Towards Automated Risk Assessment and Mitigation of Mobile Applications

Yiming Jing, Gail-Joon Ahn, *Senior Member, IEEE*, Ziming Zhao, *Student Member, IEEE*, and Hongxin Hu, *Member, IEEE*

Abstract—Mobile operating systems, such as Apple's iOS and Google's Android, have supported a ballooning market of feature-rich mobile applications. However, helping users understand and mitigate security risks of mobile applications is still an ongoing challenge. While recent work has developed various techniques to reveal suspicious behaviors of mobile applications, there exists little work to answer the following question: *are those behaviors necessarily inappropriate*? In this paper, we seek an approach to cope with such a challenge and present a continuous and automated risk assessment framework called RiskMon that uses machine-learned ranking to assess risks incurred by users' mobile applications, especially Android applications. RiskMon combines users' coarse expectations and runtime behaviors of trusted applications to generate a risk assessment baseline that captures appropriate behaviors of applications. With the baseline, RiskMon assigns a risk score on every access attempt on sensitive information and ranks applications by their cumulative risk scores. Furthermore, we demonstrate how RiskMon supports risk mitigation with automated permission revocation. We also discuss a proof-of-concept implementation of RiskMon as an extension of the Android mobile platform and provide both system evaluation and usability study of our methodology.

Index Terms—Smartphones, android, risk assessment, risk mitigation

1 Introduction

Mobile operating systems, such as Android and iOS, have tremendously supported an application market over the last few years [2], [3]. Such a new paradigm drives developers to produce feature-rich applications that seamlessly cater towards users' growing needs of processing their personal information such as contacts, locations and other credentials on their mobile devices. Unfortunately, the large installed base has also attracted attention of unscrupulous developers who are interested in users' sensitive information. For example, spyware tracks users' locations and reports to remote controllers, and adware collects users' identities for enforcing an aggressive directed marketing.

To defend against such rogue applications, Android assists users to review them at install time. Primarily, Android relies on permissions to help users understand the security and privacy risks of applications. In Android, an application must request permissions to be allowed to access sensitive resources. In other words, it is mandatory for Android applications to present its expected behaviors to users. Even though permissions outline the resources that an application attempts to access, they do not provide fine-grained information about how such resources will be used. Suppose a user installs an application and allows it to access her location information. It is hard for her to determine whether the application accesses her locations on her demand or periodically without asking for her explicit consent. Therefore, it is imperative to continuously monitor the installed applications so that a user could be informed when rogue applications abuse her sensitive information. Previous work has proposed real-time monitoring to reveal potential misbehaviors of third-party applications [4], [5], [6], [7]. While these techniques partially provide valuable insights into a user's installed applications, it is still critical to answer the following challenge: *are the behaviors in mobile applications necessarily inappropriate*?

To answer this question, it is an end-user's responsibility to conduct risk assessment and make decisions based on her disposition and perception. Risk assessment is not a trivial task because it requires the user to digest diverse contextual and technical information. In addition, the user needs to apprehend *expected behaviors* of applications under different contexts prior to addressing her risk assessment baseline. However, it is impractical for the normal users to distill such a baseline. Instead, it is essential to develop an automated approach to continuously monitor applications and effectively alert users upon security and privacy violations.

In this paper, we propose an automated and continuous risk assessment framework for mobile platforms, called RiskMon. RiskMon requires a user's coarse expectations for different types of applications while user intervention is not required for the subsequent risk assessment. To this end, RiskMon leverages a one-time initialization process where

- *Y. Jing and Z. Zhao are with the Security Engineering for Future Computing (SEFCOM) Laboratory, and the Ira A. Fulton School of Engineering, Arizona State University, Tempe, AZ 85287. E-mail: {ymjing, zmzhao}@asu.edu.*
- *G.-J. Ahn is with the Security Engineering for Future Computing (SEFCOM) Laboratory, the Ira A. Fulton School of Engineering, Arizona State University, Tempe, AZ 85287, and GFS Technology, Inc. E-mail: gahn@asu.edu.*
- *H. Hu is with the Division of Computer Science, School of Computing, Clemson University, Clemson, SC 29634. E-mail: hongxih@clemson.edu.*

Manuscript received 2 May 2014; revised 16 Sept. 2014; accepted 17 Sept. 2014. Date of publication 30 Oct. 2014; date of current version 16 Sept. 2015. 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.2014.2366457

a user provides her selection of trusted applications and her ranking of permission groups in terms of their relevancy to the corresponding applications. Then, RiskMon builds a risk assessment baseline based on the runtime behaviors of the user's selected trusted applications and her ranking of permission groups. After the baseline is established, RiskMon continuously monitors every installed application's behaviors, including their interactions with other applications and system services. The risk of each interaction is measured by how much it deviates from the risk assessment baseline. For a better risk perception, RiskMon ranks the installed applications based on the risk assessment results in a real-time manner. Intuitively, the user can deem an application as safe if it is less risky than any of her trusted applications.

To facilitate risk mitigation, we further propose a decision process to automatically revoke risky permissions. Whereas simply uninstalling risky applications might disrupt user experiences, our decision process automatically confines unnecessary privacy-infringing code and meanwhile retains core functionalities of applications. Tools like RiskMon would practically help raise awareness of security and privacy problems and lower the sophistication required for concerned users to better understand and mitigate the risks of third-party mobile applications.

This paper makes the following contributions:

- We propose a methodology for establishing a risk assessment baseline from a user's trusted applications and her coarse expectations;
- We propose a machine-learned ranking based framework that continuously monitors the behaviors of installed applications, automatically measures their risks, and intuitively presents the risks;
- We propose an automated decision process that selectively revokes risky permissions with minimal impact on an application's usability;
- We implement a proof-of-concept prototype of RiskMon and demonstrate how it can be seamlessly deployed in Android; and
- We evaluate RiskMon with comprehensive experiments, case studies, and crowd-sourced user surveys. Our experimental results demonstrate the feasibility and practicality of RiskMon.

The remainder of this paper proceeds as follows. Section 2 provides the motivation and background of this paper. Section 3 provides an overview of RiskMon and illustrates the stages of automated risk assessment. Section 4 describes how RiskMon supports automated permission revocation. Section 5 presents the prototype implementation and evaluation. Section 6 discusses limitations of our approach. Section 7 discusses related work and Section 8 concludes this paper.

2 Motivation and Background

Recent work has proposed mechanisms to extract risk signals from meta information on application markets such as permissions [8], [9], [10], [11], ratings [12], [13], and application descriptions [14]. Their limitation is that such information is not *directly* related to how and when sensitive resources are used. Whereas an application requests location-related permissions, it may stay in the background and keep probing a user's locations and surroundings. Furthermore, users deserve the rights to know what is happening on their own devices. Therefore, continuously monitoring applications' behaviors is indispensable towards effective risk assessment.

Previous research concerning applications' runtime behaviors specifies a set of risk assessment heuristics tailored to their specific problems. For example, TaintDroid [4] considers a case in which sensitive data is transmitted over the network. DroidRanger [15] and RiskRanker [16] assume that dynamically loaded code is a potential sign of malware. While these techniques provide valuable insights about runtime behaviors of mobile applications, they do not justify the appropriateness of the revealed behaviors. We argue that meta information can provide the necessary operational contexts that justify runtime behaviors for risk assessment. For example, a location-based application has good reasons to upload a user's locations for discovering nearby restaurants. In contrast, it does not make sense for a video player to use the locations and such behaviors should be considered as more risky.

