Generating Summary Risk Scores for Mobile Applications

Christopher S. Gates, Ninghui Li, *Senior Member, IEEE*, Hao Peng, Bhaskar Sarma, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, *Member, IEEE Computer Society*, and Ian Molloy

Abstract—One of Android's main defense mechanisms against malicious apps is a risk communication mechanism which, before a user installs an app, warns the user about the permissions the app requires, trusting that the user will make the right decision. This approach has been shown to be ineffective as it presents the risk information of each app in a "stand-alone" fashion and in a way that requires too much technical knowledge and time to distill useful information. We discuss the desired properties of risk signals and relative risk scores for Android apps in order to generate another metric that users can utilize when choosing apps. We present a wide range of techniques to generate both risk signals and risk scores that are based on heuristics as well as principled machine learning techniques. Experimental results conducted using real-world data sets show that these methods can effectively identify malware as very risky, are simple to understand, and easy to use.

Index Terms—Risk, mobile, malware, data mining

1 Introduction

Mobile devices are becoming ubiquitous, and they provide access to personal and sensitive information such as phone numbers, contact lists, geolocation, and SMS messages, making their security an especially important challenge. Compared with desktop and laptop computers, mobile devices have a different paradigm for installing new applications. For traditional personal computers, a typical user installs relatively few applications, most of which are from reputable vendors, with niche applications increasingly being replaced by web-based or cloud services. For mobile devices, one often downloads and uses many applications (or apps) with limited functionality from multiple unknown vendors. Therefore, the defense against malicious applications must depend to a large degree on decisions made by the users. An important part of malware defense on mobile devices is to communicate the risk of installing an app to users, and to enable the user to make informed decisions about whether to choose and install specific apps.

We study how to effectively evaluate the risk of mobile applications, with a focus on the Android platform. The Android platform has emerged as one of the fastest growing operating systems. In May 2013 Google Inc. announced that 900 million Android devices have been activated. Additionally Google Play (formerly known as Android Market) crossed more than 48 billion downloads, and is now averaging about 2.5 billion downloads per month. Such a wide user base, coupled with ease of developing and distributing applications, makes Android an attractive target for malicious developers that seek personal gain while costing users' money and invading users' privacy.

One of Android's main defense mechanisms against malicious apps is a risk communication mechanism which warns the user about the permissions an app requires before the app is installed by the user, trusting that the user will make the right decision. The specific approach used in Android has been shown to be ineffective at informing users about potential risks [8], [16], [17], [20]. The majority of Android apps request multiple permissions. When a user sees what appears to be the same warning message for almost every app, warnings quickly lose any effectiveness as the users are conditioned to ignore such warnings.

We believe that the main reason for the failure of the current Android warning approach is that it presents the risk information of each app in a "stand-alone" fashion and in a way that requires too much technical knowledge and time to distill useful information. Recently, binary risk signals based on the set of permissions an app requests have been proposed as a mechanism to improve the existing warning mechanism. In [11], requesting a certain permission or certain combinations of two or three permissions triggers a warning that the app is risky.

In this paper, we investigate permission-based risk signals that use the rarity of critical permissions and pairs of critical permissions. In this approach, initially reported in [28], 26 permissions that have significant security or privacy impact are identified as critical, and if an app requests a critical permission (or a pair of critical permissions) that is rarely requested by apps in the same category as the app, the app is labeled as risky.

Using a binary risk signal, i.e., labeling each app as either risky or not risky, however, has significant limitations. While some apps are clearly malicious, many more apps are

- *C.S. Gates, N. Li, H. Peng, B. Sarma, Y. Qi, R. Potharaju, and C. Nita-Rotaru are with the the Center for Education and Research in Information Assurance and Security, Department of Computer Science, Purdue University, West Lafayette, IN. E-mail: {gates2, ninghui, alanqi, crisn}@cs.purdue.edu, {pengh, bsarma, rpothara}@purdue.edu.*
- *I. Molloy is with IBM TJ Watson Research Center, Yorktown Heights, NY. E-mail: molloyim@us.ibm.com.*

Manuscript received 30 July 2013; revised 26 Nov. 2013; accepted 9 Jan. 2014; date of publication 22 Jan. 2014; date of current version 14 May 2014.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2014.2302293

in a gray area where their behaviors are questionable, but not overly malicious. Furthermore, whether an app is malicious or not may depend on the user's privacy preference. A binary risk signal forces the designer to draw a line somewhere, when no line may be "correct."

We thus propose the concept of risk scoring functions. Such a function assigns to each app a numerical score, which indicates how risky the app is. This approach presents "comparative" risk information, i.e., each app's risk is presented in a way so that it can be easily compared to other apps. Given a risk scoring function, one can construct a risk signal by choosing a threshold above which the signal is raised. However, we believe that it is better to use a risk scoring function for risk communication in the following way. Given this function, one can compute a risk ranking for each app, identifying the percentile of the app in terms of its risk score. This percentile number has a well-defined and easy-to-understand meaning. Users can appreciate the difference between an app ranked in the top 1 percent group versus one in the bottom 50 percent. This ranking can be presented in a more user-friendly fashion, e.g., translated into categorical values such as high risk, medium risk, low risk, and very low risk. An important feature of the mobile app ecosystem is that users often have choices and alternatives when choosing a mobile app. If the user knows that one app is significantly more risky than another with similar functionality, then that may cause the user to choose the less risky one. Such an approach complements well other approaches that try to identify malicious apps [1], [19]. After malicious apps are removed, the remaining ones can be ranked according to their risks.

In this paper, we examine two approaches to develop risk scoring functions based on permissions requested by apps. The first approach, initially reported in [25], uses probabilistic generative models [5], which have been used extensively in a variety of applications in machine learning, computer vision, and computational biology, to model complex data. One strength of this technique is that it can model features from a large amount of unlabeled data. Using these models, we assume that some parameterized random process generates the app data and learn the model parameters based on the data. Then we can compute the probability that each app was generated by the model. The risk score can be any function that is inversely related to the probability, so that lower probability translates into a higher risk score. We consider four models: Basic Naive Bayes (BNB), Naive Bayes with informative Priors (PNB), Mixture of Naive Bayes (MNB), and Hierarchical Mixture of Naive Bayes (HMNB), and explore their pros and cons. Our results suggest that the PNB model offers the best tradeoff among the desirable properties.

The second approach is motivated by the observation that the probabilistic modeling approach and the risk signal technique are closely connected. In particular, by taking logarithm of the computed probabilities, the PNB model can be naturally interpreted as a simple function capturing the rarity of permissions. We thus present a simple framework so that the PNB model and rarity-based signals are all instances in this framework. This also presents a simple interpretation of the PNB model that may be easier to understand. We then investigate two other instantiations of the framework and evaluate them.

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

- We introduce the idea of risk score functions to improve risk communication for Android apps, and identify three desiderata for an effective risk scoring function.
- We examine several Bayesian approaches for risk scoring functions, and analyze whether they satisfy the desiderata.
- We introduce a framework that includes both the rarity-based risk signals and probabilistic models, and explore other ways to instantiate the framework.
- We conduct extensive evaluations using real-world data sets. Our experimental results demonstrate the effectiveness and relative strengths and weaknesses of different approaches.

The remainder of this 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 data sets that we have collected. In Section 4 we discuss the idea of a heuristics based approach to generating a risk signal. In Section 5 we discuss different generative models for risk scoring. The rarity-based risk signal and probabilistic models are linked in Section 6. We then present experimental results and discussion in Section 7. We finish by discussing related work in Section 8 and concluding in Section 9.

2 Android Platform

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 Linux. The application framework uses the Dalvik Virtual Machine. Applications are written in Java using the Android SDK, compiled into Dalvik Executable files, and packaged into `.apk` (Android package) archives for installation.

The app store hosted by Google is called Google Play. In order to submit apps to Google Play, an Android developer first needs to obtain a publisher account. After submission, each *.apk* file gets an entry on the market in the form of a webpage, accessible to users through either the Google Play homepage or the search interface. This webpage contains meta-information that keeps track of information pertaining to the app (e.g., name, category, version, size, prices) and its usage statistics (e.g., rating, number of installs, user reviews). This information is used by users when they are deciding to install a new app.

The Android system's built-in defense against malware consists of two parts: *sandboxing* each app and *warning* the user about the permissions that the app is requesting. Specifically, each app runs with a separate user ID, as a separate process in 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, and have to explicitly request these privileges through permissions. In tandem with the sandboxing approach is a communication mechanism. When a user downloads an app through the Google Play website or

through the Google Play app on the device, the user is shown a screen that displays the permissions requested by the app. This provides a final chance to verify that the user is allowing the app access to the requested resources. Installing the app means granting the app all the requested permissions.

