Android Permissions: A Perspective Combining Risks and Benefits

Bhaskar Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru
Department of Computer Science and CERIAS, Purdue University
305 N. University Street, West Lafayette, Indiana 47907-2107
{bsarma, ninghui, gates2, rpothara, crisn}@cs.purdue.edu

Ian Molloy
IBM Research TJ Watson
Hawthorne, NY, USA
molloyim@us.ibm.com

ABSTRACT

The phenomenal growth of the Android platform in the past few years has made it a lucrative target of malicious application (app) developers. There are numerous instances of malware apps that send premium rate SMS messages, track users' private data, or apps that, even if not characterized as malware, conduct questionable actions affecting the user's privacy or costing them money. In this paper, we investigate the feasibility of using both the permissions an app requests, the category of the app, and what permissions are requested by other apps in the same category to better inform users whether the risks of installing an app is commensurate with its expected benefit. Existing approaches consider only the risks of the permissions requested by an app and ignore both the benefits and what permissions are requested by other apps, thus having a limited effect. We propose several risk signals that and evaluate them using two datasets, one consists of 158,062 Android apps from the Android Market, and another consists of 121 malicious apps. We demonstrate the effectiveness of our proposal through extensive data analysis.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection, Access controls; K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection, Invasive software

General Terms

Security, Measurement

Keywords

Android, Malware

1. INTRODUCTION

As mobile devices become increasingly popular for personal and business use it is becoming increasingly more important to provide users with the ability to understand and control the benefit and risk of running apps on these devices. Mobile devices contain both traditional types of private data, such as contacts, email, and credit card numbers, and new types of resources, including accurate geolocation, audio recording, and making phone calls or sending premium SMS messages, all while maintaining constant internet connectivity on high speed wireless networks. Because of this shift in computing, a compromise can lead to greater exposure of personal information as well as direct financial impact. Mobile phones are increasingly being used for authentication at banks, as credit cards, e.g., Google Wallet, and to access corporate information remotely. At the same time, users seem to ignore potential problems, choosing to trust an app store to identify malware instead of evaluating risk on their own.

The Android platform has emerged as the fastest growing smartphone operating system being used by about 200 million devices, with around 700,000 devices being activated around the world daily. An increasing number of applications (or apps) are available for Android. The Google Android Market recently crossed more than 10 billion downloads. Such a wide user base coupled with ease of developing and sharing applications with the help of Android Market makes Android an attractive target for malicious application developers that seek personal gain while costing users' money and invading users' privacy. Indeed, recent events indicate an exponential increase in the number of malware for the Android system. Most of these malware are trojans that along with overt useful functionality perform covert malicious activities in the background. Examples of such malware activities include spyware that track users' private data and sending SMS to premium rate numbers.

To limit damages from security breaches, Google relies on the "principle of least privilege" and requires that an application request only for the most restrictive set of permissions for performing the task at hand. Android's current defense against malicious apps is to warn the user about permissions an app requires before an app is installed with the hope that the user will make the right decision. Specifically, Google's standard comment on malicious apps is: *"When installing an application, users see a screen that explains clearly what information and system resources the application has permission to access, such as a phone's GPS location. Users must explicitly approve this access in order to continue with* *the installation, and they may uninstall applications at any time. They can also view ratings and reviews to help decide which applications they choose to install. We consistently advise users to only install apps they trust."* This approach, however, is ineffective. The vast majority of Android apps require multiple permissions to execute. When a user sees essentially the same warning for almost every app, warnings quickly lose any effectiveness as the users are conditioned to ignore such warnings. There has been recent research [16] that confirms this ineffectiveness in the case of User Account Control (UAC), an attempt by Microsoft to protect its users in the context of Windows Vista that in some ways is similar to Android's approach. Motiee *et al.* [16] reported that 69% of the survey participants ignored the UAC dialog and proceeded directly to use the administrator account. Microsoft itself concedes that about 90% of the prompts are answered by "yes", suggesting that "users are responding out of habit due to the large number of prompts rather than focusing on the critical prompts and making confident decisions" [8].

Recently, risk signals based on the set of permissions an app requests have been proposed [7] as a mechanism to improve the existing warning mechanism for apps. Specifically, in [7], several rules that represent risky permissions are used to flag apps. However, such an approach is not very effective because it does not take into account the intended functionality of an app, that is, what the user expects the app to do, or what permissions are requested by other apps with similar functionality. While the potential risk of installing an app is best described by the set of permissions it requests, an approach using only permissions is insufficient because it does not capture the benefit offered by the application, or whether the risk is commensurate with the benefit. For example, SEND_SMS is a critical permission, as it enables an app to send out premium short messages, potentially costing the user money. While the permission can be used maliciously, it is also legitimately needed by certain communication applications—simply highlighting the fact that an app needs the permission is not effective. On the other hand, a game or a wallpaper app does not normally need to send short messages. If such an app requests the permission, then this represents an unusual risk not commensurate to its benefit.

In this paper, we focus on creating more effective risk signals about apps. An effective risk signal is a signal that: (1) has a simple semantic meaning that is easy to understand by both the users and the developers; (2) is triggered by a small percentage of apps; and (3) is triggered by many malicious apps. When a user observes that a risk signal is triggered by an app, understanding the reason helps the user make the decision whether to use the app. When a developer observes that her app triggered a risk signal, understanding the reason helps the developer to decide whether the app can be changed to not raise the signal.

Our approach takes into account both the benefit and the risk present with installing an app to create a more effective risk signal. Specifically, we propose to capture the benefit of an app by using the category and sub-category of the app. The Android Market currently divides apps into "Games" and "Applications", which are further divided into 8 and 26 sub-categories, respectively. We also propose a more effective way to capture risk by taking into account the occurrence of the permissions across apps with similar functionality. Our observation is that if a permission requested by an app is also requested by a large number of applications with similar functionality, then the permission is more likely to be needed and the risk associated with installing the app is smaller. On the other hand, if a permission requested by an app appears to be requested by a very small number of applications with similar functionality, then the risk of allowing the permission by installing the app is higher.