Finally, we need to consider how users participate in risk assessment. First, different users would have disparate security requirements. Thus, we should grant users the capabilities to specify their preferences in terms of accessing their own sensitive information. Moreover, normal users do not possess the necessary technical knowledge for assessing applications' runtime behaviors and interpret numerical risk scores. Therefore, it is imperative to automate risk assessment in a way that requires less sophistication and intervention.

2.1 Background: Android Platform

Permission groups. Permission group is a logical collection of related permissions defined by Android. For example, `SOCIAL_INFO` includes permissions that access a user's contacts and call logs. Most permission groups are self-descriptive, such as `LOCATION` and `CAMERA`. Android also provides a short description for each permission group to elaborate its corresponding resources.

Binder IPC framework. While APIs enable applications to interact with each other and system services in their respective process sandboxes, they are implemented based on an underlying inter-process communication framework called *Binder*. It serializes data objects as *parcels* for sender process, and de-serialize parcels for recipient process. Binder also manages IPC transactions in which parcels are processed and delivered.

3 Automated Risk Assessment

IT risk assessment guidelines, such as NIST SP 800-30 [17] and CERT OCTAVE [18], provide a foundation for the development of effective risk management processes. They illustrate comprehensive methodologies that enable organizations to understand, assess and address their information risks. While these guidelines deal with the infrastructure and organizational risks by security experts, our framework attempts to adapt and automate the sophisticated risk assessment tasks for general users.

To this end, RiskMon needs to acquire a user's expected appropriate runtime behaviors, assess the risks of installed applications, and intuitively present the risks to the user.

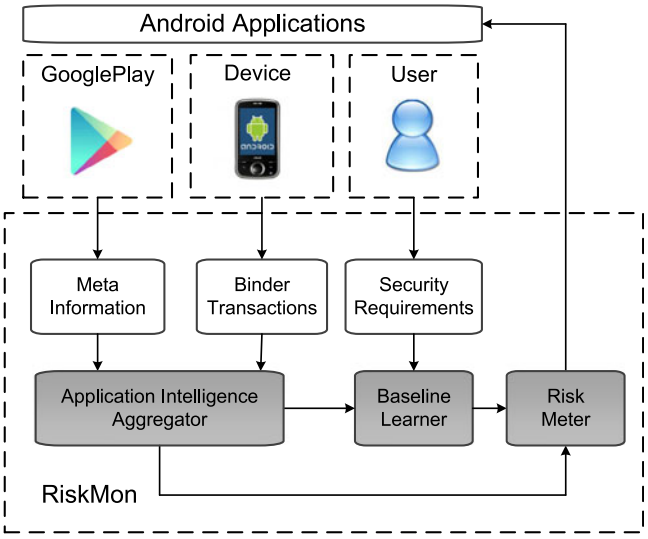


Fig. 1. RiskMon architecture for android.

There remain several challenges in achieving these goals. First, suppose users cannot directly specify runtime behaviors. RiskMon addresses this issue by leveraging a user's trusted applications to provide her expected behaviors. For example, Netflix and Pandora share the same core functionalities such as the streaming personalized media contents from remote servers. Hence, if a user trusts Netflix and derives a risk assessment baseline from its behaviors, the deviation or "distance" of the behaviors between Pandora and the baseline indicates Pandora's additional risks. However, a subsequent problem is how to measure the distance between runtime behaviors. Our proposed solution is to define a space with features extracted from Binder transactions and meta information of applications. Finally, a numerical distance does not appeal to users with respect to effective risk perception. We adopt a ranking of applications by their risks and a compositional view of risks for each application to intuitively present the measured risks.

Fig. 1 depicts the RiskMon architecture for Android. Our framework consists of three components: an application intelligence aggregator, a baseline learner, and a risk meter. The remainder of this section describes each component in detail.

3.1 Application Intelligence Aggregator

This component aggregates intelligence about a user's installed applications, including their runtime behaviors and operational contexts. As we capture applications' runtime behaviors by interposing the Binder IPC framework and log the Binder transactions, we propose a set of features tailored to the peculiarity of Binder IPC. Also, we derive operational contexts from meta information available on application markets. We also propose features to represent and characterize them. These features build a space of application intelligence and enable subsequent baseline generation and risk measurement.

3.1.1 Features for Binder Transactions

Android applications' runtime behaviors are essentially Binder IPC transactions that interact with system services and other applications. In this work, we only analyze permission-protected Binder transactions, assuming that a user's assets are only reachable through these transactions. Therefore, we need to identify the mappings from permissions to Binder transactions.

Specifically, we adopt existing work [19], [20] to provide the mappings from permissions to APIs. Meanwhile, we parse the AIDL[1] files in the AOSP repository to generate the mappings from APIs to Binder transactions. Connecting these two mappings together, we derive 1,003 types of permission-protected Binder transactions. Each of them is identified by a unique Binder interface name, direction of control flow (synchronous call or asynchronous callback), and a numerical command code. For example, a permission `ACCESS_FINE_LOCATION` protects a type of Binder IPC transaction "`ILocationManager-callback-1`". We note that one permission may protect multiple types of Binder transactions.

We attempt to represent a Binder transaction with its internal properties and contents. For a specific Binder transaction between an application and a system service, we are interested in its type to identify the corresponding asset. Also, we need to know the direction of control flow for determining who initiates the transaction. As users trust the system services more than applications, RiskMon should differentiate Binder transactions initiated by applications or system services. Thus, we propose the following Boolean features to capture the internal properties:

- *Type of Binder transaction*. One thousand and three Boolean features as a bit array, where one bit is set to 1 for the corresponding transaction type and the others are 0; and
- *Direction of control flow*. Another Boolean feature, where 0 for transactions initiated from applications (calls), 1 for transactions initiated from system services (callbacks).

In terms of contents, parcels in Binder transactions are unstructured and highly optimized, and it is hard to restore the original data objects without the implementation details of the sender and recipient. Therefore, we use length as one representative feature of parcels. A motivating example is accesses on contacts. From the length of a parcel we can infer whether an application is reading a single entry or dumping the entire contact database. In summary, we propose the following two features for parcels:

- *Length of received parcel*. Length of the parcel received by an application in bytes; and
- *Length of sent parcel*. Length of the parcel sent by an application in bytes.

These features are scaled and then standardized to zero mean and unit variance. Note that we empirically choose 0-4 KB as the range of parcel lengths. In our experiments, we found that the parcels larger than 4 KB exceeded the limit of the Binder transaction buffer and were discarded. However, Android applications sometimes encapsulate file descriptors in parcels to transfer large bulks of data. We will deal with this situation in our future work.

1. Android Interface Definition Language, http://developer.android.com/guide/components/aidl.html

3.1.2 Features for Meta Information

We inspect three application markets, including Apple App Store, Google Play, and Amazon Appstore. They all share several common meta properties that reflect users' and developers' opinions. We leverage such properties to propose corresponding features for meta information.

Specifically, we use the following features to represent users' opinions:

- *Number of installs*. A range of total number of installs since the first release.[2] We use logarithmic value of the lower bound, i.e., *log(1+lower bound of #installs)* and scale to [0,1];
- *Number of reviews*. A number of reviews written by unique users. We use the logarithmic value, i.e., *log(1 +#reviews)* and scale to [0,1]; and
- *Rating score*. A number indicating the user-rated quality of the application ranged from 1.0 to 5.0, scaled to [0,1].