Google also scans all apps using Bouncer [1], which provides automated analysis of apps on Google Play to identify potentially malicious apps. Once an app is uploaded, both signature based and behavioral based malware detection is performed. Risk communication is complementary to malware detection, as the line between malicious apps and non-malicious apps is fuzzy, and the behavior of many apps cannot be classified as malicious, yet many users will find them risky and intrusive.

Other third party app websites exist, e.g., Amazon Appstore for Android, GetJar, SlideMe Market, etc. Their risk communication mechanisms are essentially the same as Google Play. While this type of interface has been shown to be ineffective [8], [15], [16], [17], [20], [21], [34], few alternatives have been proposed. We argue that a promising alternative is to present relative or comparative risk information. This way, users can select apps based on easy-to-consume risk information. This could also incentivize developers to better follow the least-privilege principle and request only necessary permissions.

3 Data

In this section, we describe the data sets we used in this paper.

3.1 Data Descriptions

Market data sets. We have collected two data sets from Google Play spaced one year apart. The data sets were collected by crawling the Google Play app store, starting with the top apps in each category and then following all the links to other apps on each page. Market2011, the first data set, consists of 157,856 apps available on Google Play in February 2011. Market2012, the second data set, consists of 324,658 apps, and has been collected in February 2012. For each app, we have the application meta-information consisting of the developer name, its category and the set of permissions that the app requests. We assume that apps in these two data sets are mostly benign. While we believe that a small number of malicious apps may be present in them, we assume that these data sets are dominated by benign ones. We leverage the Market2011 data set for our model generation and testing, and use Market2012 data set for validation and market evolution analysis.

Malware data set. Our malware data set consists of 808 unique `.apk` files that are known to be malicious. We obtained this data set from the authors of [36]. For each malware sample, we extract the permissions requested using the AndroidManifest.xml file present inside the package file. For these malicious apps we do not have their category information.

3.2 Data Cleansing

In the two market data sets, Market2011 and Market2012, we have observed the presence of thousands of apps that have similar characteristics. This kind of "duplication" can occur due to the following reasons:

- *Slight variations (R1)*. One developer may release hundreds or even thousands of nearly identical apps that provide the same functionality with slight variation. A few examples include wallpaper apps, city or country specific travel apps, weather apps, or themed apps (i.e., a new app with essentially the same functionalities can be written for any celebrity, interest group, etc).
- *App maker tools (R2)*. There are a number of tools that enable non-programmers to create Android apps. Often times many apps that are generated by these tools have similar app names and the same set of permissions. This occurs when the developer uses the default settings in the tool.

We decided to consolidate duplicate apps from the same developer (R1) into a single instance in the data set to prevent any single developer from having a large impact on the generated statistics and model. We detect apps due to R1 by looking for instances where apps belong to the same developer and have the same set of permissions.

We decided to keep apps due to R2 unchanged in the data sets. We do this because: (1) we observed instances where apps due to R2 have different functionality and many developers using these tools do modify the permissions given to their app and (2) the line between such apps and all apps that use a specific ad-network which require a certain set of permissions is blurry.

After cleansing is complete we have 71,331 apps in the 2011 market data set, and 136,534 apps in the 2012 market data set. This represents a reduction of around 55 percent, and demonstrates the prevalence of apps that are slight variations of other apps, justifying our decision to combine these so as not to allow one developer to overly influence the overall statistics.

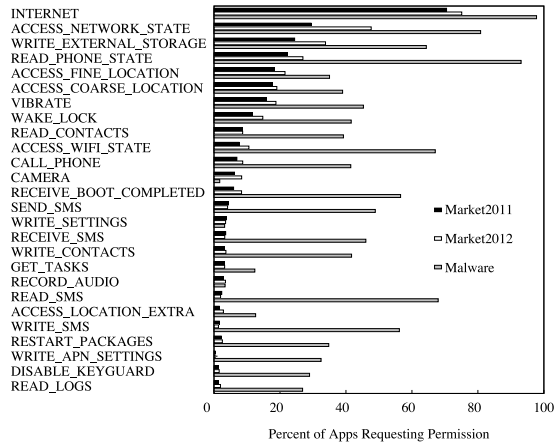
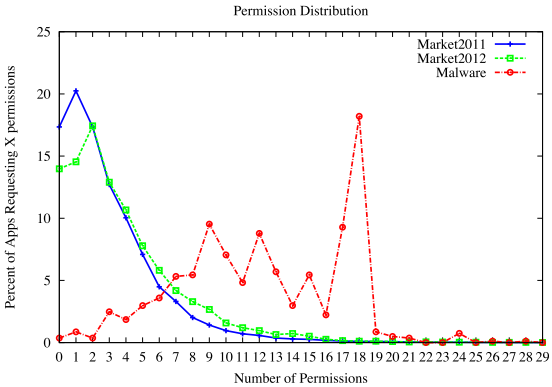


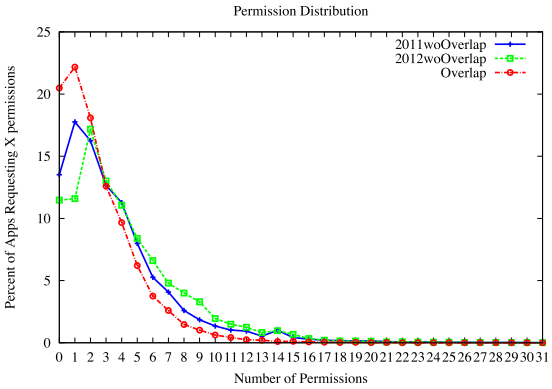
Fig. 1. The top 20 most used permissions in all three data sets as a percent of apps that request those permissions. Due to overlap, we must show 26 permissions to cover the 20 most used in all data sets.

3.3 Data Discussion

The frequencies of the most frequently requested permissions in the three data sets are presented in Fig. 1. There are 26 permissions in this figure, which include the top 20 for



(a) The percent of apps that request a specific number of permissions for each dataset.



(b) The percent of apps that request a specific number of permissions in the market datasets. Apps that appear uniquely in 2011, uniquely in 2012, and the intersection of those two datasets

Fig. 2. Permission information for various data sets.

all three data sets. Clearly, for every permission, the percentage of malware apps is significantly higher than those in the two market data sets. For some permissions, the difference is quite striking. For example, READ_SMS is requested by 67.95 percent of the malicious apps, but only 2.34 percent from Market2011, and 1.99 percent from Market2012. Furthermore, for almost every permission, a higher percent of apps in Market2012 request the permission when compared to the Market2011 data set. This shows a trend that proportionally more applications are requesting sensitive permissions. The exceptions to this are all related to SMS, where Market2012 actually saw a slight decrease for all permissions related to SMS.

Fig. 2a shows the percentages of apps that request different numbers of permissions in the three data sets. From this graph, we again observe that malicious apps are requesting significantly more permissions than apps in the market data sets. However, there are many market data set apps that request many permissions as well. Between Market2011 and Market2012, we also see a confirmation that apps are requesting a greater number of permissions on average. With proportionally fewer apps requesting zero or one permissions in Market2012. Overall, this indicates that the malicious apps are requesting permissions in different ways from normal apps, and indicates that looking at permission information is in fact useful. It also shows that there may be a slow evolution in the market data set.

Fig. 2b shows a similar graph when we divide the two market data sets into three groups: one, labeled as 'Overlap,' includes the apps that are common to both Market2011 and Market2012, and the other two are obtained by removing those in 'Overlap' from Market2011 and Market2012, respectively. Interestingly, apps in the overlap data set, which are the "longer-lived" and more stable, generally request fewer permissions than other apps that are entering and exiting the app store or changing the permissions they are requesting.

4 Risk Signals

In this section we discuss a heuristic based approach to identifying apps that are requesting certain permissions. This approach relies on statistics gathered from the market data to create more dynamic warnings.

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 is blurred since many apps are unnecessarily over-privileged [14] or may contain invasive ad networks [33]. 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. In general, we desire risk signals to have a relatively low (between 1 and 10 percent) warning rate, and a relatively 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 they 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.