As an example, one risk signal that we propose is what we call the Category-based Rare Critical Permission signal, denoted CRCP(θ). From Android's current list of 122 permissions, we choose 26 that we call critical permissions. For each app category, we call any critical permission that is requested by less than θ percent of apps in this category a θ-Rare Critical Permission (θ-RCP) in this category. Any app that requests one of the θ-RCP's in its category triggers the CRCP(θ) risk signal. We also consider the RCP(θ) signal, which is triggered when an app requests a critical permission that is requested by less than θ percent of all apps, and signals based on rare pair of permissions.

We envision such risk signals to be used as follows, using CRCP(θ) as an example. The Android Market's webpage for an app can indicate whether the app triggers CRCP(θ) for some standard values of θ. We could present them with the ability to select a category for the app other than its assigned category before installing it. This step of the user selecting the category is essentially identifying the potential benefit of installing the app. We may also give the user a choice to select a threshold for θ to display the CRCP(θ) signal. For each choice of (θ), it is also displayed what percentage of apps will trigger the signal for that (θ) so that the user has a better understanding of how (in)frequently this signal is triggered. Note that the CRCP(θ) signal is often triggered by more than θ percent of apps, because there is often more than one θ-RCP for a category, and requesting any one triggers the signal. The user is then warned if the app triggers the signal. As typical users have many apps to choose from for a certain task, users can choose to avoid apps that trigger the signal if it is raised for a small percentage (e.g., less than 10%) of apps. If the user is unable to select a category, then we can resort to the RCP(θ) signal. Such a risk signal indicates to the user when an application may be over provisioned, and thus represents excessive risk given the benefit they expect to receive.

We point out that our idea of utilizing category of apps and rarity of permissions can be deployed, even without the risk signal notion. For example, rather than showing all permissions requested by an app. The interface for showing permissions (both on Android market webpage and on the permission warning page shown to the user before the installation of an app) should sort the permissions by their frequency within its category (or over all apps when the category information is not available), list the least frequent first, and include the frequency together with each permission. The frequency can also be color coded, e.g., using red for the rarest permissions. Furthermore, the frequent permissions can be hidden by default (and available with a "show all" button).

In summary, the contributions of this paper are as follows:

- We introduce the notion of risk signals combining risks with benefits. This approach can provide a first line of defense in the case of downloading apps on Android platforms. It is applicable to other contexts such as Facebook applications and Chrome extensions as well.

- We propose a general formulation of risk signals exploiting rare critical permissions and rare pairs of critical permissions, as well as the category information of an app.

- We evaluate the effectiveness of our proposed risk signals by using two datasets. The first dataset consists of 158,062 Android apps and it was collected from Android market website in February 2011. The second dataset consists of 121 malicious apps and it was obtained from the Contagio Malware

Dump repository (http://www.contagiodump.blogspot.com). We show that our proposed risk signals, especially CRCP, are effective.

The rest of the paper is organized as follows. We present a description of the Android platform and the current warning mechanism in Section 2. Section 3 discusses the date sets that we have collected and certain characteristics about permissions in that data. In Section 4 we discuss how this data can be used to measure the risk that a certain app might introduce. We then present results of our finds for these risk signals in Section 5. We finish by discussing related work and concluding in Section 6 and 7.

2. ANDROID PLATFORM

In this section we provide an overview of the current defense mechanism provided by the Android platform and discuss its limitations.

2.1 Android Development Process

Android is an open source software stack for mobile devices that includes an operating system, an application framework, and core applications. The operating system relies on a kernel derived from the Linux kernel. The application framework consists of the Dalvik Virtual Machine that runs `.dex` files. Applications are written in Java using the Android SDK, compiled into `.dex` (Dalvik Executable), and packaged into `.apk` (Android package) archives for installation.

To be able to submit applications to the Android Market, an Android developer should obtain a publisher account. When submitting an Android application to the Android Market, each *.apk* binary is assigned a webpage on the Android Market. This webpage contains *meta-information* that keeps track of information pertaining to the application (name, category, version, size, prices) and its usage statistics (rating, number of installs, number of reviews).

2.2 Permissions in Android Platform

The current support provided by Android in addressing the problem of malware consist of sandboxing each application and warning the user about the permissions that the application requested. Specifically, each application runs as a separate process on a virtual machine of its own and by default does not have permissions to carry out actions or access resources which might have an adverse effect on the system or on other apps. For example, an application cannot send SMS, read contacts, or change system settings like Bluetooth, by default. However, an application can explicitly request these privileges through permissions.

When a user downloads an app through the Android Market, the user is taken through two screens. The first screen has information such as description, reviews, and screenshots of the app. The user has to select "Download" to move to the next screen. The second screen displays permissions requested by the application. Installing the application means granting the application all the requested permissions. The permissions are displayed under various categories to indicate their functionality. For example, permissions associated with messaging like READ_SMS and WRITE_SMS are grouped under the same category. A user can find out detailed information about a permission by clicking or tapping on it. This helps the user understand the potential risks of installing the application. For example, "FINE_LOCATION", a GPS-related permission, carries the following description "Access fine location sources such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are, and may consume additional battery power."

2.3 Limitations

Android's current permission warning approach has been very ineffective in curbing malicious applications. This is partly because the current mechanism of displaying permissions fails as an effective risk communication mechanism, as it warns the user about dangerous permission on almost all permissions. Many applications may have a legitimate need to access fine-grained GPS locations, for example, an application for reporting local weather can provide a benefit to the user by accessing their location. Many other applications use the FINE_LOCATION permission to provide a benefit to the user, hence the user will see the same warning again and again. Most of the time, the user will want to install the app despite this warning. This conditions users to ignore such warnings. When a malicious app comes along, a user has already been conditioned to ignore such information and most likely does not even look at the permissions.