These three features capture an application's popularity and reputation. The first two features are similar to number of views or comments in online social networks. Recent studies [21] demonstrated that online social networks and crowd-sourcing systems expose a long-tailed distribution. Therefore, we assume they follow the same distribution and use logarithmic values.

We emphasize that we do not attempt to extract risk signals from these features. Instead, we adopt these features to capture the underlying patterns of a user's trusted applications as specified by the user and apply the patterns for the subsequent risk assessment.

Next, we propose a feature to capture developers' opinions:

- *Category*. A tuple of two numerical values normalized to $[-0.5, 0.5]$.

An application's category describes its core functionalities (e.g., "Communication"). Note that each application market may define its specific application categories. As we focus on Android applications in this work, we use the categories defined by Google Play throughout the remainder of this paper.

We adopt the Self-Organizing Map (SOM) in a previous work by Barrera et al. [22] to derive a two-dimensional representation of categories. SOM can produce a discretized representation of permissions requested by different categories of Android applications. Categories in which applications request similar permissions are clustered together. Therefore, the x and y coordinates in the map can represent a category, and categories with similar core functionalities would be closer to each other. Fig. 2 depicts the coordinates of 13 categories as an example. Apparently, some categories bear underlying similarities, such as "Entertainment", "Media and Video" and "Music and Audio".

An unscrupulous developer can claim an irrelevant category to disguise an application's intended core functionalities. However, a user can easily notice the inconsistencies and remove such an application. In addition, falsifying an application's meta information violates the terms of application market's developer policies and may lead to immediate takedown.

2. The number of installs is specified with exponentially increasing ranges: 1+, 5+, . . . , 1K+, 5K+, . . . , 1M+, 5M+.

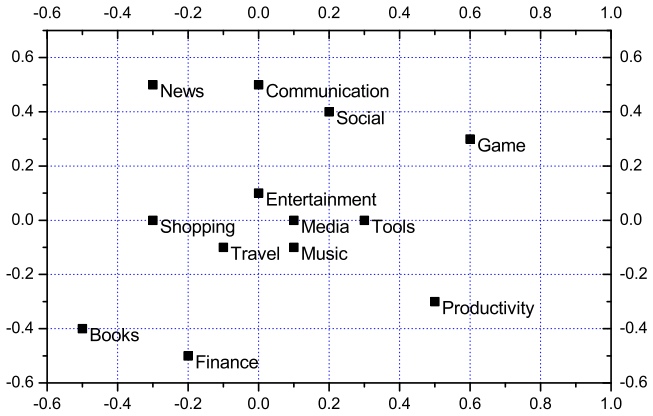


Fig. 2. SOM representation of 13 categories.

Finally, based on the scheme defined by these features, the application intelligence aggregator generates a data set consisted of feature vectors extracted from Binder transactions and meta information of each installed application.

3.2 Baseline Learner

The baseline learner is the core module of RiskMon. It takes two types of inputs, which are a user's expectations and feature vectors extracted by the application intelligence aggregator. Then the baseline learner generates a risk assessment baseline that is represented as a predictive model.

3.2.1 Acquiring Security Requirements

It is challenging for most users to express their security requirements accurately. We aim to find an approach that could be mostly acceptable by users. Krosnick and Alwin's dual path model [23] demonstrated that a *satisficing* user would rely on *salient cues* to make a decision. Based on this model we develop a simple heuristic:

> *For a specific application, accesses on resources that are more irrelevant of a user's expected core functionalities incur more risks.*

Core functionalities of an application are obvious to most users, and we use such "salient cues" to support automated risk assessment. Furthermore, the heuristic captures a user's expectations by risk aversion, which implies the reluctance of a user to use a functionality with an unknown marginal utility [24]. For example, a user may consider that, microphone is necessary to a VoIP application such as Skype. But location seems not because she does not understand the underlying correlation between disclosing her location and making a phone call. Thus, microphone is more relevant and less risky than location in her perception.

Based on this, the risk learner asks a user to specify a relevancy level for each permission group requested by her trusted applications. We choose permission groups to represent resources because it is much easier for general users to learn 20+ permission groups than 140+ permissions. And recent usability studies demonstrated the ineffectiveness of permissions due to limited comprehension [25], [26]. Although users tend to overestimate the scope and risk of

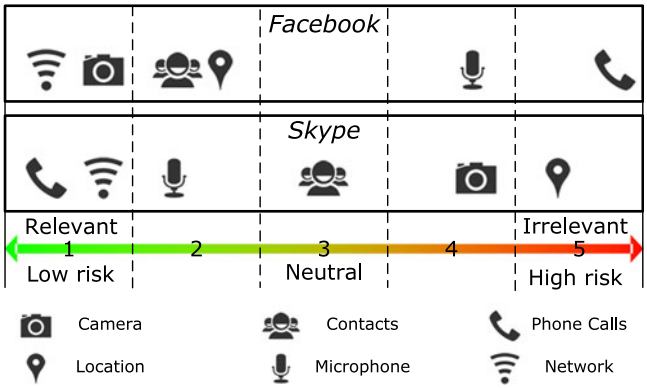


Fig. 3. An example of specifying relevancy for permission groups.

permission groups, they are more intuitive and reduce warning fatigue [25].

The process for users to communicate their security requirements with RISKMON is similar to a short questionnaire. Each permission group requested by a user's trusted applications corresponds to a five-point Likert item. The user specifies the level of relevancy on a symmetric bipolar scale, namely *relevant*, *probably relevant*, *neutral*, *probably irrelevant* or *irrelevant*. Fig. 3 shows an example of relevancy of permission groups for Facebook and Skype. Permission groups are represented by self-descriptive icons, which are identical to those shown in Android Settings. `CAMERA` preceding `LOCATION` for Facebook is possibly due to the user's preference to photo sharing compared to check-ins.

Note that the relevancy levels specified by users are *subjective*. With that said, users' biased perception of applications and resources may affect their specified relevancy levels. From our user study, a user told us that `PHONE_CALLS` is relevant to Google Maps because he tapped a phone number shown in Google Map and then the dialer appeared. Although the dialer rather than Google Map has the capability to make phone calls, the baseline learner considers it as the security requirements for inter-application communication.

We next formalize the problem of acquiring security requirements as follows:

- $A = \{a_1, a_2, \ldots, a_n\}$ is a set of a user's installed applications;
- A_T is a set of a user's trusted and installed applications and $A_T \subseteq A$;
- $PG = \{pg_1, pg_2, \ldots, pg_m\}$ is a set of permission groups available in a mobile operating system;
- $RL = \{1, 2, 3, 4, 5\}$ is a set of relevancy levels, where a larger value indicates higher relevancy and less risk and vice versa; and
- Req is a user's security requirement, which is essentially a mapping $Req : A_T \times PG \rightarrow RL$.

3.2.2 Compiling Training Set

Next we describe how the baseline learner compiles a training set. Simply put, it annotates vectorized Binder transactions with user-specified relevancy levels.

To bridge the gap between permission groups and feature vectors, we extract mappings of permission groups and permissions from the source code of Android. Meanwhile, existing work has provided mappings between permissions and APIs [19], [20]. Therefore, we can assign the relevancy level on feature vectors because each vector represents an API call or callback.