TABLE 1
Table Displaying List of Critical Permissions and the Percent of Apps that Request Those Permissions in the Market2011, Market2012 and Malware Data Sets

Permission	2011	2012	Mal
ACCESS_COARSE_LOCATION	17.64	19.05	38.99
ACCESS_FINE_LOCATION	18.22	21.52	35.02
BLUETOOTH	1.08	1.48	0.87
BLUETOOTH_ADMIN	0.87	1.15	0.74
CALL_PHONE	6.94	8.70	41.46
GET_ACCOUNTS	1.98	0.12	2.48
INTERNET	70.49	75.10	97.77
(UN)MOUNT_FILESYSTEMS	0.83	1.21	10.40
NFC	0.01	0.16	0.00
PROCESS_OUTGOING_CALLS	0.95	0.94	6.56
READ_CALENDAR	1.02	1.76	0.00
READ_CONTACTS	8.49	8.69	39.23
READ_HISTORY_BOOKMARKS	0.00	1.82	0.00
READ_LOGS	1.33	1.95	26.86
READ_PHONE_STATE	22.15	26.96	93.07
READ_SMS	2.34	1.99	67.95
RECEIVE_MMS	0.33	0.31	1.61
RECEIVE_SMS	3.55	3.26	46.04
RECEIVE_WAP_PUSH	0.08	0.09	1.36
RECORD_AUDIO	2.78	3.47	3.34
SEND_SMS	4.34	4.12	48.89
WRITE_CALENDAR	0.80	1.30	0.87
WRITE_CONTACTS	3.06	3.61	41.71
WRITE_EXTERNAL_STORAGE	24.44	33.80	64.36
WRITE_HISTORY_BOOKMARKS	0.00	1.47	0.00
WRITE_SMS	1.62	1.35	56.19

All values are percents.

4.2 Permission Based Risk Signals

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 data set, and aim only at detecting apps that request too many permissions. Furthermore, it may be desirable for the signals to use only critical permissions so that such signals are more difficult to evade.

From Android's list of permissions, we choose 26 permissions that we call critical permissions. They are listed in Table 1. 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 either infringe upon privacy, or cause monetary loss or other kinds of damage.

Rare critical permissions ($\#RCP(\theta) \geq \alpha$). The first risk signal we consider is whether an app has at least α rare critical permissions. We say that a critical permission is rare with respect to a threshold θ if it occurs in less than θ percent of the Android Market applications. This signal is triggered by an app if it requests α 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 ($\#RPCP(\theta) \geq \alpha$). We consider a pair of critical permissions to be rare with respect to a threshold θ if the individual permission's frequency is above threshold θ but the frequency of occurrence of the two permissions as a pair is below θ, and we define this as $\#RPCP(\theta)$. 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 θ) but their occurrence together in an app is rare (below threshold θ).

Combination of RCP and RPCP ($\#RCP(\theta) + w * \#RPCP(\theta') \geq \alpha$). In this signal we use a linear combination of $\#RCP(\theta)$ and $\#RPCP(\theta')$ to calculate a risk score, and then choose a threshold α 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 "$\#RCP(\theta) \geq \alpha$" 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 Category and Permission Based Risk Signal

Different apps have different functionalities, and thus may require different permissions; it thus makes sense to take into account the intended functionality of an app when deriving a risk signal based on permissions. We use the category of an app to approximate the intended functionality of an app. This is partially supported by the analysis in [4], in which it was shown that in general, the category an app was in affects the permissions it requests.

Category-based rare critical permissions ($\#CRCP(\theta) \geq \alpha$). For each category, c, we call any critical permission that is requested by less than θ percent of apps in category c a Rare Critical Permission in c. Any app from category c that requests α or more rare critical permissions in c triggers the $\#CRCP(\theta)$ risk signal. The CRCP signal can be intuitively viewed as comparing the intended functionality of an app (inferred from its category) with its requested permissions and generates a report when over requesting is detected.

4.4 Results for Risk Signals

We evaluate these risk signals using the market data set and malware data set and report the results here. In this specific data set we have apps for 30 categories, some categories at the time of data collection were not yet available.

In the first row in Table 2, we observe that for a α of 2 percent, RCP has a warning rate of 7.63 percent, and a

TABLE 2
The Effect of Various Risk Signals

Risk Signal	AM%	Mal%	AM%	Mal%	AM%	Mal%	AM%	Mal%
#RCP(θ) ≥ 1	$\theta = 5$		$\theta = 2$		$\theta = 1$		$\theta = .5$	
	14.04	84.41	7.63	78.84	3.78	34.41	0.69	23.27
#RCP(2) + #RPCP(1)$\geq \alpha$	$alpha = 5$		$alpha = 2$		$alpha = 1$		$alpha = .5$	
	9.91	81.06	5.04	71.53	2.89	65.97	2.13	62.25

The first column gives the description of the risk signal. The remainder are the α values along with results for that α. AM percent refers to the percent of Android Market apps that receive the warning, Mal percent refers to malicious apps receiving the warning.

TABLE 3
Percent of Apps Triggering the CRCP(θ) $\geq$ 1 Signal in Their Own Category Compared to the Percent of Malware that Trigger the Signal for Different Number of Categories

	Android Market apps	Malicious apps (total 808)				
	Own category	All 30 cat.	At least 27 cat.	At least 25 cat.	At least 18 cat.	At least 1 cat.
$\theta = 1\%$	2.87	18.81	23.39	27.23	76.73	85.40
$\theta = 2\%$	5.82	32.67	74.50	74.75	82.67	85.40
$\theta = 3\%$	8.82	33.17	79.33	80.32	84.41	85.40
$\theta = 4\%$	11.31	57.05	80.45	84.16	84.53	85.40
$\theta = 5\%$	12.49	57.30	82.92	84.16	84.53	91.58

78.84 percent detection rate. That is, 7.63 percent apps in the market data set request a critical permission that is requested by less than 2 percent of apps, and 78.84 percent of apps in the malware data set do the same.

The second row of Table 2 demonstrates the results of this approach for $\#RCP(2) + \#RPCP(1) \geq \alpha$. Note that for $\alpha = 2$, that is, the signal is raised when an app either requests a 2 percent-rare critical permission, or a pair of critical permissions that is 1percent-rare, this method identifies 71.53 percent of malicious apps with relatively low warning rate of 5.04 percent. Kirin [11] can identify close to 54.57 percent malware apps at 4.17 percent false positive rate. $\#RCP(2) + \#RPCP(1) \geq 3$ provides higher malware warnings (65.97 percent) with lower benign app warnings (2.89 percent) in our data set.

The Category Based Risk Signals take into account the category of an app, which can indicate some of the functionality that is expected from apps within that category. 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 3 shows the evaluation results of the CRCP(θ)$\geq$ 1 signal, which is raised when an app requests a critical permission that is requested by less than θ percent of apps in the category. The table has one row for each threshold. The second column shows the warning rate for Android Market Data set 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 one categories of the Android Market. We consider the percentages of malware classified as risky in at least 25 categories as an indicator of how successful the category based Risk Signal would be in case we could determine the category of the malware accurately. We see a warning rate of 8.82 percent with a corresponding detection rate of 80.32 percent, which is comparable to RCP's 7.63 percent warning rate and 78.84 percent detection rate. In general we see similar or better results when compared to the best non-category based risk signals, but without category labels on the malware it is only a comparison of the likely outcomes, and not the actual outcomes.

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 that belongs to a particular category. Therefore it is natural for the category of these malware apps to be the similar to the original app. These new apps, in a similar category, will typically be requesting more permissions then the original and should trigger a warning. Second, to speed up propagation a malware may be uploaded to more than one categories, in some categories a warning may be raised.

5 Bayesian Risk Score

In the previous section we looked at risk signals that operate as a binary signal for some risk that might be present from an app. While this appears to work reasonably well, the binary signal approach is inherently limited. A risk score can be used to compare several different applications and understand a relative risk of installing one app versus another.

In this section we describe desirable characteristics of a risk score and then present details for several techniques to generate a risk score for Android applications based on the permissions they request. While the focus of this work is on permissions, the idea of generating a risk score can be extended to account for other features such as source code, developer information, user reviews, privacy policies and various other attributes related to an app. A permission-based approach, however, has several advantages. It is easy to understand, is compatible with the current Android risk communication approach, and it is clear what to do when a developer wants to reduce the risk score for their app.

5.1 Design Goals for Risk Score

We aim at coming up with a risk score for apps based on their requested permission sets and categories. Let the ith app in the data set be represented by $a_i = (c_i, \mathbf{x}_i = [x_{i,1}, \ldots, x_{i,M}])$, where $c_i \in C$ is the category of the ith app, M is the number of permissions, and $x_{i,m} \in \{0, 1\}$ indicates whether the ith app has the mth permission. Our goal is to come up with a risk function $\mathsf{rscore} : C \times \{0, 1\}^M \rightarrow \mathbb{R}$ such that it satisfies the following three desiderata.