Such effect has been discussed in the literature. In [13], Felt *et al.* analyzed 100 paid and 856 free Android applications, and found that "*Nearly all applications (93% of free and 82% of paid) ask for at least one 'Dangerous' permission, which indicates that users are accustomed to installing applications with Dangerous permissions. The INTERNET permission is so widely requested that users cannot consider its warning anomalous. Security guidelines or anti-virus programs that warn against installing applications with access to both the Internet and personal information are likely to fail because almost all applications with personal information also have INTERNET.*"

Felt *et al.* argued "*Warning science literature indicates that frequent warnings de-sensitize users, especially if most warnings do not lead to negative consequences [22, 15]. Users are therefore not likely to pay attention to or gain information from install-time permission prompts in these systems. Changes to these permission systems are necessary to reduce the number of permission warnings shown to users.*"

There is a parallel between the Android Model and Windows UAC prompt. Both are designed to inform the user of some potentially harmful action that is about to occur, in UAC's case that a process is trying to elevate it privileges in some way, and in Android's case that you are installing an app that will have all these elevated privileges. The difference is that UAC encourages the developer to work with fewer privileges since this will lead to a smoother user experience. However with Android there is no obvious feedback loop to the developer. An application requires the same effort to use if it requires one privilege or multiple, the only difference is the length of the permission list which is difficult to see on a small screen anyway.

While the ineffectiveness of the Android permission warning mechanism has been recognized, no effective solution has been proposed. In this paper, we aim at investigating how to improve the current state of the art in communicating risky permissions to the user.

3. DATASETS

In this section, we describe the two datasets we used in our study of Android app permissions. Below we describe the datasets and their characteristics.

3.1 Datasets Description

The "market dataset". The first dataset consists of 158,062 Android applications. We created this dataset during February 2011 by crawling the Android Market website and downloading the webpages for all the applications. We then extract the application package name, the category of the application, and the list of permissions that this application requests during installation. The Android Market divides apps into "Games" and "Applications", each of which is further divided into sub-categories. Back in February 2011, there were 6 sub-categories under Games and 24 sub-categories under "Applications". Since February 2011, Android market has created some new sub-categories, and currently the Android market has 34 sub-categories in total.

This dataset is more comprehensive and significantly larger than others that have been studied in the literature. For example, a dataset of 311 apps was used in [7], 940 was used in [11], and 1,100 was used in [6]. We expect that the vast majority of the apps in the market dataset to be benign; however, it may contain a very small percentage of malicious apps, as well as some apps that may be called grayware, as they may carry out questionable actions without sufficient user notification or approval.

The "malware dataset". The second dataset consists of 121 apps that are known to be malicious. We obtained this dataset from the Contagio Malware Dump (http://www.contagiodump.blogspot.com) repository. We downloaded all samples of malicious Android apps from the website, and obtained 180 Android Package (APK) files. However, there are duplicates in this dataset. After removing the duplicates, we are left with 121 samples. For each malware sample, we extracted the permissions requested using the AndroidManifest.xml file present inside the package file. For these malicious apps we do not have their category information. We call this the "malware dataset". This dataset is larger than other malicious Android datasets in the literature. As a comparison, 46 were used in [12], however these include malware for iOS, Android and Symbian platforms.

3.2 Frequently Requested Permissions

Table 1 shows the top-20 most frequently requested permissions by applications in the two datasets. We observe that overall malicious apps request more permissions than those in the market dataset. For some permissions, the percentages of malware apps requesting them are much higher than those in the market dataset. For example, SEND_SMS is requested by 64.46% of the malicious apps, but only 4.83% in the market dataset. This is probably due to the fact that many malicious apps use premium SMS messages to benefit financially. Also interesting is the fact that READ_HISTORY_BOOKMARKS is requested by 42.12% of the malicious apps, but none of the apps in our market dataset.

Another observation is that some permissions are requested by such a high percentage of apps in the market that warning that an app requests the permission is meaningless. We also observe that one Android permission often controls several different types of accesses, often with very different sensitivities. These observations lead us to argue that some of the Android permissions should be further divided. For example, it is clearly desirable to control the SEND_SMS permission, however, it is requested by 4.83% of apps in the market dataset. For apps in some categories, this ratio is much higher, because many apps have legitimate need to send SMS messages. We argue that premium-rate SMS messages should be controlled by a separate permission, as such messages incur much higher monetary costs to the user from normal SMS messages, and very few apps have legitimate reasons to send premium-rate SMS messages. For another example, the "READ_PHONE_STATE" permission is requested by 24.99% of apps in the market dataset and 80.99% in the malware dataset. This permission enables an app to get several kinds of information: including the phone number, the serial number of this phone, whether a call is active, the number that call is connected to, and so on. It seems much more natural to separate these into different permissions.

Permission	Benign	Malicious
INTERNET	68.50 (1)	93.38 [1]
ACCESS_NETWORK_STATE	30.97 (2)	42.98 [8]
READ_PHONE_STATE	24.99 (3)	80.99 [2]
WRITE_EXTERNAL_STORAGE	24.14 (4)	59.50 [4]
ACCESS_COARSE_LOCATION	18.17 (5)	43.80 [7]
ACCESS_FINE_LOCATION	17.22 (6)	35.53 [12]
WAKE_LOCK	13.07 (7)	23.14 [18]
VIBRATE	12.84 (8)	23.14 [19]
ACCESS_WIFI_STATE	8.09 (9)	28.92 [16]
RECEIVE_BOOT_COMPLETED	7.99 (10)	23.14 [20]
READ_CONTACTS	7.50 (11)	47.11 [6]
GET_TASKS	5.32 (12)	5.78 [30]
CALL_PHONE	5.10 (13)	31.40 [14]
SEND_SMS	4.83 (14)	64.46 [3]
SET_WALLPAPER	4.75 (15)	30.57 [15]
CAMERA	4.35 (16)	5.78 [30]
GET_ACCOUNTS	4.31 (17)	4.95 [31]
RECEIVE_SMS	4.29 (18)	40.49 [10]
WRITE_SETTINGS	3.90 (19)	7.44 [27]
PROCESS_OUTGOING_CALLS	3.64 (20)	4.13[36]
READ_SMS	3.43 (21)	47.11 [5]
READ_HISTORY_BOOKMARKS	0 (113)	42.14 [9]
WRITE_HISTORY_BOOKMARKS	0(113)	37.19 [11]
WRITE_CONTACTS	1.99 (23)	32.23 [13]
MOUNT_UNMOUNT_FILESYSTEMS	1.25 (28)	26.44 [17]

Table 1: Table showing the top 20 most used permissions in the two datasets. 15 permissions occurred in both of top-20 lists. A total of 25 permissions are included. Column 2 shows the percentage (and ranking) of permissions in the market dataset, and column 3 shows the percentage (and ranking) in the malicious dataset.

