A Multi-dimensional Measure for Intrusion - the Intrusiveness Quality Attribute

Ashish Agrawal
Department of Computer Science & Eng.
Indian Institute of Technology Kanpur
Kanpur, India-208016
agrawala@cse.iitk.ac.in

Balwinder Sodhi
Department of Computer Science & Eng.
Indian Institute of Technology Ropar
Ropar, India-140001
sodhi@iitrpr.ac.in

Prabhakar TV
Department of Computer Science & Eng.
Indian Institute of Technology Kanpur
Kanpur, India-208016
tvp@cse.iitk.ac.in

ABSTRACT

Security in personal devices like mobile phones, tablets, is a major concern because these devices often carry sensitive information. Device platforms (e.g. Android) implement "limit access" and "authorize" security tactics to protect privacy/security-sensitive resources against misuse by an app. For instance, Android defines a set of 100+ permissions that guard resources such as phonebook data, network sockets and so on. However, due to poor understanding of these complex permissions, users inadvertently grant dangerous permissions to the apps, which defeat the security tactics implemented.

Thus, security of a device is directly related to the capabilities granted to the intruder (app in this case). In this paper, we define a new quality attribute (QA) called Intrusiveness of an app, which characterizes the capabilities of an app to cause violation of personal and operational information of the user/device. We suggest a framework to compute "intrusiveness" on a given platform. Intrusiveness of an app is represented as a 4-tuple. This tuple characterizes the extent to which the permissions, that are being sought by an app, could compromise in 4 dimensions of information, viz. User, Device, Carrier and the External World. It helps the user to realize the nature of privacy-sensitive resources that (s)he is exposing to the app. Efficacy of our framework is demonstrated by examining intrusiveness of 814 most popular free apps on Android. The Intrusiveness QA could be used to compute potential violation of User Personal Privacy, User Locational Privacy and violation of Device Integrity. Our analysis shows that 84% of apps examined are in a position to compromise User Personal Privacy, 96% can comprise Device Integrity and 92% can compromise Locational Privacy.

QoSA'13, June 17–21, 2013, Vancouver, BC, Canada.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Representation*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Design, Measurement

Keywords

Android, Intrusiveness, Privacy

1. INTRODUCTION

An app requires access to certain resources in order to fulfill its functional requirements. These resources often have different privacy or security sensitivity. For example, the phonebook data on a user's smartphone is a sensitive resource, whereas list of openly accessible Wi-Fi networks that are visible to this smartphone may not be considered as sensitive. Typically, a smartphone application platform (e.g. Android) implements some access control mechanism via which apps are granted permission to access certain privileged/sensitive resources which the app developer has indicated as required for functioning of such an app.

There is always a possibility – intentional or accidental – of an app trying to abuse the permissions that have been granted to it. Given the nature of the technical complexity involved, an end user cannot easily figure out whether an app is abusing the permissions or not. Failure in proper evaluation of app's behaviour by the user can be caused by:

1. **Poor understanding of permissions**: A user may not be able to comprehend the permissions correctly. In their research study of Android platform, Felt et al. [9] found that most of the users (97%) were not able to understand the Android permissions correctly. This may lead the user to unknowingly granting sensitive permissions to the app.

2. **Grayware apps**: An app can genuinely require sensitive permissions to provide some features to the user. However, behind the scene, the app may be exploiting these permissions to compromise user privacy or integrity of device. Figuring out such malicious intentions of an app requires analysis of it's code and

run-time behavior. Tools like Stowaway [7] and MobileScope [10] can help user in analysis of compiled code and run-time behaviour of an app.

To identify such behaviour of an app, we define a new quality attribute called Intrusiveness, which characterizes the extent of violation happening into the personal and operational space of the user.

An app may be intentionally intrusive into the user space by exploiting the permissions granted to it. Similarly, at design time, an app developer may not comprehend the implication (in terms of intrusiveness) of using certain platform features/resources in the app so that he/she ends up requiring certain sensitive permissions where they may not really be needed, thus making the app inadvertently more intrusive. Informed users view intrusive apps with caution/suspicion, thus many users may not favor an app with high intrusiveness. Given this scenario, it is desirable even for a developer to comprehend the intrusiveness of an app.