We propose the following desiderata for the risk scoring function. First, it *should* be monotonic, in the sense that for any app, removing a permission from its set of requested permissions should reduce the risk score. This way, a developer can reduce the risk score of an app by following the least-privilege principle. Monotonicity in this setting is formalized by the following definition:

Definition 1 (Monotonicity). *We say that a risk scoring function* rscore *is* monotonic *if and only if for any $c_i \in C$ and any $\mathbf{x}_i, \mathbf{x}_j$ such that*

$$\exists k(\mathbf{x}_{i,k} = 0 \wedge \mathbf{x}_{j,k} = 1 \wedge \forall m(m \neq k \Rightarrow \mathbf{x}_{i,m} = \mathbf{x}_{j,m}))$$

$$\Rightarrow \mathsf{rscore}(c_i, \mathbf{x}_i) < \mathsf{rscore}(c_i, \mathbf{x}_j).$$

The second desideratum is that malicious apps generally have high risk scores. And the third is that the risk scoring function is simple to understand.

Given any risk function, we can assign a risk ranking for each app relative to a set A of reference apps, which can be, e.g., the set of all apps available in Google Play:

$$\mathsf{rrank}(a_i) = \frac{|\{a \in A | \mathsf{rscore}(a) \geq \mathsf{rscore}(a_i)\}|}{|A|}.$$

If an app has a risk ranking of 1percent, this means that the app's risk score is among the highest 1 percent.

The above gives a risk ranking relative to all apps in all categories. An alternative is to rank apps in each category separately, so that one has a risk ranking for an app relative to other apps in the same category.

Probabilistic generative models. We propose to use probabilistic generative models for risk scoring. That is, we assume that some parameterized random process generates the app data sets and learn the parameter value θ that best explains the data. Next, for each app we compute $p(a_i|\theta)$, the probability that the app's data is generated by the model.

The risk score of an app can be a function of the probability that an app was generated, such that a lower probability means a higher risk score. For example, using $\mathsf{rscore}(a_i) = -\ln p(a_i|\theta)$ satisfies the condition.

In the rest of this section we describe several generative models—from simple Naive Bayesian models, to a novel hierarchical Bayesian model. We present estimation methods to learn the parameters for these models from the data, and evaluate whether they satisfy our desiderata.

5.2 Naive Bayes Models

In the Naive Bayes models, we ignore the category information c_i; thus each app is given by $\mathbf{x}_i = [x_{i,1}, \ldots, x_{i,M}]$. We assume that each $\mathbf{x}_i$ is generated by M *independent* Bernoulli random variables, where M is the number of permissions:

$$p(\mathbf{x}_i) = \prod_{m=1}^{M} p(x_{i,m}) = \prod_{m=1}^{M} \theta_m^{x_{i,m}} (1-\theta_m)^{(1-x_{i,m})}, \qquad (1)$$

where $\theta_m \equiv p(x_{i,m} = 1)$ is the Bernoulli parameter.

To avoid overfitting in our estimation, we use a Beta prior $\mathrm{Beta}(\theta_m|a_0, b_0)$ over each Bernoulli parameter θ_m. Using this prior, the Maximum a posteriori (MAP) estimation is

$$\hat{\theta}_m = \frac{\sum_i^N x_{i,m} + a_0}{N + a_0 + b_0}, \qquad (2)$$

where N is the total number of apps for this Naive Bayes model estimation.

The basic Naive Bayes model. In the Basic Naive Bayes mode, we use *uninformative* prior and set $a_0 = b_0 = 1$, so that the Beta prior becomes a uniform distribution on [0,1]. With such an uninformative prior, $\hat{\theta}_m$ is very close to the frequency of the mth permission being requested in the data set.

The BNB model is easy to explain, satisfying the third desideratum. Furthermore, if $\theta_m < 0.5$ for every m, then the probability provided by this model satisfies the monotonicity property. Changing any $\mathbf{x}_{i,m}$ from 0 to 1 changes the probability by a factor of $\frac{\theta_m}{1-\theta_m}$, which is less than 1 when $\theta_m < 0.5$, and thus decreases the probability and increases the risk score. As there is only one permission, namely Internet, requested by over 50 percent of the apps, removing the INTERNET permission from the feature set suffices to ensure the monotonicity property. Finally, the BNB model intuitively satisfies the second desideratum, i.e., known malicious apps generally have lower generated probabilities, because as we have seen in Section 3.3, malicious apps generally request more permissions.

NB with informative priors (PNB). BNB treats all permissions equally, and a malicious app can reduce its risk by not requesting rare permissions that are not critically needed for carrying out malicious activities. We thus consider a Naive Bayes model with *informative* priors to incorporate semantic information of app permissions. Such approach is commonly used in Naive Bayes models to model knowledge not available in the data set. The desired goal is to make requesting a more critical permission to increase risk more than requesting a less critical one, even though the two permissions have similar frequencies.

To identify critical permissions, we start from a list of 26 permissions identified in [28] as critical. We remove the INTERNET permission, and add another that we believe is critical, namely INSTALL_PACKAGES. Furthermore, among the 26 permissions, we manually selected nine of them as very high risk permissions.[1]

To incorporate the semantic information in the Naive Bayes models, we use informative Beta prior distributions $\mathrm{Beta}(\theta_m|a_m, b_m)$: for the nine most risky permissions, we set $a_m = 1, b_m = 2N$ where N is the number of apps in our data set, penalizing the use of these permissions; for the 17 other risky permissions, we set $a_m = 1, b_m = N$ with penalizes them more than most permissions but less than the most risky permissions; and for the remaining permissions, we set $a_m = 1, b_m = 1$ as in the BNB model.

When compared with BNB, PNB is slightly more complex. However, it has the advantage that requesting a more critical permission results in higher risk, when compared with requesting a similarly rare but less critical permission. One key benefit of PNB is that it is more difficult for malware apps to reduce their risks by removing rare permissions that they do not need, since they likely need some of the critical permissions to carry out their malicious activities. For this reason, we prefer PNB to BNB when other things are equal.

5.3 Mixture of Naive Bayes Models

The assumption in BNB and PNB that all apps follow a simple factorized Bernoulli distribution does not appear to be very realistic. Thus, we develop more sophisticated probabilistic generative models and experimentally compare the effectiveness of BNB with these models.

We improve the Naive Bayes model by assuming each app is sampled from multiple—instead of only one—*latent* topics, each of which follows a factorized Bernoulli

1. They are ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, PROCESS_OUTGOING_CALLS, CALL_PHONE, READ_CONTACTS, WRITE_CONTACTS, READ_SMS, SEND_SMS, INSTALL_PACKAGES.

distribution. Unlike the Naive Bayes model, this mixture model allows us to use different latent topics to capture different aspects of the apps. These topics could describe fine grained classes of applications, such as geotagging apps that request LOCATION, INTERNET, and CAMERA permissions, or applications that leverage common frameworks.

Specifically, we use an unknown indicator variable $z = 1, \ldots, K$ (K is the number of latent topics) to represent which topic an app is sampled from. We assign an uninformative uniform prior over z and assume that the topic distribution is the same as the Naive Bayes model conditioned on z; that is, $p(\mathbf{x}_i|z, \boldsymbol{\theta}_z) = \prod_{m=1}^{M} p(x_{i,m}|z, \theta_{zm})$ is a factorized Bernoulli distribution where $\boldsymbol{\theta}_z = [\theta_{z1}, \ldots, \theta_{zM}]$. Let $\Theta = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k]$ denote parameters for the app distributions for all the topics. Then the probability of the data is

$$p(\mathbf{x}|\Theta) = \sum_z p(z) \prod_{i=1}^{N} p(\mathbf{x}_i|z, \boldsymbol{\theta}_z), \tag{3}$$

which is a mixture of Naive Bayes models.

To obtain the MAP estimation of both assignments, we use an expectation maximization approach that loops over two steps, Expectation (E) and Maximization (M) steps, until convergence. In the E step, we compute the posterior of z given the current estimate of Θ:

$$p(z = k|\mathbf{x}, \Theta) = \frac{\prod_m \theta_{k,m}^{\sum_{i=1}^{N} x_{i,m}} (1 - \theta_{k,m})^{N - \sum_{i=1}^{N} x_{i,m}}}{\sum_k \prod_m \theta_{k,m}^{\sum_{i=1}^{N} x_{i,m}} (1 - \theta_{k,m})^{N - \sum_{i=1}^{N} x_{i,m}}}.$$

In the M step, we maximize the expected joint probability $Q = \sum_{i=1}^{N} \mathbf{E}_z[\ln p(x_i|z, \Theta) + \ln p(z) + \ln p(\Theta)]$. Note that we use the updated $p(z = k|\mathbf{x}, \Theta)$ in the E step to obtain the expectation. We thus obtain

$$\theta_{km} = \frac{\sum_i^N p(z = k|\mathbf{x}, \Theta) x_{i,m} + a_0}{\sum_i^N p(z = k|\mathbf{x}, \Theta) + a_0 + b_0}. \tag{4}$$

MNB models, however, no longer guarantee the monotonicity property. We have observed that the learned hidden topics can request certain permissions with probability over 0.5, resulting in the estimated θ_{km} being greater than 0.5. When this happens, the monotonicity property does not hold.