Android tries to limit the number of permissions because having more of them simply lengthen the list of permission warning a user is going to see, further decreasing the usability of something that users are likely to ignore. However, with the risk signal approach we investigate in this paper, having more permissions does not lead to more meaningless warnings. In short, if one can devise effective risk signals based on permissions, then finer-grained permissions could be deployed, improving Android security.

4. RISK SIGNALS

In this section we describe the risk signals we propose for Android applications based on the permissions they request.

4.1 Design Goals for Risk Signals

When designing a risk signal two relevant measures are the **warning rate** which defines how often a user receives warnings generated by the risk signal and the **detection rate** which defines what percentage of malicious apps will trigger the signal. To avoid over-exposing users to warnings generated by risk signals, it is desirable that a risk signal has a low warning rate. To be effective at detecting malicious applications a risk signal should have a high detection rate. Moreover a risk signal should be easily understandable by end users.

Because there is no guarantee that the market data contains no malware, a warning rate of close to 0 is not necessarily desirable. At the same time the boundary between benign and malicious apps

Risk	Permission	Permission Allows	% market	% malware
	ACCESS_COARSE_LOCATION	access to coarse (e.g., Cell-ID, WiFi) location	18.17	43.80
	ACCESS_FINE_LOCATION	access to fine (e.g., GPS) location	17.22	35.53
	PROCESS_OUTGOING_CALLS	monitor, modify, or abort outgoing calls.	3.64	4.13
	READ_CALENDAR	read the user's calendar data.	0.64	0.82
	READ_CONTACTS	read the user's contacts data.	7.50	47.11
Privacy	READ_HISTORY_BOOKMARKS	read the user's browsing history and bookmarks.	0	42.14
	READ_PHONE_STATE	read only access to phone state.	24.99	80.99
	READ_SMS	read SMS messages.	3.43	47.11
	RECEIVE_MMS	monitor, record, or process MMS msgs	0.18	5.78
	RECEIVE_SMS	monitor, record, or process SMS msgs	4.29	40.49
	RECORD_AUDIO	record audio	1.91	4.13
	RECEIVE_WAP_PUSH	monitor incoming WAP messages	0.063	4.13
	READ_LOGS	read low-level log msgs	0.76	8.26
	CALL_PHONE	make a phone call w/o user's confirmation.	5.10	31.40
Monetary	INTERNET	open network sockets.	68.50	93.38
	SEND_SMS	send SMS messages.	4.83	64.46
	MOUNT_UNMOUNT_FILESYSTEMS	mount / unmount file sys for removable storage.	1.25	26.44
	WRITE_CALENDAR	write the user's calendar data.	0.49	2.47
Other	WRITE_CONTACTS	write the user's contacts data.	2.00	32.23
	WRITE_HISTORY_BOOKMARKS	write the user's browsing history and bookmarks.	0	37.19
	WRITE_SMS	write SMS messages.	3.10	22.31
	WRITE_EXTERNAL_STORAGE	write to external storage	24.14	59.50
Damage	NFC	perform I/O operations over NFC	0.006	0
	GET_ACCOUNTS	access the list of accounts in the Accounts Service	4.31	4.95
	BLUETOOTH	connect to paired bluetooth devices	0.57	4.95
	BLUETOOTH_ADMIN	discover and pair bluetooth devices	0.46	4.13

Table 2: Table displaying list of critical permissions

is blurred as many apps are unnecessarily over-privileged [11]. In this sense, raising warnings for such over-privileged apps is not a "false" positive; thus one should not equate the warning rate with the false positive rate in intrusion detection. On the other hand, an overly high warning rate is certainly undesirable because when users frequently see a warning, it becomes less effective. If a risk signal has a relative low warning rate (say, between 2% and 5%), then among every 100 apps the user investigates, on average between 2 to 5 of them raise the warning. As this is rare enough, and in the mobile platform market, a user often has choices among multiple competing apps with similar functionalities, then the user is likely to avoid these apps. In this paper, we generally restrict our warning rate to be in the range of 1% to 10%.

While we desire higher detection rate, one should be careful to assign too much weight when interpreting this rate in our analysis results. We are using a dataset of 121 malware apps. While this is the largest dataset in the literature we are aware of, it is difficult to argue that they are representative of all malware apps. More importantly, these malware apps were written when over-provisioning permissions were not punished. If approaches proposed in this paper are adopted in different forms, malware app authors may choose to request only the permissions absolutely necessary for the malicious task with the aim of avoiding detection. For example, from Table 1, we observe that 42.14% malicious apps request READ_HISTORY_BOOKMARKS, while no app in the market dataset does so. This leads to a trivial risk signal. However, it is difficult to argue that this will be effective for detecting future malware apps.

Rather than focusing only on the warning rate and the detection rate, we want to design risk signals that are more principled, in the sense that they could rule out apps with critical permissions that could potentially be abused. At the same time, we desire risk signals to have a relative low (between 1% and 10%) warning rate, and a relative high detection rate. Another property that we desire is that the risk signals should be easy for end users to understand. After all, no risk signal can be used to stop the installation of an app by itself. The ultimate decision lies with the end user. If the user can understand why a warning is raised, then there is higher chance that he can process the information accordingly.

Having an easy-to-understand risk signal also has the potential to benefit the overall eco-system of Android apps. The risk signal can be displayed on Android websites. If a small percentage of apps are identified as risky, and there is clear reason why, such as requesting a rare permission, this gives developers incentives to not request permissions the app can function without, since requiring too many permissions now reflects badly on an app. This creates a positive feedback loops as apps requesting fewer permissions will cause other apps that request many permissions to increasingly "stand out".

4.2 Permission Based Risk Signals

We consider two classes of risk signals: those that are created with some knowledge of signature of malicious apps, and those that do not use such knowledge.

Signals aware of malicious application signatures.

We consider risk signals that are constructed using the permissions of apps from both benign and malicious apps. For example, in our case the signals will be based on both the market and malicious apps datasets. We expect that risk signals in this category give the best tradeoffs between warning rate and detection rate. However, they run the risk of over fitting our particular malware dataset, which may not be representative of other yet undiscovered malware.