We provide a framework to investigate intrusiveness of apps on a given platform. Our framework helps in examining the capabilities of an app to intrude into different spaces (personal and operational) of a user. The framework is based on analysis of resource permissions of the platform. Output of the framework is an Intrusiveness Score (IS) of an app, represented as 4-tuple (U, D, C, W), to describe intrusiveness in four dimensions, User Space, Device Space, Carrier Space and External World Space. This score can be used to calculate potential violation of user personal privacy, user locational privacy and integrity of the device.

Rest of the paper is structured as follows: Section 2 describes Android Permission Ecosystem, user understanding of permissions and related literature in this space. Section 3 describes framework for investigating intrusiveness of an app. Results and analysis of our experiment on 814 Android apps is explained in Section 4. Section 5 suggests some utilities of our framework. Section 6 conclude the paper with scope of future work.

2. BACKGROUND AND RELATED WORK

2.1 Android Permissions Ecosystem

In Android platform, access to several features by an app is restricted by permissions from user. As of now, the platform specifies 130 permissions [3] related to different APIs. Before responding to API method calls from an app, the Android platform process validates whether the application has necessary permissions for invoking these methods or not. Each app needs to specify the permissions required by it which are shown to the user at the time of app installation. Thus, a user can also cancel the installation if (s)he thinks that the permissions required by the app are not appropriate.

Each Android permission has a protection level associated with it. This protection level can have four values: Normal, Dangerous, Signature, and SignatureorSystem [1]. Dangerous permissions are the one, which can compromise user's privacy, security or device performance. Signature and SignatureorSystem are the permissions which are granted only to the applications with a matching manufactured identifier as of the platform. Thus, these are not generally used by third party applications.

Android platform also categorizes permissions in 30 different groups [2] e.g. ACCOUNT, COST MONEY et cetera. Permissions related to a specific feature are put in a specific category, e.g. permissions related to ACCOUNT features are put inside ACCOUNT category. However, many of the permissions do not have a permission group associated with them.

2.2 User Understanding of Permissions

As the user is asked for permissions at the time of app installation, all the permissions required by an app are displayed to her. However, a user may not be able to interpret the description of permissions correctly, which can lead to giving dangerous permissions to the app. Research study done by Felt et al. [9] shows that only 3% of users in their survey were able to comprehend the permissions correctly. This motivates for developing a better system, which can help a user in interpreting the permissions correctly.

Another direction to help a user is to flag the app which requires permissions with dangerous protection level [1]. This can notify the user about app's intrusiveness. However, previous research studies showed that most of the applications (93% of free apps [8]) require at least one dangerous permission. This shows that flagging the app with a dangerous permission won't help much as most of the applications will end up getting flagged as dangerous.

As the number of apps with at least one dangerous permission is very high, it is clear that generally users are accustomed to use apps with dangerous permissions. To help the user in understanding and highlighting intrusiveness of apps, we need finer granularity of sensitivity even inside dangerous permissions.

2.3 Related Work

Several researchers have emphasized the role of Android permissions in user privacy and security. Tools like Stowaway [7] helps in analyzing compiled code of an app, to investigate whether an app is over-privileged or not. Felt et al. [7] found that one third of apps they examined, are over-privileged, i.e. they are asking for more permissions even they are not using them in the code. Major reasons behind such over-privileged nature are lack of proper permission documentation and developer mistakes.

Chia et al. [5] analyzed more than 2000 Android application to figure out relation between app popularity and number of permissions asked. They showed the top 12 most dangerous permission asked by the apps and found the trend that most popular apps are asking for more permission than less popular. Barrera et al. [4] analyzed the granularity of Android permissions using Self-Organizing Map algorithm. They found that the Android app categories are loosely defined and assigned on the basis of semantic classes, not on the basis of features provided by them. They suggested some enhancement to the Android permissions granularity (e.g. INTERNET is too wide) for improving security.

Xuetao et al. [11] studied the evolution of Android permission ecosystem and found that not only the total number of permissions are increasing with time, permissions in dangerous category are also growing, thus, violating the principle of least privileged.