Mixture of Naive Bayes with categories (MNBC). We also extend MNB to consider category information and call the resulting models Mixture of Naive Bayes with Categories. In MNBC, the latent topics are shared among all categories, but each category has a different multinomial distribution describing how likely an app in this category is from a particular latent topic.

5.4 Hierarchical Mixture of Naive Bayes Models

Finally, we develop Bayesian hierarchical mixture models that we can train using apps across all categories and, at the same time, account for the difference between categories. We still produce a mixture model for each category. To share information between categories we set the latent topics to be the same across categories and sample the probabilities of choosing these topics from a common Dirichlet distribution—thus these probabilities (i.e., mixture weights) are similar. Our model extends Latent Dirichlet Allocation (LDA) models [6], a popular document model, to the case of binary vector observations (each app corresponds to a word in a document and each category is a document in the latent Dirichlet allocation models).

Let us succinctly denote the permissions of app i in category c by $\mathbf{x}_{ci}$, the parameter in the multinomial topic distribution for category c by ψ_c, the topic assignment variable for each app i in category c by z_{ci}, and the hyper-parameter of the Dirichlet prior on the topic distribution by α. Then formally speaking, we have the following stochastic data generation process:

1. For each topic k and permission m, draw the app probabilities $\theta_{k,m} \sim \mathrm{Beta}(a_0, b_0)$.
2. For each category c, sample the parameter for topic distributions $\psi_c \sim \mathrm{Dir}(\alpha)$.
3. For each app i in category c,
   a. Sample the topic assignment $z_{ci} \sim \mathrm{Multi}(\psi_c)$.
   b. Generate the permissions via the factorized Bernoulli distribution (let $z_{ci} = k$)
   $$\mathbf{x}_k \sim \prod_m \theta_{k,m}^{\sum_{i=1}^{N} x_{im}} (1 - \theta_{k,m})^{N - \sum_{i=1}^{N} x_{im}}.$$

To estimate this Bayesian model, we develop a variational algorithm. It enables us to accurately approximate the exact Bayesian posterior distributions of the model parameters with a low computational cost. We give the detailed variational update in the remainder of this section.

The posterior distribution of the hidden variables is

$$p(\psi, z|x, \alpha, \theta) = \frac{p(\psi, z, x|\alpha, \theta)}{p(x|\alpha, \theta)}. \tag{5}$$

The computation of the *exact* posterior distribution is, however, intractable. Thus, we approximate the posterior distribution by

$$q(\psi, z|\beta, r) = \prod_{c=1}^{C} q(\psi_c|\beta_c) \prod_{n=1}^{N_c} q(z_{c,n}|r_{c,n}). \tag{6}$$

To obtain an accurate approximation, we use a variational Bayes approach. Specifically, we minimize the KL divergence of p and q via the following iterative variational updates.

Update r:

$$\rho_{c,n,k} = exp\left\{ F(\beta_{c,k}) - F\left(\sum_{k=1}^{K} \beta_{c,k} \right) \right\} \cdot \prod_{m=0}^{M} \theta_{k,l}^{x_{c,n,l}} (1 - \theta_{k,l})^{1 - x_{c,n,l}} \tag{7}$$

$$r_{c,n,k} = \frac{\rho_{c,n,k}}{\sum_{k=1}^{K} \rho_{c,n,k}}. \tag{8}$$

Update β:

$$\beta_{c,k} = \alpha_k + \sum_{n=1}^{N_c} r_{c,n,k}. \tag{9}$$

Update θ:

$$\theta_{k,m} = \frac{a_{0k,l} + \sum_{c=1}^{C}\sum_{n=1}^{N_c} r_{c,n,k} x_{c,n,m}}{a_{0k,l} + b_{0k,l} + \sum_{c=1}^{C}\sum_{n=1}^{N_c} r_{c,n,k}}. \quad (10)$$

Update α via Newton's method:

$$g_k = C\left[\digamma\left(\sum_{k=1}^{K}\alpha_k\right) - \digamma(\alpha_k)\right] + \sum_{c=1}^{C}\left[\digamma\left(\sum_{k=1}^{K}\beta_{c,k}\right) - \digamma(\beta_{c,k})\right], \quad (11)$$

$$q_k = -C\digamma'(\alpha_k), \quad (12)$$

$$z = C\digamma'\left(\sum_{k=1}^{K}\alpha_k\right), \quad (13)$$

$$b = \frac{\sum_{k=1}^{K} g_k/q_k}{1/z + \sum_{k=1}^{K} 1/q_k}, \quad (14)$$

$$\alpha_k^{new} = \alpha_k^{old} - \frac{g_k - b}{q_k}. \quad (15)$$

The $\digamma(.)$ denotes the digamma function.

These updates are repeated until convergence or a predefined number of iterations have occurred. The output is then approximate values for the HMNB model parameters which best describe the given data.

6 Simple Risk Scoring Method

We now show that risk ranking functions based on BNB and PNB can be viewed as using permission rarity to rank apps, and is thus closely related to risk signals discussed in Section 4.

When risk scores are used to rank apps, using the probabilities (where lower probability means higher risk) is equivalent to using the negative logarithm of the probabilities (where larger value mean higher risk). For the BNB model, we have the equivalent risk ranking function

$$\begin{aligned} f(\mathbf{x}_i) &= -\ln p(\mathbf{x}_i) = \sum_{m=1}^{M} -\ln p(x_{i,m}) \\ &= \sum_{m=1}^{M} -\ln\left[\left(\frac{c_m+1}{N+2}\right)^{x_{i,m}}\left(\frac{N+1-c_m}{N+2}\right)^{(1-x_{i,m})}\right] \\ &= M\ln(N+2) - \sum_{m=1}^{M}[x_{i,m}\ln(c_m+1) \\ &\quad + (1-x_{i,m})\ln(N+1-c_m)], \end{aligned}$$

where N is the total number of apps in the training data, $x_{i,m} = 1$ if the ith app has the mth permission and $x_{i,m} = 0$ otherwise, $c_m = \sum_i^N x_{i,m}$.

For the purpose of ranking, we can ignore the constant term, and observe that possessing one permission increases the calculated risk value by $\ln(N+1-c_m) - \ln(c_m+1)$, which is approximately $\ln(\frac{N}{c_m})$ when $1 \ll c_m \ll N$. As the majority of permissions are requested by less than 10 percent of the apps, this design penalizes requesting more permissions more than requesting a very rare permission. For example, possessing two permissions each requested by 5 percent of apps contributes $2\ln 20$, whereas possessing one permission requested by 1 percent of apps contributes $\ln 100 < 2\ln 20$.

PNB can be interpreted similarly. Possessing a permission increases the log-based risk value by approximately

$$\ln\left(\frac{w_m N - c_m}{c_m}\right) = \ln\left(N - \frac{c_m}{w_m}\right) + \ln w_m - \ln c_m,$$

where w_m is the weight used for assigning the prior to the permissions, with $w_m = 3$ for the most risky permissions, $w_m = 2$ for other risky permissions, and $w_m = 1$ for other permissions. Note that when c_m is small relative to N, then $\ln(N - c_m)$ and $\ln(N - \frac{c_m}{w_m})$ are very close, thus possessing a most risky permission results in only a penalty of $\ln w_m$, which is quite small. For example, possessing three non-critical permissions each requested by 5 percent of apps contributes $3\ln 20$, whereas possessing two most critical permissions of the same rarity contributes $2(\ln 20 + \ln 3) < 3\ln 20$.

Recognizing that BNB and PNB are essentially using rarity of permissions leads us to the following risk functions.

6.1 Rarity Based Risk Score (RS)

The first method we propose is the *R*arity Based Risk *S*core which is based strictly on the fraction of applications which are requesting a specific permission. This is closely linked to the ideas from Section 4 where the rarity of permissions is the primary indicator that contributes to raising a warning for an app. However, instead of considering only the rarest permission, we accumulate risk across all permissions that the app requests.

$$score(\mathbf{x}_i) = \sum_{m=1}^{M} x_{i,m} \cdot \ln\left(\frac{N}{c_m}\right). \quad (16)$$

In this formulation, the higher the score, the more risky the application is. This formulation also only considers permissions that are set when calculating the risk score, unlike the Bayesian methods which will affect the score for both set and unset permissions.