We formalize the problem of compiling a training set:

- X is a set of vectorized Binder transactions generated by a user's installed applications, where the features are extracted from properties of Binder transactions (Section 3.1.1) and meta information of the corresponding applications (Section 3.1.2);
- X_T is a set of vectorized Binder transactions generated by the user's trusted and installed applications, $X_T = \{\vec{x} | \vec{x} \in X, \vec{x} \text{ is generated by } a, a \in A_T\}$; and
- $T = \{(\vec{x}_1, rl_1), (\vec{x}_2, rl_2), \ldots, (\vec{x}_n, rl_n)\}$ is a training set of annotated vectors, $\vec{x}_k \in X_T$, $rl_k \in RL$.

We define two helper functions:

- $GetA : X_T \rightarrow A_T$ is a function that maps a vector to its corresponding application; and
- $GetPG : X_T \rightarrow PG$ is a function that maps a vector to its corresponding permission group.

Algorithm 1 illustrates the process to compile the training set T, where feature vectors from X_T are annotated with relevancy levels specified by a user's security requirements.

Algorithm 1. Compiling Training Set

```
Data: X_T, Req
Result T
T ← ∅;
for x⃗ ∈ X_T do
    a ← GetA(x⃗); pg ← GetPG(x⃗);
    rl ← Req(a, pg);
    add (x⃗, rl) to T;
end
return T
```

3.2.3 Generating Risk Assessment Baseline

Duh [37] shows that Ranking Support Vector Machine (RSVM) [27] performs better than regression with respect to eliciting human judgement in evaluating machine translation systems. Next, we explain how we apply RSVM to derive a risk assessment baseline for assessing mobile applications.

We assume that a set of ranking functions $f \in F$ exists and satisfies the following:

$$\vec{x}_i \prec \vec{x}_j \;\Leftrightarrow\; f(\vec{x}_i) < f(\vec{x}_j), \tag{1}$$

where $\prec$ denotes a preferential relationship of risks.

In the simplest form of RSVM, we assume that f is a linear function:

$$f_{\vec{w}}(\vec{x}) = \langle \vec{w}, \vec{x} \rangle, \tag{2}$$

where $\vec{w}$ is a weight vector, and $\langle \cdot, \cdot \rangle$ denotes inner product.

Combing (1) and (2), we have the following:

$$\vec{x}_i \prec \vec{x}_j \;\Leftrightarrow\; \langle \vec{w}, \vec{x}_i - \vec{x}_j \rangle < 0, \tag{3}$$

Note that $\vec{x}_i - \vec{x}_j$ is a new vector that expresses the relation $\vec{x}_i \prec \vec{x}_j$ between $\vec{x}_i$ and $\vec{x}_j$. Given the training set T, we

create a new training set T' by assigning either a positive label $z = +1$ or a negative label $z = -1$ to each pair $(\vec{x}_i, \vec{x}_j)$.

$$(\vec{x}_i, \vec{x}_j) : z_{i,j} = \begin{cases} +1 & \text{if } r_i > r_j \\ -1 & \text{if } r_i < r_j \end{cases} \quad \forall (\vec{x}_i, r_i), (\vec{x}_j, r_j) \in T. \tag{4}$$

To select a ranking function f that fits the training set T', we construct the SVM model to solve the following quadratic optimization problem:

$$\begin{array}{ll} \underset{\vec{w}}{\text{minimize}} & \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum \xi_{i,j} \\ \text{subject to} & \forall (\vec{x}_i, \vec{x}_j) \in T' : z_{i,j} \langle \vec{w}, \vec{x}_i - \vec{x}_j \rangle \geq 1 - \xi_{i,j} \\ & \forall i \forall j : \xi_{i,j} > 0. \end{array} \tag{5}$$

Denoting $\vec{w}^*$ as the weight vector generated by solving (5), we define the risk scoring function $f_{\vec{w}^*}$, for assigning risk scores to the feature vectors (i.e., vectorized Binder transactions):

$$f_{\vec{w}^*}(\vec{x}) = \langle \vec{w}^*, \vec{x} \rangle. \tag{6}$$

For any $\vec{x} \in X$, the risk scoring function measures its projection onto $\vec{w}^*$, or the distance to a hyperplane whose normal vector is $\vec{w}^*$. Thus, the hyperplane is indeed the risk assessment baseline.

3.3 Risk Meter

Risk meter measures the risks incurred by each installed application, including a user's trusted application as well. Note that (6) gives a signed distance. In (7) we use the absolute value to represent the deviation and risk, because the sign simply indicates whether $\vec{x}$ is on one side of the RSVM's margin or the opposite. The risks incurred by an application a_i are the cumulative risks of its Binder transactions:

$$\sum |f_{\vec{w}^*}(\vec{x})|, \text{where } \vec{x} \text{ is generated by } a_i. \tag{7}$$

Another goal of the risk meter is to provide supporting evidences to end-users. To this end, it presents the measured risks at three levels of granularities.

Application. In the simplest form, the risk meter presents a ranking of installed applications by their risks as a bar chart. The *x*-axis indicates the applications and the *y*-axis indicates the risks. A user can trust an application if it is less risky than her trusted ones. In contrast, an application that is significantly risky can also draw a user's attention.

Permission group. The ranking of applications may seem unconvincing sometimes for users. In such a case, the risk meter can provide risk composition by permission groups that is represented as a pie chart. The pie chart intuitively reveals the proportion of the risks incurred by the core functionalities of an application. As users have basic knowledge of permission groups when they specify security requirements, they should be able to interpret the risk composition correctly.

Permission. Considering a user's limited comprehension of permissions, we do not present general users with the evidences that are more fine-grained than permission groups. Evidences presented at this level are intended for experienced users who would like to tune up their security requirements. Particularly, our automated risk mitigation mechanism (Section 4) also utilizes these evidences for permission revocation.

Moreover, RiskMon allows a user to establish and revise her security requirements iteratively. RiskMon may generate biased or unconvincing evidences as a user may not have clear and accurate security requirements at the very beginning of using RiskMon. Thus, a user can provide her feedback by adjusting her security requirements and/or adding more trusted applications. RiskMon also periodically updates the security assessment baseline for observed new runtime behaviors. All of these enable RiskMon to approximate an optimum risk assessment baseline to help users make better decisions.

4 Automated Risk Mitigation

Based on the proposed risk assessment framework, we move one step further to address risk mitigation. Specifically, we propose an automated decision process that assists users to conveniently identify and revoke risky permissions from installed applications.

A typical permission framework, just like common access control systems, involves decision processes that grant and revoke permissions. While permission granting has been widely adopted in modern mobile platforms, permission revocation has not received a commensurate popularity. For example, iOS users could not deny accesses to their personal information until iOS 6. Google introduced App Ops as an experimental privacy control framework in Android 4.3, but later disabled its management interface in Android 4.4.2 [28].

Permission revocation is necessary because it enables complete and flexible control over granted capabilities. To this end, recent work has proposed enhanced middleware mandatory access control (MMAC) frameworks to support rule-driven permission revocation on Android [29], [30], [31], [32], [33]. An obvious limitation of such frameworks lies in the definition and maintenance of the rules [34], which place non-negligible burden on general users. To say the least, it remains an open question whether users can accurately cherry-pick the risky permissions that should indeed be revoked.