In the analysis done by Felt et al. [8], they found that 93% of free and 82% of paid apps have at least one dangerous permission. Also some apps have both permissions, for

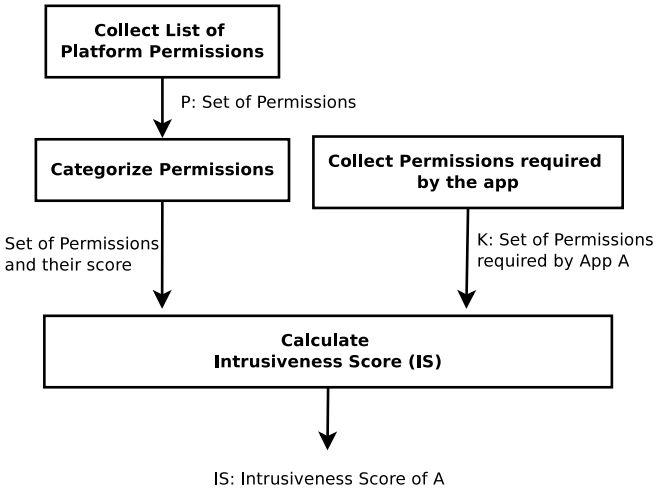

Figure 1: Methodology for Investigating Intrusiveness

reading personal information and for internet connectivity, which can compromise user privacy.

Research survey done by Felt et al. [9] on effectiveness of Android permissions shows that most of the users (83%) do not pay attention to the permissions at all. Weak comprehension of these permissions (only 3% were able to understand correctly) can be the major reason behind this. Their research shows that the current Android permission system is not able to help user in taking correct security decisions.

From existing literature, it is clear that most of the Android apps seek dangerous permissions, and there is no tool which can distinguish whether an app needs those dangerous permissions for legitimate use or for malicious features. Better understanding of implications of granting permissions, can help user in order to handle such situations. Our work can be seen as a next step of work done by Felt et al. [9], which will facilitate users in understanding these implications at a finer granularity of sensitivity.

3. FRAMEWORK FOR INVESTIGATING INTRUSIVENESS

We provide a framework for investigating intrusiveness of apps on a given platform. Basis of our framework is to analyze the permissions e.g. type of information they manipulate, type of access on information, et cetera. The framework can be used for investigating intrusiveness on various platforms e.g. Android, IOS, Chrome, et cetera. Figure 1 depicts the methodology of our framework. Steps of the framework are explained in following subsections.

3.1 Collect List of Platform Permissions

To investigate the intrusiveness, first step is to collect list of all available resource permissions of the candidate platform. This can be generated using platform documentation, if available. In other case, platform libraries can be analyzed to generate this list.

3.2 Categorize Permissions

To help the user in better understanding of permissions, we need to categorize the permissions in such a way that

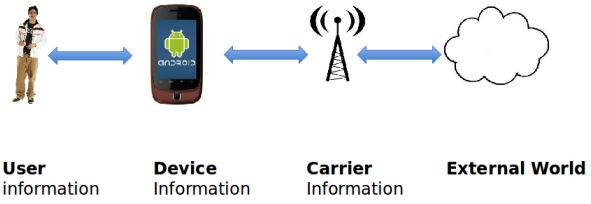

Figure 2: Information Categorization

it highlights the space in which the app is intruding. Our perspective of permissions is that each permission allows an app to either read or write some information. We even consider the inspection of device features and settings by an app as generating/manipulating information. We categorized all such information, that an app can access, based on what different entities that such information belongs to. The four categories that we defined for classifying such information are:

- **User Space (U)**: This space contains all information related to the actual user of a device. For example, profile information, account informations, text messages, call logs, et cetera. Compromising these informations will directly affect user personal privacy.
- **Device Space (D)**: This space contains all information related to the device. For example, running processes, cache, frame-buffer data, status bar, et cetera. Compromising this space will affect integrity of device.
- **Carrier Space (C)**: All information related to carrier of device is categorized into this space. For example, available networks, connection status, protocols, location, et cetera. Compromising this information will affect user location privacy.
- **External World Space (W)**: All information related to communication with external world is categorized into this space. For example, opening an arbitrary network socket for communicating with some processes listening on a URL somewhere.