6.2 Rarity Based Risk Score with Scaling (RSS)

The next method we propose is *R*arity Based Risk *S*core with *S*caling, which allows the use of scaling factor to penalize requesting high risk permissions more then requesting lower risk permissions. While the PNB method could have more dramatic impact using the prior, the beta values used to define the prior would have grown exponentially to have significant impact on the outcomes.

$$score(\mathbf{x}_i) = \sum_{m=1}^{M} x_m \cdot w_m \cdot \ln\left(\frac{N}{c_m}\right). \quad (17)$$

Using this equation we are able to scale the importance of a permission relative to its risk. So a medium risk permission can have $w_m = 2$ times the impact on the overall score, and a high risk permissions can have $w_m = 3$ times the impact on the overall score. For the evaluation we use w_m values that reflect the values from PNB for each permission. Further fine-tuning the w_m values is possible. However, we consider it undesirable to tune w_m based on the malware data set because of potential for overfitting. In any case, this would make any comparison with the probabilistic generative models unfair, since they were constructed using only the market data sets.

7 Experimental Results

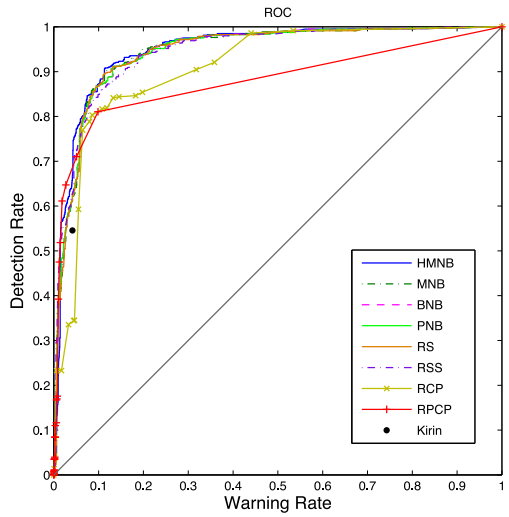
In the experiments we aim at understanding how well the different techniques satisfy the second desideratum, namely, able to assigning high risks to known malware apps, and compare them to results from Section 4 as well as methods in the literature [11].

Methodology. Most of our experiments are conducted with the 2011 data set with 10 fold cross validation. We divide the 2011 data set randomly into 10 groups. In each of the 10 rounds, we choose one different group as the test data set, and the remaining nine groups as the training data set. The models or statistics are trained on the training set, the generated model/statistic is used to compute the probabilities/scores of apps in the testing set and the malware data set, and rank them together.

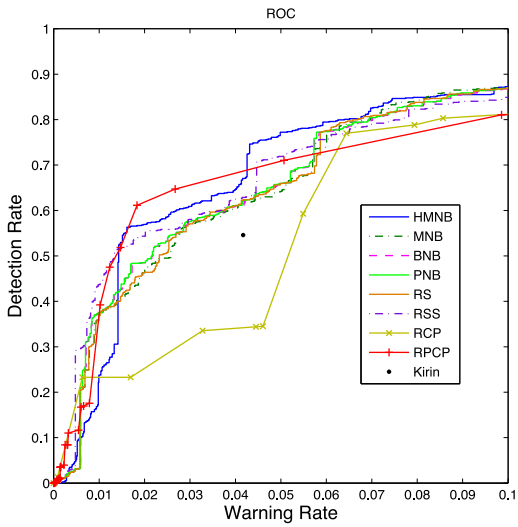
When reporting the results, we use ROC curves, which plot the detection rate against warning rate if one chooses a particular risk value as indicative of malicious app. We also compute Area Under Curve (AUC), which quantifies the quality of the ROC curves for a method. Here, AUC is the probability that a randomly selected malicious application will have a higher risk score than a randomly selected benign application. AUC values, however, can be misleading. Knowing what percentage of malicious apps are among the top x percent high-risk apps for $x > 20$ is of little interest, since there are so many benign apps that have similar risks. This information, however, is more useful for smaller x values. We therefore also compute AUC for the X-axis (warning rate) of up to 5 and 10 percent, and use AUC_5 and AUC_{10} to denote them. They evaluate what percentage of malicious apps will be rated among the apps with the highest risks under a ranking function.

Parameter selection. For the Bayesian methods, both MNB and HMNB can be used with different number of hidden topics. From our experiments, we find that the maximum AUC for MNB is achieved with five hidden classes. And the maximum for HMNB is achieved with 80 hidden classes. We use these parameters when comparing with other methods. For a more complete discussion of the parameter selection, including graphs, we refer you to [25].

Furthermore, both MNB and HMNB require the category information, which malware apps do not have. When assigning a category to malware apps, we use a method that gives the most advantage to malware. More specifically, we compute the probability of each malicious app for every category and assigns the app to the category in which the app has the highest probability.



(a) ROC for the best performing parameters for each method.

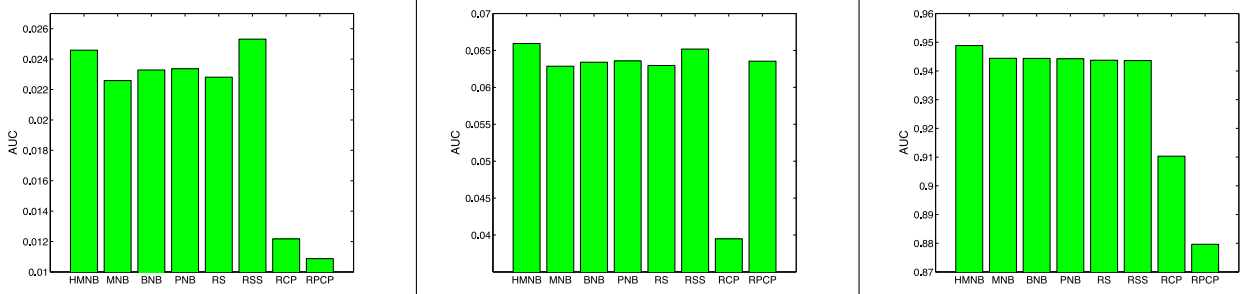


(b) Close up of Figure 3(a) to capture performance differences in the first 10% warning rates

Fig. 3. Comparison of different models using the best performing parameters for each models.

Comparing different methods. In Fig. 3, we compare methods in this paper and Kirin [11]. Fig. 3a shows the full ROC curves. Because several curves are clustered together, we use Fig. 3b to show a close-up of the ROC curves for x-axis of up to 0.1. Kirin is represented by a static set of rules, therefore we only illustrate it as a single point in the figures, and that is why it has no AUC value. It can identify 54.57 percent malware apps at 4.17 percent warning rate. From the ROC curves, we see that the probabilistic models and RS and RSS perform similarly when viewing across all warning rates. For low warning rates, however, there are some differences. RPCP has higher detection rates for low warning rates, but the detection rate improves slowly. RCP performs strictly worse than the probabilistic models.

Fig. 4 demonstrate the AUC_5, AUC_{10} and full AUC values for all methods. When looking at the final AUC values in Fig. 4c HMNB is best performing, with MNB, BNB, PNB, RS and RSS close behind and almost equivalent, and RCP and RPCP significantly underperform other methods. We note that even a difference of 0.01 is statistically significant



(a) The AUC_5 values for each method. This shows the area under the curve for the warning rate up to 5%.

(b) The AUC_{10} values for each method. This shows the area under the curve for the warning rate up to 10%

(c) The AUC values for the full ROC curves presented in Figure 3a

Fig. 4. These histograms shows the AUC value comparisons of Fig. 3a at various Warning Rate cutoffs.

given the small standard deviation found in our 10x cross fold validation.

When looking at AUC_5 and AUC_{10}, we see a slightly different picture. First, RSS outperforms other methods in Fig. 4a and is comparable to the most complex model, HMNB, in Fig. 4b. BNB, PNB, MNB, and RS all perform similar. RPCP performs well under AUC_{10}, though the worst for AUC_5.

