of our apps are from third-party markets where apps that require root access are allowed. One would expect that if a developer were to include a system-level permission that requires their app run as root, they would be sure to use it, otherwise they are limiting their target audience for no reason. However, two other system level permissions which we observe in our dataset, INSTALL_LOCATION_PROVIDER and ACCOUNT_MANAGER have very high rates of misuse. Further study can shed more light on this trend.

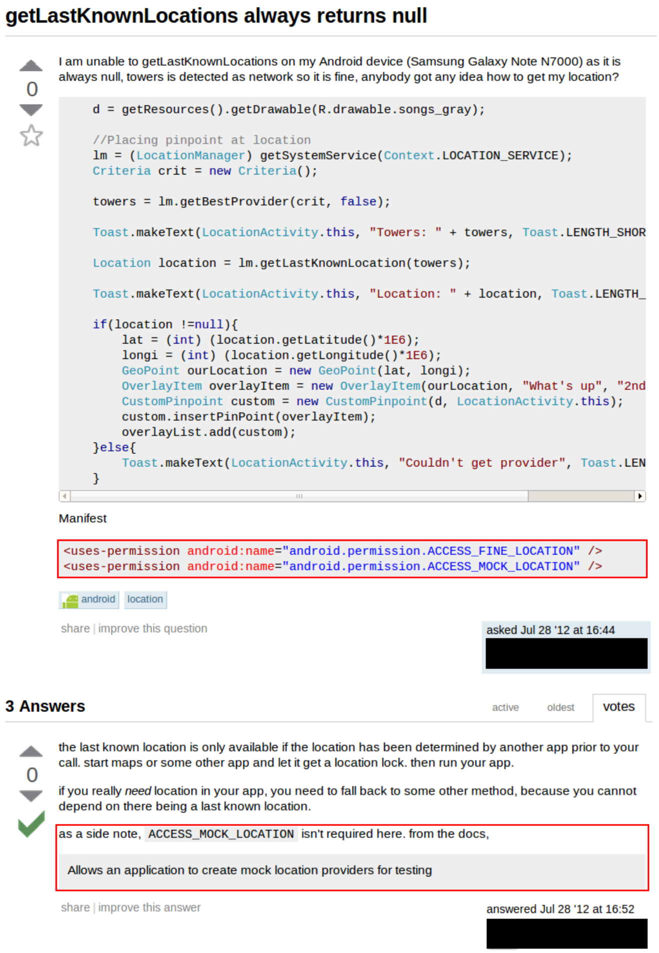
We have presented a number of statistically significant trends in our dataset and attempted to gain better insight into these trends through a few case studies. We would now like to point out potential threats to validity that may have affected our analysis, followed by related work, and finally our conclusions.

VII. Threats To Validity

A. Stowaway

Stowaway consists of two distinct parts: a dynamic analysis tool that builds the permission mapping, and a static analysis tool that uses the mapping to find overprivileged Android apps. Although the dynamic analysis tool is very accurate, the static analysis tool suffers from shortcomings when developers request a permission but use it outside the Android API, for example through `exec` or native code. We were able to identify a few permissions for which this behavior produced a significant number of false positives (an example presented in Section VI). However, these permissions constitute a small fraction of all permissions we observed.

Perhaps a more serious threat is that Stowaway's permission mapping was derived from Android 2.2 (API version 8), which was released in May 2010. Although most of the apps in our dataset were downloaded more recently, by analyzing their Manifest files we determined that 94% (9,632 apps) of them have a `minSdkVersion` at or below 8. This means that these apps are designed to be backwards compatible with Android 2.2, and our Stowaway results should not be heavily impacted by changes to the API since version 8. Additionally, we can see from [22] that developer permissions (that is, "dangerous" and "normal") are not added or removed often, and these permissions constituted the bulk of our analysis.



getLastKnownLocations always returns null

I am unable to getLastKnownLocations on my Android device (Samsung Galaxy Note N7000) as it is always null, towers is detected as network so it is fine, anybody got any idea how to get my location?

```
    d = getResources().getDrawable(R.drawable.songs_gray);

    //Placing pinpoint at location
    lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    Criteria crit = new Criteria();

    towers = lm.getBestProvider(crit, false);

    Toast.makeText(LocationActivity.this, "Towers: " + towers, Toast.LENGTH_SHOR

    Location location = lm.getLastKnownLocation(towers);

    Toast.makeText(LocationActivity.this, "Location: " + location, Toast.LENGTH_

    if(location !=null){
        lat = (int) (location.getLatitude()*1E6);
        longi = (int) (location.getLongitude()*1E6);
        GeoPoint ourLocation = new GeoPoint(lat, longi);
        OverlayItem overlayItem = new OverlayItem(ourLocation, "What's up", "2nd
        CustomPinpoint custom = new CustomPinpoint(d, LocationActivity.this);
        custom.insertPinPoint(overlayItem);
        overlayList.add(custom);
    }else{
        Toast.makeText(LocationActivity.this, "Couldn't get provider", Toast.LEN
    }
```

Manifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
```

asked Jul 28 '12 at 16:44

[redacted]

3 Answers

the last known location is only available if the location has been determined by another app prior to your call. start maps or some other app and let it get a location lock. then run your app.

if you really *need* location in your app, you need to fall back to some other method, because you cannot depend on there being a last known location.

as a side note, `ACCESS_MOCK_LOCATION` isn't required here. from the docs,

> Allows an application to create mock location providers for testing

answered Jul 28 '12 at 16:52

[redacted]

Fig. 5: Example StackOverflow question where the poster is confused about the getLastKnownLocations API function. The poster includes the permissions they think they need (the first red rectangle), and the top answer mentions that one of them is unnecessary (the second red rectangle).

B. *StackOverflow*

We present two potential threats to validity in our analysis of the StackOverflow data. First, the StackOverflow dataset represents a snapshot of the site from August 2012, but as previously mentioned, Stowaway's mapping is for a version of Android released May 2010. A simple way to alleviate the mismatch between the datasets would be to truncate the StackOverflow dataset and only consider posts before a certain date. However, there is a delay between when a new version of Android is released, and when it is adopted by the general public (as of March 2013, [17] shows only 14% of users are using the most recent version of Android). Thus, we expect developers to ask questions about older API versions, even as new ones come out. Another approach would be to infer what API version a post is asking about, but the vast majority of posts contain no information about what version the developer is targeting. Additionally, in manually analyzing posts we observed many are applicable to multiple versions of Android and thus any attempt to infer the target API version would likely muddy our results more than it would help. Regardless, we acknowledge there may be noise in the posts we selected from StackOverflow as a result of developers asking questions that are specific to an Android version other than 2.2.

The second threat to validy is regarding the precision and recall of StackOverflow posts. When selecting relevant posts from the StackOverflow corpus, we chose to be conservative in the posts we selected as relevant (see Section IV-C). Although manually analyzing the results indicates that the selected posts are related to the permission, it is hard to state a concrete definition of a post representing demand for documentation regarding a permission. Recall is even more difficult to empirically measure, although we can speculate that most of our false negatives would occur from posters asking about API functions without knowing they are controlled by a permission. Assuming this is not more likely to occur with one permission over another, this should only reduce the number of posts we selected, not skew them towards selecting one permission over another. Thus, in our selection of posts we attempt to be precise over being complete, but we acknowledge that we may have false positives and false negatives in the selected posts.

VIII. Related Work

Alternatives to Stowaway include Andrubis, an extension of the Windows malware analysis tool Anubis, which is able to detect superfluous permissions in addition to malicious apps [2], and AndroidLeaks which is a static analysis tool for Android that is able to build permission mapping by finding permission checks in the Android API [8]. This differs from Stowaway which uses dynamic analysis to build its mapping. Vidas et al. [21] built a permission checking tool as a plugin for Eclipse, which notifies developers when a permission is not used in their app (similar to how Eclipse notifies developers about unused imported libraries).

Previous work has looked at privilege escalation in Android through inter-process communication (IPC) between applications. One such attack is called the confused deputy attack, where a benign-but-priviliged app mistakenly leaks sensitive data through IPC to less priviliged apps. Similarly, two malicious apps may collude to share sensitive data over IPC. Quire, an extension to the Android framework, is able to detect such attacks by keeping track of the origin of all data sent over IPC [5]. AppFence is another Android modification which is able to protect users from permission-hungry applications by blocking transmissions of data which the user does not want to leave the device, and can replace such data with shadow data [4]. Additionally, Grace et al. [9] found that many stock Android smartphones come with apps installed that expose sensitive device functionality through IPC, allowing apps to bypass certain permissions.