All permissions are examined and assigned a score on the basis of the space in which they are manipulating information. Each permission will have a score in each space, which is one of read/write/none, depending upon whether they are reading or writing or doing nothing in that space. Thus, a permission can have different scores in different spaces. For example, permission to "access contacts and profile information" allows an app to read personal information of user, thus will have a read score in user space. However, this permission does not protect any information related to other spaces apart from user space. So score in other spaces for this permission will be none. Similarly, permission to "modify browser's history" allows an app to change information related to user, and will have a write score in user space. Figure 2 depicts different entities/stakeholders in our ecosystem.

For quantitative analysis that we discuss in following steps, scores values (i.e. for read/write/none) in each of the above mentioned four spaces needs to be mapped to numerical values. Such a mapping can be decided depending upon the

intended application of Intrusiveness Score. For example, if the user does not want to make any difference between read or write, both "read" and "write" can map to the numerical value 1 and "none" can map to 0.

Let us say that P is the set of all permissions available for a device platform DP. By using the above mentioned score mappings, each permission $p \in P$ will have a Intrusiveness Score $IS_p = (U_p, D_p, C_p, W_p)$, where U_p, D_p, C_p, W_p are score value in User, Device, Carrier, and External Space respectively.

3.3 Collect Permissions required by the App

In order to examine intrusiveness of an app, we need to find out all permissions sought by the app. In platforms like Android, where an app needs to specify all required permissions at the time of installation, collecting such information is easier – one can simply examine the app's metadata to figure out what permissions are sought. However, in platforms where the permissions are granted during run-time (iOS 6), one needs to scan the app's code to figure out permissions required by it. Tools like PiOS [6] can be used in such scenarios.

3.4 Calculate Intrusiveness Score (IS) of an App

In this step, a single Intrusiveness Score (IS) is generated for the app. Let us say, $K \subset P$ is the set of permissions required by an app A. Each permission $k \in K$ has a score in each space (U_k, D_k, C_k, W_k) representing whether they are reading/writing or doing nothing in that space.

Various models can be possible for calculating Intrusiveness Score value from individual category-wise scores of sought permissions. Total number of permissions sought by an app can also be a parameter in the model, which represent the degree by which the app is operating on protected information. Summation of category-wise scores of sought permissions, can represent the number of ways an app can intrude into each category. Selection of functions and parameters for the model depends upon the aim for analysis of results.

In our experiment, our aim is to investigate whether an app can violate user privacy and/or device integrity. For such analysis, we chose the MAX function because it gives a conservative IS estimate, that is, an intrusive app will never be flagged as non-intrusive. We calculate Intrusiveness Score for an app A as:

$$IS_A = (U, D, C, W)$$

where $U = Max(U_k : \forall k \in K)$, $D = Max(D_k : \forall k \in K)$, $C = Max(C_k : \forall k \in K)$, and $W = Max(W_k : \forall k \in K)$

4. EXPERIMENTS

For our experiment, we considered top 814 free Android applications which are available on Google Playstore under all the categories defined there. As we are considering only free applications, our results may be biased. However, this does not affect the utility of our framework.

Research study done by Felt et al. [7] shows that Android permission documentation is poor and contains 15 unused permissions as well as 6 errors. For our experiments, we generated a set of 116 permissions by examining the metadata specified by the 814 applications that we considered. Using the methodology specified in Section 3, we analyzed these 116 permission on the basis of information they are manipulating, and assigned each of them a score in four categories: User, Device, Carrier, and External World. Table 1 represents some sample permissions and score assigned to them. For quantitative analysis, we assigned numerical values to these score as: read => 1 and write => 5. By assigning higher numerical value to write permission, we assume that write permission is more intrusive than read permission. Overall intrusiveness of an app depends on the total permissions that are asked by the app. For instance, an app that requires just the read permissions for some sensitive information of the user (e.g. phone book) in itself may not be dangerous unless the app also requests for a write permissions for network sockets via which the app may be able to send this sensitive user information to some external entity outside of this device. So in that sense, there must be at least one "write" permission sought by the app to be able to cause any kind of violation. Therefore, we assign higher score to a write permission.

Using this permission scoring method, we generated a cumulative score for each app, on the basis of permissions asked by it. Such overall category-wise score is calculated using maximum score for each category from the score of permissions in that category. For example, if an app sought 4 permissions viz. P1 to P4, and each of P1 to P4 had scores 1, 1, 1, 5 respectively for the category User Space, then the overall score for this app for category User Space is MAX(1,1,1,5) i.e. 5. Table 2 represents Intrusiveness Score for some sample apps.