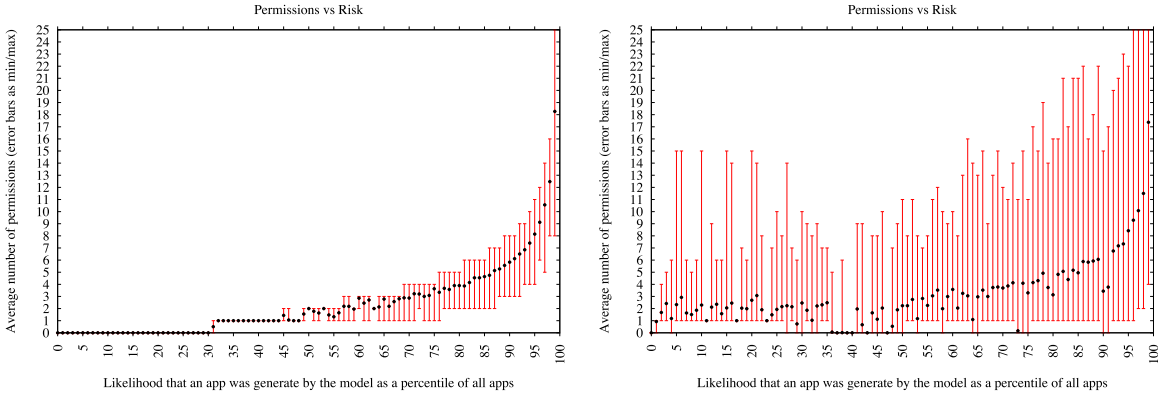
Monotonicity. HMNB performs consistently well for all AUC values we look at; this makes it attractive as a risk scoring method. However, one drawback of HMNB is that it does not have the monotonicity property. In Fig. 5 we plot the average number of permissions for each percentile of the apps in the Market2011 data set, when they are ranked by the risk value according to the PNB model and to the HMNB model. It is clearly seen that in the PNB model the average number of permissions is almost nondecreasing as the risk goes up. On the other hand, in the HMNB model we observe apps with large number of permissions that have low risk. This suggests that HMNB flatly fails the monotonicity requirement.

Model stability. Finally, we conducted experiments to check whether models trained on one data set can be used without retraining to compute the risk scoring on a new data set. For this purpose, we use the divided data sets described in Section 3. That is the overlap data between 2011 and 2012, and the 2011 data set with overlap removed and 2012 data set with overlap removed.

For each of the six possible ordered pairs, we train on one data set and then test on the other together with the malware data set. Fig. 6 shows the result. Somewhat interestingly, when testing on the overlap data set, training either on the 2011woOverlap data set or the 2012woOverlap data set gives excellent result. However using any other combination leads to results that perform worse. This is to some degree to be expected from Fig. 2b. As the "overlap" apps generally request fewer permissions than the other two data sets. The other apps appear to be more varied and require training using part of them to get good results.

As we have seen in Fig. 2 the permission data has changed over time. Therefore, if a system like this were to be implemented, the models should be periodically regenerated to achieve the best results and to keep up to date with the trends that are occurring within the market.

Discussions. We believe that the RSS approach is most attractive. It performs the best for AUC_5, and just slightly worse than HMNB for AUC_{10}. It is easy to understand and does not require category information. And it offers slightly more intuitive customizations then the generative models. Users who care twice as much about being tracked by advertisers can increase the scaling factors on permissions related to geo-location, users who care more about apps



(a) PNB (b) HMNB80

Fig. 5. Average number of permissions for every 1 percent percent division of apps, sorted in descending order on the basis of likelihood. The points represents the average number of permissions requested, and the error bars indicate the min and max at that percentile.

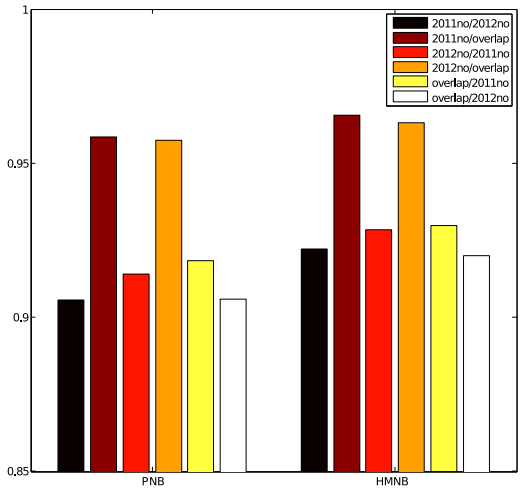


Fig. 6. Comparison of 2011 and 2012 Data for PNB and HMNB models. 'no' = no overlap, 'over' = only overlap. 'first/second' means the first data set was used to train, and the second data set was used to test along with the malware. The AUC value is presented under each scenario.

stealing their contact lists can give a higher weight to READ_CONTACTS. This allows for simpler feedback mechanisms, and more easily customized risk scores.

While we desire higher detection rate for malware, one should be cautious when interpreting this rate in our results. We are using a data set of 808 malware apps. It is important to note that these malware apps were written when over-provisioning permissions was not punished. If permissions had greater impact on what users were willing to install, malware authors may choose to request only the permissions absolutely necessary for the specific malicious task. In that case, the ROC curve for different methods would change. However, such permission-based risk ranking functions can still rank apps' permission-based risk.

8 Related Work

8.1 Permissions and Usefulness

There is a body of work showing the difficulty of using Android permissions in their current form. Felt et al. [12] 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 Felt et al. [13] surveys applications (free and paid) from the Android Market. Their key observation was that 93 percent of free apps and 82 percent 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. Chin et al. [8] conduct a user study which shows that users are more concerned with privacy on their phones then on their computers. In contrast, users tend to not focus on permission during app browsing and installation [17], [20]. Users tend to rely on user ratings and reviews when making judgments on an app, but this approach is not always reliable. The work in this paper looks at effective new indicators that could be used to help give users a deeper insight into an app without burdening them with too many details and technical jargon that they will not understand.

PScout [3] performs static code analysis on the Android source to extract function to permission mappings. They find that the Android permission system has little redundancy and it remains relatively stable as the Android OS evolves. They also show how many functions require specific permissions, demonstrating the complexity of the system and that a permission may have many reasons for being requested. So while the permission system is complex, the findings of this work show that it is at least useful at least from a system perspective.

Barrera et al. [4] 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 one thousand applications and report results on permission use and how usage relates to categories.

Enck et al. [10] make an effort to decompile and analyze the source of applications to detect further leaks and usage of data via static code analysis. Another work by Enck et al. [11] which we have already discussed, 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 data set of apps to identify potentially malicious apps within that set.

MAST [7] identifies the applications that are most likely malware by extracting features from the source code of an app, including permissions and intents. These features are then input into Multiple Correspondence Analysis (MCA) which is a technique that takes in categorical features and labels and comes up with the best way to explain the data. Using the generated model on new and unknown data, they are able to focus their efforts on the apps that are most likely malware and show that in large scale markets this can be an effective way to utilize limited resources.

8.2 Android Security

Dynamic analysis. Another research direction in Android security is to use dynamic analysis. Portokalidis [26] 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. [9] 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 [35] that enforce principle of least privilege, extending permission models and defining user-defined runtime constraints [23], [24] to limit application access and detecting applications with a malicious intent [9], [27].

Nauman et al. [23] 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 et al. [24] present an infrastructure that governs install-time permission assignment and their runtime 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 et al. [35] presents 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.

8.3 Machine Learning in Security

Naive Bayes has been extensively used both in the context of spam detection [18], [22], [29], [32] and anomaly detection [2], [30] in network traffic flows. In the context of Android, however, there has been limited work. Shabtai and Elovici [31] presents a behavioral-based detection framework for Android that realizes a host-based intrusion detection system that monitors events originating from the device and classifies them as normal or abnormal. Our work differs in that we use machine learning for the purpose of risk communication.

9 Conclusions

We discuss the importance of effectively communicating the risk of an application to users, and propose several methods to rate this risk. We test these methods on large real-world data sets to understand each method's ability to assign risk to applications. One effective method is the RSS method which has several advantages. It is monotonic, and can provide feedback as to why risk is high for a specific app and how a developer could reduce that risk. It performs well in identifying most current malware apps as high risk. This method allows for highly critical permissions and less-critical permissions to affect the overall score in an easy to understand way, making it more intuitive as well as difficult to evade when compared with other models.

Acknowledgments

We would like to thank Xuxian Jiang and Yajin Zhou who provided us with their collection of Android malware samples. Work by C. Gates, B. Sarma, N. Li were supported by Army Research Office Award 2008-0845-04 through North Carolina State University, and by the National Science Foundation under Grant No. 1314688. H. Peng and Y. Qi were supported by NSF IIS-0916443, NSF CAREER award IIS-1054903, and the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370. Work by R. Potharaju and C. Nita-Rotaru were supported by NSF TC 0915655-CNS.