Prior work identifying APIs on StackOverflow includes Parnin et al. [16] who measured the coverage of the Android and Java APIs on StackOverflow, and Parnin and Truede [15], who measured the coverage of jQuery's API on various development sites, including StackOverflow. Barua et al. characterize popular topics on StackOverflow via natural language processing [3]. Additionally, previous work has been done identifying experts in community forums, such as StackOverflow [14, 13, 18].

IX. Conclusion

We have gathered apps from Android markets and connected them to questions about security permissions use on StackOverflow. We presented statistical models for predicting permission misuse and demand for permission documentation. We found that the popularity of a permission is strongly associated with its misuse, while other factors such as influence and interference had little effect. Our analysis of the StackOverflow data indicates that more widely used permissions get mentioned more often in questions, and furthermore, that these questions do get answered on StackOverflow. These findings suggest one reason for the decreasing likelihood of misuse with increasing popularity: this is a result of more complete documentation being available to more popular permissions. This indicates that developers do better at properly using a permission as it is discussed on community fora such as StackOverflow.

Acknowledgements

This paper is based upon work supported by the National Science Foundation under Grant No. 1018964.

References

[1] *Android Developer Documentation*. URL: http://developer.android.com/.

[2] *Anubis: Analyzing Unknown Binaries*. Jan. 2013. URL: http://anubis.iseclab.org/.

[3] Anton Barua, Stephen W. Thomas, and Ahmed E. Hassan. "What are developers talking about? An analysis of topics and trends in Stack Overflow". In: *Empirical Software Engineering* (2012), To.

[4] Sven Bugiel et al. "Towards taming privilege-escalation attacks on Android". In: *Proceedings of the 19th Annual Symposium on Network and Distributed System Security*. 2012.

[5] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S Wallach. "Quire: Lightweight provenance for smart phone operating systems". In: *20th USENIX Security Symposium*. 2011.

[6] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. "Android permissions demystified". In: *Proceedings of the 18th ACM conference on Computer and communications security*. ACM. 2011, pp. 627–638.

[7] Sean Gallagher. *Researchers find privacy and security holes in Android apps with ads*. Mar. 2012. URL: http://arstechnica.com/business/2012/03/researchers-find-privacy-and-security-holes-in-android-apps-with-ads/.

[8] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. "AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale". In: *Trust and Trustworthy Computing* (2012), pp. 291–307.

[9] Michael Grace, Yajin Zhou, Zhi Wang, and Xuxian Jiang. "Systematic detection of capability leaks in stock Android smartphones". In: *Proceedings of the 19th Annual Symposium on Network and Distributed System Security*. 2012.

[10] Chris Hewgill. *Stack Overflow Creative Commons Data Dump*. June 2009. URL: http://blog.stackoverflow.com/2009/06/stack-overflow-creative-commons-data-dump/.

[11] Rob Lightner. *How to check your Android device for rogue apps*. June 2011. URL: http://howto.cnet.com/8301-11310_39-20073434-285/how-to-check-your-android-device-for-rogue-apps/.

[12] Josh Lowensohn. *Photo theft security loophole found in Android too*. Mar. 2012. URL: http://news.cnet.com/8301-1035_3-57388797-94/photo-theft-security-loophole-found-in-android-too/.

[13] Aditya Pal, Rosta Farzan, Joseph A. Konstan, and Robert E. Kraut. "Early Detection of Potential Experts in Question Answering Communities". In: *UMAP*. 2011, pp. 231–242.

[14] Aditya Pal, F. Maxwell Harper, and Joseph A. Konstan. "Exploring Question Selection Bias to Identify Experts and Potential Experts in Community Question Answering". In: *ACM Trans. Inf. Syst.* 30.2 (2012), p. 10.

[15] Chris Parnin and Christoph Treude. "Measuring api documentation on the web". In: *Proceedings of the 2nd international workshop on Web*. Vol. 2. 2011, pp. 25–30.

[16] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow". In: *Georgia Institute of Technology, Tech. Rep* (2012).

[17] *Platform Versions*. Mar. 2013. URL: http://developer.android.com/about/dashboards/index.html.

[18] Daryl Posnett, Eric Warburg, Premkumar Devanbu, and Vladimir Filkov. "Mining Stack Exchange: Expertise is Evident From Initial Contributions". In: *2012 ASE International Conference on Social Informatics*. 2012.

[19] *StackOverflow*. URL: http://stackoverflow.com/.

[20] *Stowaway*. URL: http://android-permissions.org.

[21] Timothy Vidas, Nicolas Christin, and L Cranor. "Curbing android permission creep". In: *Proceedings of the Web*. Vol. 2. 2011.

[22] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. "Permission Evolution in the Android Ecosystem". In: (2012).