Intuitively, RiskMon could provide the necessary evidences to support a permission revocation decision process. However, Android by default only allows users to mitigate unnecessary risks is removing risky applications. Such an arbitrary approach may disrupt user experiences. For example, grey applications (e.g., ad-supported games) are likely to request excessive permissions for harvesting user information. Revoking all the granted permissions (i.e., removing application) seems unnecessary because some permissions are not major sources of risks and they may support functionalities that a user needs. Our goal is to selectively revoke risky permissions and mitigate future risks to a user's expected level. Therefore, those grey applications might still retain necessary functionalities and users could stay protected from privacy-infringing code.

We identify three key challenges in bridging the gap between risk assessment and risk mitigation: (1) selecting reference applications; (2) estimating risk budgets; and (3)

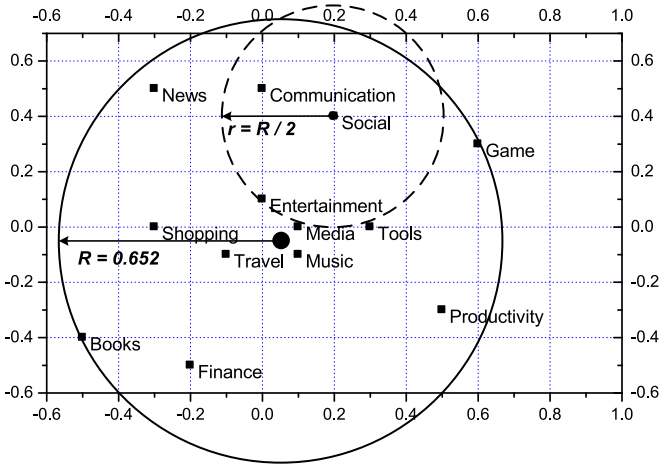


Fig. 4. An example of selecting reference applications from close categories.

enforcing decisions with minimal user intervention. Reference applications implicitly provide a user's expected runtime behaviors and upper bounds of acceptable risks. Risk budgets quantitatively determine decision thresholds that line up with the user's risk mitigation strategies. Moreover, we need to minimize user intervention in decision enforcement, because general users would be incapable and reluctant to create and manage security policies. We next describe how we address these challenges.

4.1 Selecting Reference Applications

As we previously assumed, a user's trusted applications define her expected appropriate behaviors for similar applications. To select a set of reference applications for a target application, we prefer trusted applications that are under the *same* or *close* categories because their core functionalities tend to be similar. Therefore, we assign coordinates to all the installed applications according to their categories in the category SOM. Then, we select the reference applications by computing a set of k-nearest trusted applications based on their Euclidean distances.

The best choice of k depends on the category SOM and the number of the trusted applications. Here we adopt a conservative approach to avoid over-generalization that could lead to over-estimation of risk budgets. First, we start from $k \leq \lfloor log_2|A_T| \rfloor$. Meanwhile, we need to filter this set by removing applications that are not close enough to the target application. To quantitatively define "close", we compute the smallest enclosing circle of the category SOM and its radius R, and choose $R/2$ as the threshold of close categories. In summary, a target application a's reference application set A_{Ra} is the intersection of the following sets:

1) $\lfloor log_2|A_T| \rfloor$-nearest trusted applications; and
2) the trust applications whose Euclidean distance from a is no larger than r, where $r = R/2$.

Fig. 4 demonstrates an example of selecting reference applications for a social application. The result is no more than $\lfloor log_2|A_T| \rfloor$ applications under the "Social", "Communication", and/or "Entertainment" categories.

Automated risk mitigation is also limited by the same problem of insufficient trusted applications as automated risk assessment. A_{Ra} could be empty because A_T does not cover sufficient categories. In such a case, reselecting reference applications is scheduled after a user adds trusted applications and improves coverage.

4.2 Estimating Risk Budgets

Risk budgets define decision thresholds used in our automated decision process. Our goal is to derive a risk budget for each permission of a target application from its reference applications.

We next formalize the problem of estimating risk budgets for a target application a as follows:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a set of permissions available in a mobile operating system;
- $UsedP : A \rightarrow 2^P$ is a function that maps an application to a set of permissions whose usage patterns have been observed by RiskMon;
- $PAR : P \times A \rightarrow \mathbb{R}$ is a function that maps a granted permission of an application to its measured risk score; and
- $BI_a = \bigcup_{ta \in A_{Ra}} UsedP(ta)$ is a set of permissions that are the budget items for an application a.

We then introduce the following *budget estimation functions* to support different risk mitigation strategies, where $p \in UsedP(\text{a})$, $a \in A$, $a \notin A_T$, $A_{Ra} \subset A_T$:

$$
\begin{aligned}
Strict_a(p) &= \begin{cases} \min\limits_{ta \in A_{Ra}} PAR(p, ta) & \text{if } p \in BI_a \\ 0 & \text{if } p \notin BI_a \end{cases} \\
Average_a(p) &= \begin{cases} \operatorname{avg}\limits_{ta \in A_{Ra}} PAR(p, ta) & \text{if } p \in BI_a \\ 0 & \text{if } p \notin BI_a \end{cases} \\
Relaxed_a(p) &= \begin{cases} \max\limits_{ta \in A_{Ra}} PAR(p, ta) & \text{if } p \in BI_a \\ \operatorname{avg}\limits_{ta \in A_T} PAR(p, ta) & \text{if } p \notin BI_a. \end{cases}
\end{aligned} \tag{8}
$$

The **strict** function prefers the most privacy-preserving practices of the reference applications. The **average** function attempts to reduce the risks below the average practices. For the permissions not among the budget items, the **strict** and **average** functions both opt for a zero tolerance strategy. In contrast, the **relaxed** function allows such permissions but their incurred risks should not exceed the average of all the trusted applications.

4.3 Generating and Enforcing Decisions

To generate a decision for a permission p of an application a, we compute its cumulative risks as $Risk_a(p)$ and apply a user-specified budget estimation function, for example:

$$
Decision(a, p) = \begin{cases} \text{Keep} & \text{if } Risk_a(p) \leq Strict_a(p) \\ \text{Revoke} & \text{if } Risk_a(p) > Strict_a(p). \end{cases} \tag{9}
$$

Note that an important criterion of our decision process is *revoking by observed behaviors*.[3]

Managing security policies for complex information systems has been a challenging task. It is even harder for

3. Intuitively, dormant permissions do not incur any risks so we choose not to revoke them because we have no observed evidence to prove that such permissions will be abused.

TABLE 1
Applications Assumed to be Trusted by the Participants in the User Study

Application	Category
AmazonMobile	Shopping
BejeweledBlitz	Game
ChaseMobile	Finance
Dictionary.com	Books & Reference
Dropbox	Productivity
Google+	Social
GooglePlayMovies&TV	Media & Video
Hangouts(replacesTalk)	Communication
MoviesbyFlixster	Entertainment
Yelp	Travel & Local

TABLE 2
Demographics of the Participants

	Category	# of users
Gender	Male	29 (87.9%)
	Female	4 (12.1%)
Age	18-24	15 (45.5%)
	25-34	16 (48.5%)
	35-54	2 (6.1%)
Education	Graduated high school or equivalent	3 (9.1%)
	Some college, no degree	6 (18.2%)
	Associate degree	1 (3.0%)
	Bachelor's degree	11 (33.3%)
	Post-graduate degree	12 (36.4%)

dynamic systems such as the Android middleware, whose security policies have to confine various applications that rapidly update themselves. Enforcing security decisions for such systems would be unrealistic for general users because it consumes much user attention and leads to habituation [35]. This partially implies why Android community has been careful with integrating user-oriented and generic permission revocation [28].