References

[1] Google Bouncer. http://goo.gl/QnC6G 2014.
[2] N. Amor, S. Benferhat, and Z. Elouedi, "Naive Bayes versus Decision Trees in Intrusion Detection Systems," *Proc. ACM Symp. Applied Computing*, pp. 420-424, 2004.
[3] K.W.Y. Au, Y.F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permission Specification," *Proc. ACM Conf. Computer and Comm. Security (CCS '12)*, pp. 217-228, 2012.
[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," *Proc. 17th ACM Conf. Computer and Comm. Security*, pp. 73-84, 2010.
[5] C.M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.
[6] D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent Dirichlet Allocation," *J. Machine Learning Research*, vol. 3, pp. 993-1022, 2003.
[7] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST: Triage for Market-Scale Mobile Malware Analysis," *Proc. Sixth ACM Conf. Security and Privacy in Wireless and Mobile Networks (WISEC '13)*, pp. 13-24, 2013.
[8] E. Chin, A.P. Felt, V. Sekar, and D. Wagner, "Measuring User Confidence in Smartphone Security and Privacy," *Proc. Eighth Symp. Usable Privacy and Security*, (SOUPS '12), article 1, 2012.
[9] W. Enck, P. Gilbert, B. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Proc. Ninth USENIX Conf. Operating Systems Design and Implementation*, article 1-6, 2010.
[10] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security," *Proc. 20th USENIX Conf. Security*, (SEC '11), pp. 21-21, 2011.
[11] W. Enck, M. Ongtang, and P. McDaniel, "On Lightweight Mobile Phone Application Certification," *Proc. 16th ACM Conf. Computer and Comm. Security*, (CCS '09), pp. 235-245, 2009.
[12] A.P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," *Proc. 18th ACM Conf. Computer and Comm. Security*, pp. 627-638, 2011.
[13] A.P. Felt, K. Greenwood, and D. Wagner, "The Effectiveness of Application Permissions," *Proc. Second USENIX Conf. Web Application Development*, 2011.
[14] A.P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," *Proc. 18th ACM Conf. Computer and Comm. Security*, (CCS '11), pp. 627-638, 2011.
[15] 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, Univ. of California, Berkeley, Dec. 2010.
[16] A.P. Felt, K. Greenwood, and D. Wagner, "The Effectiveness of Application Permissions," *Proc. Second USENIX Conf. Web Application Development*, (WebApps '11), 2011.
[17] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," *Proc. Eighth Symp. Usable Privacy and Security*, article 3, 2012.
[18] J. Goodman and W. Yih, "Online Discriminative Spam Filter Training," *Proc. Third Conf. Email and Anti-Spam (CEAS '06)*, 2006.

[19] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and Accurate Zero-Day Android Malware Detection," *Proc. 10th Int'l Conf. Mobile Systems, Applications, and Services*, (MobiSys '12), pp. 281-294, 2012.
[20] P.G. Kelley, S. Consolvo, L.F. Cranor, J. Jung, N. Sadeh, and D. Wetherall, "A Conundrum of Permissions: Installing Applications on an Android Smartphone," *Proc. Workshop Usable Security (USEC '12)*, Feb. 2012.
[21] W.A. Magat, W.K. Viscusi, and J. Huber, "Consumer Processing of Hazard Warning Information," *J. Risk and Uncertainty*, vol. 1, no. 2, pp. 201-32, June 1988.
[22] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam Filtering with Naive Bayes-Which Naive Bayes," *Proc. Third Conf. Email and Anti-Spam (CEAS '06)*, vol. 17, pp. 28-69, 2006.
[23] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android Permission Model and Enforcement with User-Defined Runtime Constraints," *Proc. Fifth ACM Symp. Information, Computer and Comm. Security*, pp. 328-332, 2010.
[24] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically Rich Application-Centric Security in Android," *Proc. IEEE Ann. Conf. Computer Security Applications (ACSAC '09)*, pp. 340-349, 2009.
[25] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using Probabilistic Generative Models for Ranking Risks of Android Apps," *Proc. Conf. Computer and Comm. Security*, (CCS '12), pp. 241-252, 2012.
[26] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Versatile Protection for Smartphones," *Proc. 26th Ann. Conf. Computer Security Applications*, pp. 347-356, 2010.
[27] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, "Plagiarizing Smartphone Applications: Attack Strategies and Defense," *Proc. Fourth Int'l Conf. Eng. Secure Software and Systems*, pp. 106-120, 2012.
[28] B.P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android Permissions: A Perspective Combining Risks and Benefits," *Proc. 17th ACM Symp. Access Control Models and Technologies (SACMAT '12)*, 2012.
[29] K. Schneider, "A Comparison of Event Models for Naive Bayes Anti-Spam E-Mail Filtering," *Proc. 10th Conf. European Chapter of the Assoc. for Computational Linguistics*, vol. 1, pp. 307-314, 2003.
[30] A. Sebyala, T. Olukemi, and L. Sacks, "Active Platform Security through Intrusion Detection Using Naive Bayesian Network for Anomaly Detection," *Proc. London Comm. Symp.*, 2002.
[31] A. Shabtai and Y. Elovici, "Applying Behavioral Detection on Android-Based Devices," *Proc. Mobile Wireless Middleware, Operating Systems, and Applications*, pp. 235-249, 2010.
[32] Y. Song, A. Kocz, and C.L. Giles, "Better Naive Bayes Classication for High-Precision Spam Detection," *Software Practice and Experience*, vol. 39, no. 11, pp. 1003-1024, 2009.
[33] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating User Privacy in Android Ad Libraries," *Proc. IEEE Mobile Security Technologies (MoST '12)*, 2012.
[34] D.W. Stewart and I.M. Martin, "Intended and Unintended Consequences of Warning Messages: A Review and Synthesis of Empirical Research," *J. Public Policy & Marketing*, vol. 13, no. 1, pp. 1-19, 1994.
[35] T. Vidas, N. Christin, and L.F. Cranor, "Curbing Android Permission Creep," *Proc. Workshop Web 2.0 Security and Privacy*, vol. 2, 2011.
[36] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *Proc. 33rd IEEE Symp. Security and Privacy*, May 2012.

Christopher S. Gates received the BS degree in computer science and mathematics, and the MS degree in computer science from Rutgers University in 2002 and 2005, respectively. After this, he worked in industry for several years until 2009 when he returned to academia to pursue the PhD degree in computer science at Purdue University.

Ninghui Li received the BEng degree in computer science from the University of Science and Technology of China in 1993, and the MSc and PhD degrees in computer science from New York University, in 1998 and 2000, respectively. He is currently an associate professor of computer science at Purdue University. His research interests include security and privacy in information systems. He is a senior member of the IEEE, and an ACM distinguished scientist.

Hao Peng received the bachelor's degree in computer science from the Chinese University of Hong Kong in 2011. He is currently working toward the PhD degree in computer science at Purdue University. His research interests include machine learning and data mining.

Bhaskar Sarma received the BS degree from the Indian Institute of Technology at Guwahati in 2011 and the master's degree in computer science from Purdue University in 2013. He currently works as a security researcher in industry.

Yuan Qi received the PhD degree from MIT in 2005, where he worked as a postdoctoral researcher from 2005 to 2007. He is currently an associate professor of computer science and statistics at Purdue University. He received the A. Richard Newton Breakthrough Research Award from Microsoft Research in 2008, the Interdisciplinary Award from Purdue University in 2010, and the US National Science Foundation (NSF) CAREER Award in 2011.

Rahul Potharaju is currently working toward the PhD degree in the Department of Computer Science at Purdue University. He has more than three years of industrial research experience working with Microsoft Research, Redmond. His research interests include analyzing the reliability aspects of data centers from both hardware and software perspective.

Cristina Nita-Rotaru received the BS and MS degrees from the Politechnica University of Bucharest, Romania, in 1995 and 1996, respectively, and the PhD degree in computer science from The Johns Hopkins University in 2003. She is currently an associate professor in the Department of Computer Science at Purdue University. She leads the Dependable and Secure Distributed Systems Laboratory. She served on the technical program committee of more than 40 conferences in networking, distributed systems, and security. Her research interests include security and fault-tolerance for distributed systems and networks. She received the US National Science Foundation (NSF) Career Award. She is a member of the IEEE Computer Society and the ACM.

Ian Molloy received the PhD degree in 2010 working under Dr. Ninghui Li. He is currently a research staff member in the Information Security Group at IBM TJ Watson Research Center. His research interests include the application of machine learning and data mining to problems in identity management and insider threats, privacy, access control, applied cryptography, and general problems in security.