Support Vector Machines: Support Vector Machines (SVMs) [26] have gained significant popularity in the research community in the recent years. In its simplest linear form, an SVM is a hyperplane that separates a set of positive examples from a set of negative examples with maximum interclass distance, the *margin*. Our datasets, however, cause difficulty for the standard SVM algorithm, because the size of the market dataset is several order of magnitude larger than that of the malware dataset. There has been recent research [4] that indicated that when the training data sets with uneven class sizes were used, the standard support vector machine was undesirably biased towards the class with the large training size. Thus, we leverage the weighted variation of SVMs introduced by Huang et al. [14] due to the uneven nature of our datasets. Weighted SVMs can be expressed as an optimization problem:

$$\begin{aligned} \underset{w,b,\xi}{\text{minimize}} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{l} s_i\xi_i \\ \text{subject to} \quad & y_i(w \times \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi \geq 0, \forall_i i = 1, 2, .\therefore, l \end{aligned}$$

where C is a parameter that is empirically selected and is taken for each training sample without discrimination, s_i is a weighting factor for the i^{th} training sample. We leverage the SVM package called LibSVM [3] to carry out our analysis using the Weighted SVM variation and an RBF [23] kernel.

In order to run support vector machines we classify our data into two categories with labels, -1 for benign apps and 1 for malicious apps. Combining this data and using the permissions as features of this data, we are then able to use SVM to train and classify our datasets.

Signals oblivious of malicious application signatures.

Risk signals based only on apps from the Android market are more robust as they are not tuned to detect malicious apps in our particular dataset, and aim only at detecting apps that request too many permissions. Furthermore, we want a principled approach where the signals use only critical permissions so that such signals are more difficult to evade.

From Android's current list of 122 permissions, we choose 26 permissions that we call critical permissions. They are listed in Table 2. These 26 permissions were chosen because we believe they are critical for the security and privacy of end users. These permissions allow an app to infringe upon the privacy, cause monetary loss or damage otherwise. They were chosen before we conducted any experiments with the malware dataset. After conducting experiments, we realized that two of 26 permissions were very helpful in identifying malware apps in our dataset. They are READ_HISTORY_BOOKMARKS (requested by 42.14% of malware apps) and WRITE_HISTORY_BOOKMARKS (requested by 37.19% of malware apps); both are not requested at all in the market dataset. We feel that using these two permissions inflated our results. To avoid such a positive bias, in most experiments we remove these two from the critical set, and use the remaining 24 permissions. When we compare results from these two sets of critical permissions, we use P26 and P24 to differentiate them.

Rare Critical Permissions (#RCP(x) $\geq \theta$). The first risk signal we consider is whether an app has a rare critical permission. We say that a critical permission is rare with respect to a threshold x if it occurs in less than x% of the Android Market applications. This signal is triggered by an app if it requests one or more rare critical permissions. One advantage of this signal is that the semantic meaning is very simple and easy to understand.

Rare Pairs of Critical Permissions ($\#RPCP(y) \geq \theta$). We consider a pair of critical permissions to be rare with respect to a threshold y if the individual permission's frequency is above threshold y but the frequency of occurrence of the two permissions as a pair is below y, and we define this as #RPCP(y). That is, we consider a pair of critical permissions to be rare if the permissions involved in the pair are themselves not rare (above threshold x) but their occurrence together in an app is rare (below threshold y).

Combination of RCP and RPCP ($\#RCP(x) + w * \#RPCP(y) \geq \theta$). In this signal we use a linear combination of #RCP(x) and #RPCP(y) to calculate a risk score, and then chose a threshold θ to determine whether the signal should classify an app as risky for our experiments. The value w can be viewed as representing the importance of rare pairs of critical permissions relative to rare critical permissions. We point out that while this is more general than the signal "#RCP(x) $\geq \theta$" and may give better results, it is more complicated for users to understand and for developers to take actions to avoid triggering the signal.

4.3 Permission and Benefit Based Risk Signal

We believe that taking into account the intended functionality or benefit provided by an app should result in an effective risk signal. We use category of an app to determine the intended benefit, because we hypothesize that apps in different categories often request different kinds of permissions. To test this hypothesis, we studied the percentages of applications requesting the SEND_SMS, FINE_LOCATION and READ_CONTACTS permission across the 30 categories in the market dataset, which is shown in Figure 1. As expected SEND_SMS and READ_CONTACTS are used the most in the Communication category, while FINE_LOCATION is used most frequently in Transportation, Travel, and Weather. The resulting graph supports our hypothesis.

We propose the Category-based Rare Critical Permission (CRCP) signal. We use CRCP(θ) to denote this signal, and it is defined as follows. For each category, we call any critical permission that is requested by less than θ percent of apps in this category a θ-Rare Critical Permission (θ-RCP) in this category. Any app that requests one of the θ-RCP's in its category triggers the CRCP(θ) risk signal. Similarly, we define the RCP(θ) signal to be triggered when an app requests a critical permission that is requested by less than θ percent of all apps.

The central idea behind the CRCP signal can be summarized as *comparing the intended functionality of an app (inferred from its category) with its actual functionality (obtained from its permission set) and reporting if there is any mismatch between the two.*

5. EXPERIMENTAL RESULTS

We evaluate the risk signals introduced in Section 4.2 using the market dataset and malware dataset and report the results here.

Analysis of Permission Based Risk signals. In order to get a baseline we first apply the only other mechanism that has been published to identify risk based on permissions, namely Kirin [7]. Kirin has several rules that dictate when an app is considered risky. We considered only 7 of the 9 rules in Kirin, because the other 2 rules refer to permissions that are no longer supported. Table 3 shows the result. It shows that at a 6.53% warning rate, Kirin has a 32.2% detection rate. We consider the warning rate acceptable; however, its ability to warn a user for malware in our dataset is low.