We introduce automated policy generation to address this challenge. Specifically, automated permission revocation and policy generation are activated after (1) a user installs or updates a new application; (2) a user updates her risk assessment baseline; or (3) a pre-defined time period. Note that we do not attempt to implement our own policy enforcement mechanisms. Instead, our framework could be easily adapted to support new middleware MAC frameworks with an intuitive policy translation module.

5 Implementation and Evaluation

In this section we first discuss a proof-of-concept implementation of RiskMon. Then, we present the results of our online user study followed by the case studies of automated risk assessment and mitigation. We conclude our evaluation with the usability and performance.

5.1 Implementation and Experimental Setup

We implemented a proof-of-concept prototype of RiskMon on the Android mobile platform. In terms of continuous monitoring, we implemented a reference monitor for Binder IPC by inserting hooks inside the Binder userspace library. The hooks tap into Binder transactions and log the parcels along with senders' and recipients' UIDs.[4] In addition, we implemented automated risk assessment based on SVMLight[5] and its built-in Gaussian radial basis function kernel. To tune the SVM for better performance, we used a grid-search to test an exponential sequence of $C = 10^{-5}, 10^{-4}, \ldots, 10^{5}$, where C is the penalty parameter. The other parameters kept their default values as provided by SVMLight.

We conducted a user study of 33 participants to evaluate the practicality and usability of RiskMon. We hand-picked 10 applications (Table 1) that were most downloaded from Google Play in their respective categories. We assumed that all the participants trust them. Then we used participants' security requirements for the 10 applications and their application intelligence to generate the baselines. We also randomly selected four target applications from the Top Charts of Google Play to calculate their risks based on the generated baselines, including: a) CNN App for Android Phones (abbreviated as CNN); b) MXPlayer; c) Pandora Internet Radio (abbreviated as Pandora); and d) Walmart. For both trusted (10) and target (4) applications, we collected their one-day runtime behaviors on a Samsung Galaxy Nexus phone. In addition, we developed a web-based system[6] that acquires a participant's security requirements, feeds them to RiskMon and presents the results calculated by RiskMon to the participant. A participant was first presented with a tutorial page that explains how to specify relevancy levels as her security requirements. Then she was required to set relevance levels for each permission group requested by each trusted application after reading the application's descriptions on Google Play. Afterwards, RiskMon generated a risk assessment baseline for the participant based on her inputs and runtime behaviors of the 10 trusted applications. Then RiskMon applied the baseline on each of the 14 applications, and displayed a bar chart that illustrates a ranking of 14 applications by their measured cumulative risks. Finally, an exit survey was presented to collect the participant's perceived usability of RiskMon. Our study protocol was reviewed by our institution's IRB. And we recruited participants through university mailing lists and Amazon MTurk. Table 2 lists the demographics of the 33 participants.

5.2 Empirical Results

5.2.1 Security Requirements

From our user study on the applications shown in Table 1, we highlight the results of Chase Mobile and Dropbox because they both request some ambiguous permission groups that are hard to justify for users. Fig. 5 demonstrates the average relevancy levels set by the participants for each permission group requested by Chase Mobile and Dropbox. The error bars indicate the standard deviation.

Chase Mobile is a banking application with functionalities like depositing a check by taking a picture and locating nearest branches. Apparently NETWORK is more relevant than

4. Ad libraries are assessed separately in case of ADSplit [36].
5. http://svmlight.joachims.org/
6. Screenshots are available at http://goo.gl/xIuYp1

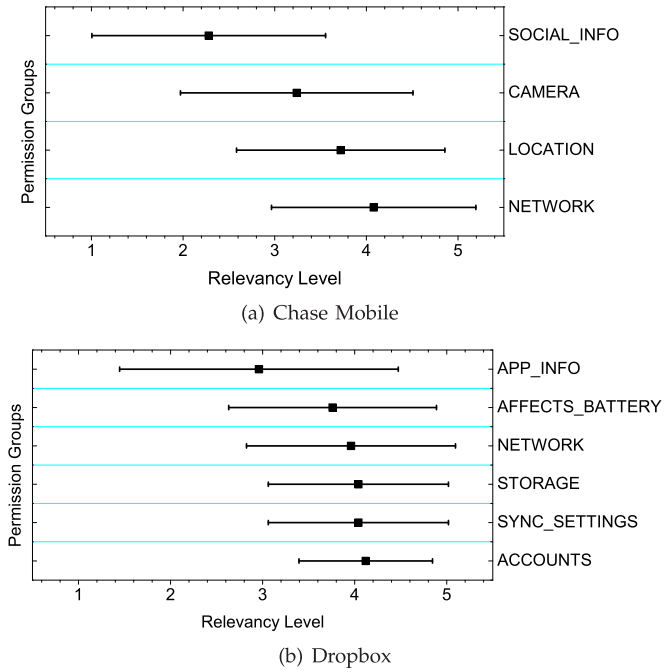


Fig. 5. Average relevancy levels specified by the participants for chase mobile and dropbox.

others as participants agree that Chase Mobile needs to access the Internet. Even though Chase Mobile uses LOCATION to find nearby bank branches and CAMERA to deposit checks, both LOCATION and CAMERA have lower relevancy levels than NETWORK. We believe it is because some participants do not have the experiences of using such functionalities, but the averages are still higher than neutral. Furthermore, SOCIAL_INFO falls below "neutral", showing participants' concerns of why Chase Mobile uses such information.

Dropbox is an online file storage and synchronization service. From its results, we identified an interesting permission group, APP_INFO, whose description in Android's official document is: *group of permissions that are related to the other applications installed on the system*. This authoritative description does not provide any cue of negative impacts, which leads to user confusion as we can see that APP_INFO has the largest standard deviation. STORAGE, SYNC_SETTINGS and ACCOUNTS are all above "probably relevant" possibly due to their self-descriptive names that are semantically close to Dropbox's core functionalities.

Moreover, we noticed that the participants tend to set higher relevancy levels for self-descriptive permission groups, while they tend to be conservative for other permission groups. We note that this does not affect RISKMON in acquiring a user's security requirements, because RISKMON captures the precedence of one permission group over another. Thus, the least relevant permission group always gets the highest risk scores.

5.2.2 Application Risk Ranking

Fig. 6 illustrates the ranking of 14 applications by their average cumulative risk scores as measured by 33 risk assessment baselines generated for the participants. We can see that MXPlayer (2.55) and Walmart (12.72) fall within the trusted applications, while CNN (54.15) and Pandora (69.22) are ranked with the highest risk scores.

Note that both Pandora and CNN are renowned applications developed by experienced developers. Seemingly, they should use sensitive information appropriately. Hence, we verified them by manually dissecting their captured Binder transactions. We found that they kept polling ConnectivityManager for a fine-grained state of the current network connection, which generated hundreds of Binder transactions albeit each transaction was not very risky. This is an unexpected practice with respect to privacy and performance, because the official Android documents[7] suggest developers register CONNECTIVITY_CHANGE broadcasts to get connectivity updates instead of polling. On the contrary, Hangouts incurred almost imperceptible amount of risks, although it has similar requirements for connectivity. Therefore, RISKMON showed that even popular applications might use sensitive information in a way that incurs potential risks for users.