Table 1 : Some sample permissions and their scores

Permission Description	Score			
	U	D	C	W
Allows apps to access the contacts and profile information of account(s) stored on this Android device	R	-	-	-
Allows the app to modify the Browser's history or bookmarks stored on your tablet	W	-	-	-
Allows the app to retrieve information about currently and recently running tasks	-	R	-	-
Allows the app to set the system wallpaper	-	W	-	-
Allows the app to get your precise location using the Global Positioning System (GPS) or network location sources such as mobile towers and Wi-Fi	-	-	R	-
Allows the app to connect the tablet to and disconnect the tablet from WiMAX networks	-	-	W	-
Allows the app to receive and process SMS messages	W	-	-	R
Allows the app to create network sockets and use custom network protocols	-	W	-	W

Table 2: Intrusiveness Score of some sample apps

App Name	App Domain	Intrusiveness Score (IS)			
		U	D	C	W
Pencil Sketch	PHOTOGRAPHY	5	5	1	5
Funny Facts	COMICS	5	5	1	5
Tabla	MUSIC_AND_AUDIO	0	5	1	5
Official eBay Android App	SHOPPING	5	5	1	5
Spider Solitaire	CARDS	5	5	1	5
Windows 7 Go Launcher EX Theme	PERSONALIZATION	0	5	0	0

4.1 Results and Analysis

Our experiment on examination of 814 apps resulted in 116 permissions, score of each permission, and intrusiveness

score for all examined apps. Some points we found after analysis of our results are:

Table 3 : Top 5 permissions sought by all examined apps

Permission Description	Scores				Number of Apps
	U	D	C	W	
Allows the app to create network sockets and use custom network protocols	0	5	0	5	778
Allows the app to view information about network connections such as which networks exist and are connected	0	0	1	0	733
Allows the app to write to the USB storage	5	5	0	0	551
Allows the app to test a permission for USB storage that will be available on future devices	0	1	0	0	550
Allows the app to access the phone features of the device	1	0	0	0	504

Table 4 : Number of permissions in each category out of total 116 permissions of all examined apps

Permission Categories	Number of Permissions	
	Read Access	Write Access
User	22	38
Device	6	43
Carrier	7	6
External World	5	9

1. Table 3 shows the top five permissions sought by the apps. Among them, Internet is at the top as sought by around 96% of examined apps. Thus, majority of apps ask for permission to write into external world. Two of the top five permissions are to write in Personal, Device or the Outside World.
2. Table 4 depicts the number of permissions reading or writing in each space. This shows that around 52% of permissions are protecting information in user space. Around 42%, 11%, and 12% of permissions are protecting information in device space, carrier space, and external world space respectively.
3. Table 5 depicts number of apps asking for read/write/none permission in the spaces. It shows that a lot of the examined apps are asking for write permissions (75% in user space, 96% in device space, and 96% in external world space).
4. We found that even apps with similar functionality are seeking for different set of permissions. For example, among the examined apps, Viber app (for making free calls) is asking maximum permissions, whereas Skype, which provides similar functionality, is asking for less permissions.
5. Very few apps, only 26 (3%), are not asking for any permission. For example, Math Tricks (educational) and Doctor at Home (Medical).
6. We found that 276 of examined apps (34%), do not ask for any write permission apart from the top 5 permissions. Around 54% of examined apps do not ask for any write permission apart from the top 15 permissions.

Table 5: Number of apps with access in each category (from all 814 examined apps)

Permission Categories	Number of Apps with		
	Only Read Access	Write Access	No Access
User	71	612	131
Device	2	784	28
Carrier	676	70	68
External World	0	778	36

5. UTILITY OF THE PROPOSED FRAMEWORK

The implications of a permission is not often easy to evaluate, especially for a normal use. Our framework suggests a systematic and simple way to characterise and analyze what the app may be empowered to do by granting the requested permissions. This can be used to investigate the intrusiveness of apps into user space. The results can be utilized as:

5.1 Understanding Intrusiveness of App

Intrusiveness Score of an app can be used to describe its capabilities of intruding into user personal and operational space.