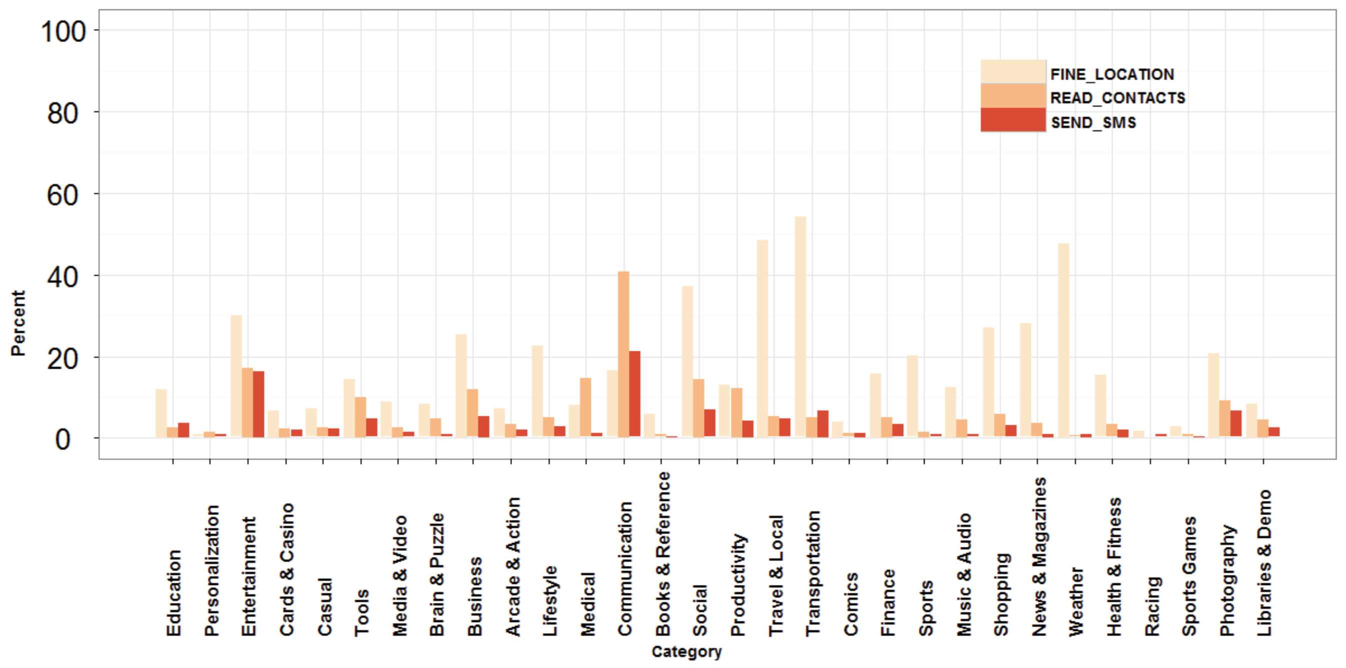

Figure 1: Graph showing percentage of applications using SEND_SMS, READ_CONTACTS and FINE_LOCATION across 30 categories

Rule	% malicious apps	% Android Market apps
SET_DEBUG_APP	0	0.01
READ_PHONE_STATE, RECORD_AUDIO, INTERNET	4.13 (5/121)	1.03
PROCESS_OUTGOING_CALLS, RECORD_AUDIO, INTERNET	0	0.08
ACCESS_FINE_LOCATION, INTERNET, RECEIVE_BOOT_COMPLETED	9.91 (12/121)	4.50
ACCESS_COARSE_LOCATION , INTERNET, RECEIVE_BOOT_COMPLETED	11.57 (14/121)	4.53
RECEIVE_SMS, WRITE_SMS	19 (23/121)	2.76
SEND_SMS, WRITE_SMS	20.66 (25/121)	2.87
Failing at least one rule	32.23 (39/121)	6.53

Table 3: Kirin results. The table shows for each rule in Kirin, the percent of malicious apps (out of 121) that fail the rule, and the percent of Android Market apps that fail it. The last row shows the percent of apps in each dataset failing at least one rule.

Figure 2 shows the ROC curves of using weighted SVM and seven other risk signals using rare critical permissions and rare pair of critical permissions. As can be seen, the SVM method unsurprisingly gives the best result. It is followed by the signal $\#RCP(2) + \#RPCP(1) \geq \theta$. Table 5 shows the numerical values of several data points for #RCP(θ) $\geq$ 1, the simplest signal among the seven, and $\#RCP(2) + \#RPCP(1) \geq \theta$, the best performing one.

In the SVM method, we use 10-fold cross validation, which randomly selects parts of the data for training and the rest for testing, repeating this 10 times to get a reasonable result. We modified the standard libSVM code to also extract warning rate and detection rate for a given test. Due to the nature of our data we used weighted SVM, varying the weights for the malicious data set so that it had more significance in the training, adjusting these weights led to different trade-offs between the warning rate and the detection rate. For a very low warning rate of .05% we can can identify 50% percent of the malware. Using a different weight when training results in 71% detection rate and 2.4% warning rate.

We point out that while SVM outperforms other methods, this is expected for several reasons. First, SVM uses all permissions in its feature selection, as opposed to only the risky permissions, while all other signals use P24. Second, SVM is also trained on malware data. Finally, SVM is a sophisticated machine learning model. We view the sophistication of SVM as also its major disadvantage, as it is very difficult to explain to a user why a warning is raised, or to a developer how to avoid the warning signal.

In the first row in Table 5, we observe that for a θ of 2%, RCP has a warning rate of 6.36%, and a 52.90% detection rate, easily outperforming Kirin. That is, 6.26% apps in the market dataset request a critical permission (in P24) that is requested by less than 2% apps, and 52.90% of apps in the malware dataset does so.

The second row of Table 5 demonstrates the results of this approach for $\#RCP(2) + \#RPCP(1) \geq \theta$. Note that for $\theta = 2$, that is, the signal is raised is the when an app either requests a 2%-rare critical permission, or a pair of critical permissions that is 1%-rare, this method identifies 66.94% of malicious apps with relatively low warning rate of 7.62%.