5.3 Case Studies

Note that there is no ground truth of users' expected appropriate behaviors. Therefore, we opt for several case studies to evaluate the effectiveness of our approaches. We handpicked two applications, SogouInput and PPS.TV, because they both request one or more sensitive and excessive permissions. We also reused the target applications that were previously selected for the user study. To assess these six applications, we specified the relevancy levels for the 10 trusted applications and generated a risk assessment baseline. We verified their identified risk composition with manual analysis. Finally, we applied automated permission revocation to identify and mitigate their unnecessary risks. Tables 3 and 4 demonstrate the results of automated risk assessment and mitigation, respectively.

5.3.1 Automated Risk Assessment

SogouInput is an input method based on the pinyin method of romanization, and PPS.TV is a video streaming application similar to its counterparts such as Hulu and Netflix. Both of them are feature-rich, free and have accumulated over 5,000,000 installs on Google Play. We note that PPS.TV and SogouInput request 22 and 29 permissions, respectively. The numbers of requested permissions make them suspicious over-privileged or privacy-infringing applications.

The measured cumulative risk scores are 179.0 for SogouInput and 366.9 for PPS.TV. First, the unusually large portion of PHONE_CALLS indicates substantial use of capabilities related to making phone calls and reading unique identifiers. We verified the corresponding Binder transactions and revealed that it attempted to read a user's subscriber ID and device ID. Second and more notably, SOCIAL_INFO contributed 4.02 percent of the total risks incurred by SogouInput. We verified the corresponding Binder transactions and found that SogouInput accessed content://com.android.contacts and received a parcel of 384 bytes. Usually an Android application queries the contact application and receives only the entries a user picks, which is several bytes long. On the contrary,

7. http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html

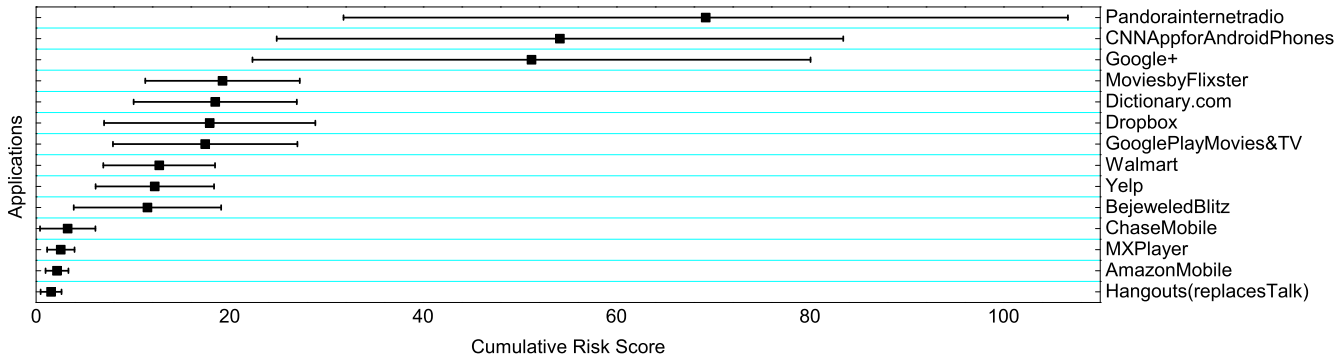


Fig. 6. Average cumulative risk scores measured by the participants' risk assessment baselines.

SogouInput attempted to dump the whole contact repository. Similar to SogouInput, PPS.TV utilized permissions related to PHONE_CALL. In addition to reading a user's device ID and subscriber ID, it also registered a callback to receive events of call states. We note that this allows PPS.TV to read the number of incoming calls. To verify which trusted applications were mostly used in assessing the appropriateness of the behaviors of SogouInput and PPS.TV, we performed an exhaustive leave-p-out cross validation on the 10 trusted applications. The results showed that Google+, MoviesbyFlixter, and Dropbox contributed most in assessing SogouInput. And Google+, Dictionary.com contributed most for PPS.TV.

As for the four target applications, three of them used the APIs related to NETWORK and LOCATION. The considerable risks of NETWORK incurred by Pandora and CNN were due to polling ConnectivityManager as we have discussed in Section 5.2.2. Meanwhile, CNN and Walmart both continuously tracked a user's location through APIs related to LOCATION. MXPlayer could be deemed as safe due to its low cumulative risks as well as reasonable risk composition. Moreover, the major source of risks could imply whether an application abuses a user's information. For example, LOCATION contributed a majority of Walmart's total risks, which simply does not make sense for a shopping application.

TABLE 3
Risk Composition of Applications in Case Studies

Application	Permission Group	Risk Score
SogouInput	LOCATION	5.6 (3.13%)
	NETWORK	104.4 (58.29%)
	PHONE_CALLS	61.8 (34.56%)
	SOCIAL_INFO	7.2 (4.02%)
	Total:	179.0 (100%)
PPS.TV	LOCATION	26.0 (7.09%)
	NETWORK	108.3 (29.52%)
	PHONE_CALLS	232.6 (63.40%)
	Total:	366.9 (100%)
Pandora	AFFECTS_BATTERY	0.27 (0.21%)
	NETWORK	131.6 (99.49%)
	PHONE_CALLS	0.4 (0.30%)
	Total:	132.3 (100%)
CNN	LOCATION	26.7 (20.75%)
	NETWORK	101.8 (79.25%)
	Total:	128.5 (100%)
Walmart	LOCATION	40.8 (72.05%)
	NETWORK	15.8 (27.95%)
	Total:	56.6 (100%)
MXPlayer	NETWORK	5.3 (100.00%)
	Total:	5.3 (100%)

5.3.2 Automated Risk Mitigation

Based on the measured risks of the six applications, we further applied our automated risk mitigation approach. In particular, we used Fig. 2 to guide our selection of reference applications out of 10 trusted applications. Therefore, $r = R/2 = 0.326$ as shown in Fig. 4 and k was no more than 3. Afterwards, we chose the **average** budget estimation function to reduce the incurred risks of the applications that are below the average level of their respective reference applications. Table 4 shows the revoked permissions and risk reduction of the assessed applications. In this table, we have denoted the specific reason for each revoked permission. "(O)" indicates that the revoked permission was used by one or more reference applications but exceeded the threshold set by the budget estimation function. "(N)" means that the permission was not used by any of the reference applications. Such permissions were also revoked in our case studies due to the **average** function's zero tolerance strategy.