1. **Potential violation of user personal privacy**: An app can violate user personal privacy either by modifying the personal information (needs write permission in user space) or sending out personal information to outside world (needs read permission in user space and write permission in external world space). Given, $IS_A = (U, D, C, W)$ is the intrusiveness score for an app A, its capability to violate user personal privacy C_{upp} is:

$$C_{upp} = \begin{cases} True & \text{if U=5 or (U=1 and W=5)} \\ False & \text{otherwise} \end{cases}$$

2. **Potential violation of integrity of device**: An app can affect integrity of device if it has write permission on information related to device space. Given, $IS_A = (U, D, C, W)$ is the intrusiveness score for an app, its capability to affect integrity of device is :

$$C_{di} = \begin{cases} True & \text{if D=5} \\ False & \text{otherwise} \end{cases}$$

3. **Potential violation of user locational privacy**: An app can violate user locational privacy by sending out location related information to outside world. Thus such app should have at least read permission in carrier space along with write permission in external world space. Given, $IS_A = (U, D, C, W)$ is the intrusiveness score for an app, its capability to violate user locational privacy is :

$$C_{ulp} = \begin{cases} True & \text{if C>=1 and W=5} \\ False & \text{otherwise} \end{cases}$$

Table 6 depicts C_{upp}, C_{di}, and C_{ulp} values of examined apps.

Table 6: C_{upp}, C_{di}, C_{ulp} values for all examined apps

Intrusiveness	Number of Apps (out of 814)	
	TRUE	FALSE
Capability to violate user personal privacy : C_{upp}	683	131
Capability to violate integrity of device : C_{di}	784	30
Capability to violate user locational privacy : C_{ulp}	746	68

5.2 Helping developers in reducing intrusiveness of an App

The framework enables users to investigate an app for its power to intrude into different dimensions of information. On the other side, this should also enable the app developer to quickly evaluate the implications of the permission assignment on intrusiveness score and tune the app appropriately, in order to get a low intrusiveness score. Intrusiveness score is a better way to understand and annotate the permission system.

6. CONCLUSIONS

An app can, incidentally or accidentally, abuse the permissions granted to it. To avoid the undesirable manipulation of user personal information or operations of device, it is very important for the user to understand the permissions and implications of granting them. To identify such behaviour of an app, we define a new quality attribute called Intrusiveness, which characterizes the extent of violation happening into the personal and operational space of the user. We provide a framework, Intrusiveness Framework, to investigate intrusiveness of an app, by analysing the permissions and their impact in four categories : User Space, Device Space, Carrier Space, and External World Space. We examined 814 free Android apps using our framework. Results of our experiment shows that more than 50% of Android permissions have read or write access to user personal information. We found that only 16% of examined apps do not have any access to information in user space, which is 3% for device space, 8% for carrier space, and 4% for external world space. Our framework can be used to identify whether an app has capabilities to violate user personal privacy, user locational privacy or integrity of device. In our experiment, we found that 84% of apps are in a position to compromise User Personal Privacy, 96% can comprise Device Integrity and 92% can compromise Locational Privacy. Using this, a user can take decisions on whether to install the app or not, depending upon its capabilities to compromise privacy and integrity. Our framework can also be used by developers to reduce intrusiveness of their apps.

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support from Tata Consultancy Services and Department of Science & Technology, Govt. of India for this work.

8. REFERENCES

[1] Android api, permission element. `http://developer.android.com/guide/topics/manifest/permission-element.html`. Retrieved: February 2013.

[2] Android api, permission groups. `http://developer.android.com/reference/android/Manifest.permission_group.html`. Retrieved: February 2013.

[3] Android api, permissions. `http://developer.android.com/reference/android/Manifest.permission.html`. Retrieved: February 2013.

[4] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM.

[5] P. H. Chia, Y. Yamamoto, and N. Asokan. Is this app safe?: a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 311–320, New York, NY, USA, 2012. ACM.

[6] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*. The Internet Society, 2011.

[7] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[8] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[9] 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 Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[10] MobileScope. Mobile scope. `https://mobilescope.net/`. Retrieved: February 2013.

[11] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40, New York, NY, USA, 2012. ACM.