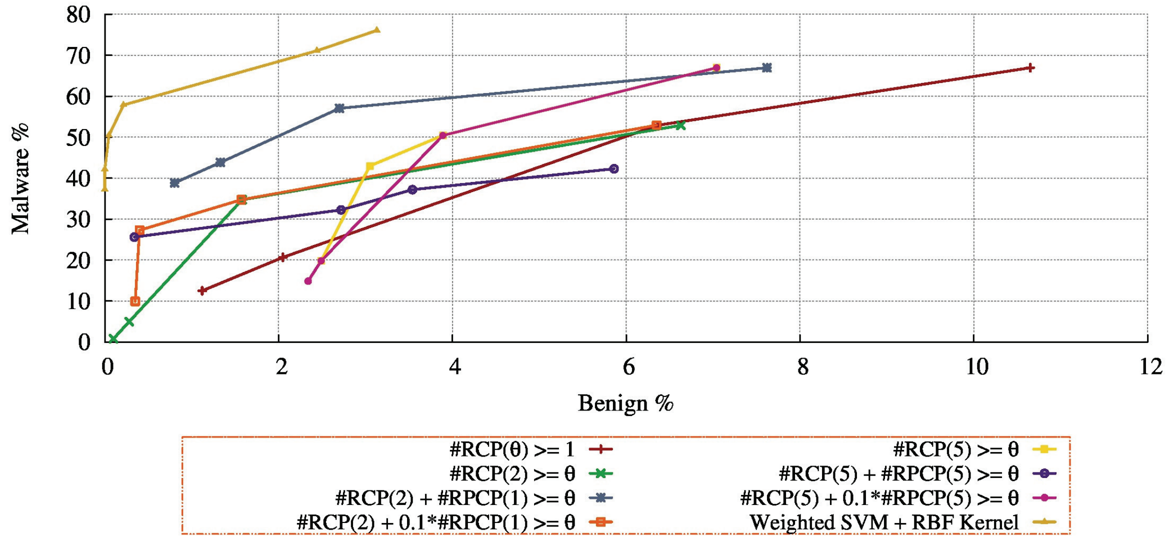

Figure 2: ROC curves for seven risk signals plus the SVM method. The X-axis is the warning rate, and the Y-axis is the detection rate.

	Android Market apps	Malicious apps (total 121)								
	Own category	All 30 cat.		At least 27 cat.		At least 25 cat.		At least 18 cat.		At least 1 cat.
	P26 / P24	P26	P24	P26	P24	P26	P24	P26	P24	P26 / P24
$\theta = 1\%$	2.32	49.59	4.13	47.11	14.05	55.37	41.32	68.60	57.85	86.78
$\theta = 2\%$	4.90	60.33	30.57	62.81	51.23	66.11	55.38	74.38	68.59	88.42
$\theta = 3\%$	6.34	66.12	37.19	66.94	56.20	68.60	62.81	83.47	83.47	88.42
$\theta = 4\%$	8.12	71.07	44.62	70.24	60.33	80.99	80.99	83.47	83.47	88.42
$\theta = 5\%$	9.17	75.20	44.62	77.69	77.69	80.99	80.99	84.29	84.29	88.42

Table 4: Percentages of apps triggering the CRCP(θ) signal in their own category. It also shows the percentages of malware that trigger the signal for different number of categories.

Results for Benefit Adjusted Risk Signals.

The Benefit Adjusted Risk Signal works by taking into account the category of an app. Since the malicious apps did not come with a category we count the number of categories a malicious app is marked as risky in.

Table 4 shows the evaluation results of the CRCP(θ) signal, which is raised when an app requests a critical permission that is requested by less than $\theta\%$ apps in the category. The table has one row for each threshold. The second column shows the warning rate for Android Market Dataset apps. The remaining columns show the numbers of malware that trigger the risk signal in all 30, at least 27, at least 25, at least 18, and at least 1 categories of the Android Market. The label P26 indicates that the analysis results is using 26 critical permissions and P24 indicates usage of 24. A label "P26 / P24" indicates that the results of using either set of permissions are the same. We consider the percentages of malware classified as risky in at least 25 categories as an indicator of how successful the Benefit Adjusted Risk Signal would be in case we could determine the category of the malware accurately. We see a warning rate of 6.34% with a corresponding detection rate of 62.81%, which is an improvement over RCP's 6.36% warning rate and 52.89 detection rate, but is similar to the best non-category based risk signals. We also see a warning rate of 8.12% corresponds to a 80.99% detection rate. This seems to suggest that at a slightly higher warning rate, this risk signal performs really well.

Discussion. There are several reasons why the distribution of a malware may be affected if it raises a risk signal for some categories, but not others. First, many malware apps try to impersonate a popular app, such as Angry Bird, which belongs to a particular category. Hence the category of these malware apps are limited to be the same as the original app, especially when users are asked to select the category. Second, to speed up propagation a malware may be uploaded in more than one categories. For many of these categories, a warning may be raised.

Moreover, even though the results of category based approach are comparable to the category-independent signals using permission pairs, it has the advantage of being simpler, and easier to comprehend for both users and developers as to why an app triggers the signal. Taking all the above mentioned points we believe that CRCP approach is the most promising one in practice.

6. RELATED WORK

For malware detection, a detailed knowledge of application's characteristics is essential. To achieve this, static analysis involves various binary forensic techniques, including decompilation, decryption, pattern matching [24] and static system call analysis [21]. The common ground for all these techniques is that the code being analyzed is not executed. Hence, malware are generally filtered

Risk Signal	θ	AM%	Mal%	θ	AM%	Mal%	θ	AM%	Mal%	θ	AM%	Mal%
#RCP(θ) ≥ 1	5	12.93	83.47	2	6.36	52.89	1	2.05	20.66	0.5	1.12	12.40
#RCP(2) + #RPCP(1)$\geq \theta$	1	7.62	66.94	2	2.70	57.02	3	1.33	43.80	4	0.802	38.84

Table 5: Table showing the effect of various risk signals. The first column gives the description of the risk signal: #RCP(x) is whether an app requests a critical permission that is requested by no more than $x\%$ of Android Market apps. The second row shows percent of apps having risk scores above threshold θ using #RCP(2) + #RPCP(1)$\geq \theta$ to calculate risk score. In the table AM refers to the Android Market apps and Mal refers to malicious apps

through *signatures*. While this is a popular approach amongst many anti-virus vendors, this method cannot detect new malware whose signature does not exist in the database i.e. malicious code patterns have to be known in advance.

Felt *et al.* [9] use static analysis to determine whether an Android application is overprivileged. It classified an application as overprivileged if the application requested a permission which it never actually used. They apply their techniques to a set of 940 applications and find that about one-third are overprivileged. Their key observation was that developers are trying to follow least privilege but sometimes fail due to insufficient API documentation. Another work by the by Felt *et al.* [10] surveys applications (free and paid) from the Android Market. Their key observation was that 93% of free apps and 82% of paid apps request permissions that they deem as "dangerous". While this does not reveal much out of context, it demonstrates that users are accustomed to granting dangerous permissions to apps without much concern. Neither of these works actually attempt to detect or categorize malicious software.

Enck et al. [7] developed a system that examined risky permission combinations for determining whether the permissions declared by an application satisfy a certain global safety policy. This work manually specifies permission combinations such as WRITE_SMS and SEND_SMS, or FINE_LOCATION and INTERNET, that could be used by malicious apps, and then performs analysis on a dataset of apps to identify potentially malicious apps within that set. Another work by Enck et al. [6] makes an effort to decompile and analyze the source of applications to detect further leaks and usage of data.

Barrera *et al.* [2] present a methodology for the empirical analysis of permission-based security models using self-organizing maps. They apply their methodology to analyze the permission distribution of close to thousand applications. Their key observations were (i) the INTERNET permission is the most popular and hypothesized that most developers request this to request advertisements from remote servers, (ii) Location-based permissions are usually requested in pairs i.e. access to both fine and coarse locations is requested by applications in a majority of cases by developers and (iii) there are some categories of applications such as tools and messaging category where pairs of permissions are requested.

Au *et al.* [1] survey the permission systems of several popular smartphone operating systems and taxonimize them by the amount of control they give users, the amount of information they convey to users and the level of interactivity they require from users. Further, they discuss several problems associated with extracting permissions-based information from Android applications.

Dynamic Analysis: Another research direction in Android security is to use dynamic analysis. Portokalidis [19] propose a security solution where security checks are applied on remote security servers that host exact replicas of the phones in virtual environments. In their work, the servers are not subject to the constraints faced by smartphones and hence this allows multiple detection techniques to be used simultaneously. They implemented a prototype and show the low data transfer requirements of their application.

Enck *et al.* [5] perform dynamic taint tracking of data in Android, and reveal to a user when an application may be trying to send sensitive data off the phone. This can handle privacy violations since it can determine when a privacy violation is most likely occurring while allowing benign access to that same data. However, there is a whole class of malicious apps that this will not defend against, namely security and monetary focused malware which send out spam or create premium SMS messages without accessing private information.

Security & Access Control: Research in this direction is geared towards furthering usable security associated with mobile phones by improving the fundamental security and access control models currently in use. This type of research entails introducing developer-centric tools [25] that enforce principle of least privilege, extending permission models and defining user-defined runtime constraints [17, 18] to limit application access and detecting applications with a malicious intent [5, 20].

Nauman et al. [17] present a policy enforcement framework for Android that allows a user to selectively grant permissions to applications as well as impose constraints on the usage of resources. They design an extended package installer that allows the user to set constraints dynamically at runtime. Ongtang [18] present an infrastructure that governs install-time permission assignment and their run-time use as dictated by application provider policy. Their system provides necessary utility for applications to assert and control the security decisions on the platform. Vidas [25] present a tool that aids developers in specifying a minimum set of permissions required for a given mobile application. Their tool analyzes application source code and automatically infers the minimal set of permissions required to run the application.

7. CONCLUSIONS

We have proposed the notion of using effective signals to improve Android security, and introduced risk signals combining information about the permissions requested by an app, the function category of an app, as well as what permissions other apps request. Through evaluation using two datasets, we demonstrate the effectiveness of our approach.

Acknowledgement

This paper is based upon work supported by the United States National Science Foundation under Grant No. 0905442.

8. REFERENCES

[1] K. Au, Y. Zhou, Z. Huang, P. Gill, and D. Lie. Short paper: a look at smartphone permission models. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 63–68. ACM, 2011.

[2] D. Barrera, H. Kayacik, P. 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*, pages 73–84. ACM, 2010.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] H. Chew, D. Crisp, R. Bogner, and C. Lim. Target detection in radar imagery using support vector machines with training size biasing. In *Proc. Int. Conf. on Control, Automation, Robotics, and Vision (ICARCV)*, 2000.

[5] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–6. USENIX Association, 2010.

[6] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.

[7] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.

[8] B. Fathi. Engineering windows 7 : User account control, October 2008. MSDN blog on User Account Control.

[9] A. 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*, pages 627–638. ACM, 2011.

[10] A. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proc. of the USENIX Conference on Web Application Development*, 2011.

[11] 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.

[12] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 3–14, New York, NY, USA, 2011. ACM.

[13] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of install-time permission systems for third-party applications. Technical Report UCB/EECS-2010-143, EECS Department, University of California, Berkeley, Dec 2010.

[14] Y. Huang and S. Du. Weighted support vector machine for classification with uneven training class sizes. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*. IEEE, 2005.

[15] W. A. Magat, W. K. Viscusi, and J. Huber. Consumer processing of hazard warning information. *Journal of Risk and Uncertainty*, 1(2):201–32, June 1988.

[16] S. Motiee, K. Hawkey, and K. Beznosov. Do windows users follow the principle of least privilege?: investigating user account control practices. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*. ACM, 2010.

[17] M. Nauman, S. Khan, and X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 328–332. ACM, 2010.

[18] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 340–349. Ieee, 2009.

[19] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356. ACM, 2010.

[20] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang. Plagiarizing smartphone applications: Attack strategies and defense. Springer, 2012.

[21] A. Schmidt, J. Clausen, A. Camtepe, and S. Albayrak. Detecting symbian os malware through static function call analysis. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*, pages 15–22. IEEE, 2009.

[22] D. W. Stewart and I. M. Martin. Intended and unintended consequences of warning messages: A review and synthesis of empirical research. *Journal of Public Policy Marketing*, 13(1):1–19, 1994.

[23] J. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 1999.

[24] P. Szor. *The art of computer virus research and defense*. Addison-Wesley Professional, 2005.

[25] T. Vidas, N. Christin, and L. Cranor. Curbing android permission creep. In *Proceedings of the Web*, volume 2, 2011.

[26] T. Zhang. An introduction to support vector machines and other kernel-based learning methods. *AI Magazine*, 2001.