The revoked permissions are lined up with the results as shown in Table 3. In particular, READ_CONTACTS

TABLE 4
Revoked Permissions of Applications in Case Studies

Application	Revoked Permissions (O): Over budget (N): Not in budget	Risk Reduction
SogouInput	ACCESS_NETWORK_STATE (O) READ_PHONE_STATE (O) READ_CONTACTS (N) VIBRATE (N)	169.5 (94.7%)
PPS.TV	ACCESS_LOCATION (O) ACCESS_NETWORK_STATE (O) ACCESS_WIFI_STATE (O) CHANGE_WIFI_STATE (N) READ_PHONE_STATE (O)	367.0 (99.8%)
Pandora	ACCESS_NETWORK_STATE (O)	130.3 (98.5%)
CNN	ACCESS_LOCATION (N) ACCESS_NETWORK_STATE (O) WAKE_LOCK (N)	128.5 (100.0%)
Walmart	ACCESS_LOCATION (O) ACCESS_NETWORK_STATE (O)	56.6 (100.0%)
MXPlayer		0.0 (0.0%)

TABLE 5
Usability Evaluation Results

Metric	Average	Lower bound on 95% confidence interval
Likeability	0.811	0.797
Simplicity	0.674	0.645
Risk perception	0.758	0.751

TABLE 6
Microbenchmark Results

Benchmark	Average (s)	Standard Deviation (s)
Feature extraction	8.27	0.07
Baseline generation (10 apps)	289.56	235.88
Risk measurement (per app)	0.55	0.17

and VIBRATE were revoked from SogouInput because they were used but not among the risk budget items. In contrast, none of permissions related to LOCATION was revoked, implying that SogouInput used LOCATION in a reasonable and conservative manner. Four out of five revoked permissions of PPS.TV were mitigated due to over-budget, demonstrating its notable tendency of abusing a user's information. Overall, these applications were confined to behave like their respective reference trusted applications.

We enforced the generated decisions through AppOps, and the revoked permissions did not break the core functionalities. However, we can not guarantee that permission revocation does not significantly impair an application's usability, for two reasons. First, our framework does not directly enforce decisions. Graceful enforcement of decisions by access control frameworks is still an open question that is beyond the scope of this paper. Second, risky permissions are not always excessive. Obviously, core functionalities would break if their abused permissions are revoked.

The results of the case studies leave room for further analysis. How come an input method and a video streaming application need capabilities related to PHONE_CALLS, LOCATION and SOCIAL_INFO? Why does Walmart need to continuously access users' location? Possibly users could get personalized services through disclosing private information. However, it comes with a price. RiskMon is a necessary step towards highlighting and mitigating the excessive risks.

5.4 System Usability

The criteria for usability were split into three areas: *likeability*, *simplicity* and *risk perception*. Likeability is a measure of a user's basic opinion towards automated risk assessment. This identifies whether users would like to accept the proposed mechanism. Simplicity is a measure of how intuitive the concepts and procedures are, which is useful in evaluating the burden placed on users. Risk perception is a measure of a user's perceived awareness of risks through risk assessment, which evaluates how users interpret the presented risk.

After using RiskMon, an exit survey was presented to collect users' perceived usability of RiskMon. In the survey, we asked users a set of questions on *likeability*, *simplicity*, and *risk perception*. Questions were measured with a five-point Likert scale. A higher score indicates a positive opinion or agreement and vice versa. Then scores were adjusted to [0,1] for numerical analysis.

We analyzed a 95 percent confidence interval for users' answers. Specifically, we are interested in determining the average user's minimum positive opinions. Hence, we looked at the lower bound of the confidence interval. Table 5 shows that an average user asserts 79.7 percent positively on likeability, 64.5 percent on simplicity and 75.1 percent on risk perception. The results show usability of RiskMon with the above-average feedback.

5.5 System Overhead

To understand the performance overhead of RiskMon, we performed several microbenchmarks. The experiments were performed on a Samsung Galaxy Nexus phone with a 1.2 GHz dual-core ARM CPU. The phone ran Android v4.2.2 and RiskMon built on the same version. Table 6 shows the average results.

Feature extraction. The application intelligence aggregator extracted feature vectors from 33,368,458 Binder IPC transactions generated by 14 applications in one day. We measured the CPU-time used by parsing the transactions and generating the feature vectors. The average time is 8.27 seconds, which is acceptable on a resource-constrained mobile device.

Baseline generation. We ran baseline generation based on the input acquired in the online user study. The processing time varies for different participants, while the average time is approximately 289.56 seconds due to the computation complexity of the radial basis function kernel of SVMLight.

Risk measurement. Applying the risk assessment baseline is much faster than baseline generation. We measured the time taken to apply a risk assessment baseline on 14 applications. The average time per application is 0.55 seconds, which is imperceptible and demonstrates the feasibility of repeated risk assessment.

Finally, we anecdotally observed that it took 5-10 minutes for the participants to set relevancy levels for 10 applications. This usability overhead is acceptable compared to the lifetime of a risk assessment baseline.

6 Discussion

To capture actual risks incurred by applications used by a user, RiskMon fundamentally requires running them on the user's device. We note that 48.5 percent of the respondents in our user study claimed that they often test drive applications on their devices. However, RiskMon itself does not detect or prevent sensitive data from leaving users' devices. We would recommend users use on-device isolation mechanisms (e.g., Samsung KNOX).

RiskMon requires users to specify security requirements through permission groups. However, some permission groups are ambiguous (e.g., APP_INFO). Although we identify permission groups as an appropriate trade-off between granularity and usability, we admit that permission groups are still a partial artifact in representing sensitive resources. As our future work, we plan to weight the measured risks of each Binder transaction with protection levels of

permissions. These levels, which pre-classify permissions and Binder transactions, can be considered as a subsidiary source of user's security requirements. We also found that several less sensitive permissions indeed need adjustment to avoid over-estimation. Moreover, generating a risk assessment baseline is a compute-intensive task that does not fit resource-constrained mobile devices. Thus, we plan to offload such a task to trusted third-parties or users' public or private clouds in the future.

Regarding our current implementation of RiskMon, it purposely monitors Binder IPC transactions that are (1) between applications and system services; and (2) between applications and system applications (e.g., Contacts). With that said, RiskMon may discard accesses on assets owned by third-party applications. Meanwhile, custom permissions defined by third-party applications are also beyond the scope of RiskMon. Furthermore, RiskMon identifies Binder transactions with UIDs and thus may not assess the risks incurred by a certain component (e.g., content provider) of an application. For our future work, we will extend the Android ActivityManagerService and PackageManagerService to address these limitations.

7 Related Work

Our previous work [1] mainly dealt with risk assessment of mobile applications. The proposed approach helped reveal high-risk applications. However, it did not consider practical methods to mitigate the identified risks. Since the core functionalities of applications might not be major root causes of risks (e.g., ad-supported games), risk mitigation through removing applications may disrupt user experiences. Instead, it is necessary to seek a novel approach for mitigating risks systematically. In this paper, we extend our previous work to address such a gap by connecting two important steps in RiskMon: risk assessment and risk mitigation. As we have discussed in Section 4, we introduce new mechanisms to automatically select and revoke risky permissions from installed mobile applications, along with newly designed case studies for both steps.

In addition, there exist several related work to the proposed approach in this paper:

Analysis of meta information. Meta information available on application markets provides general descriptions of applications. Recent work has proposed techniques to distill risk signals from them. Sarma et al. [9] propose to analyze permissions alongside with application categories in two large application data sets. Peng et al. [10] use probabilistic generative models to generate risk scoring schemes that assign comparative risk scores on applications based on their requested permissions. In addition to analysis on permissions, Chia et al. [12] and Chen et al. [13] perform large-scale studies on application popularity, user ratings and external community ratings. However, meta information does not accurately describe the actual behaviors of applications. RiskMon uses meta information to provide operational contexts to complement the analysis on the runtime behaviors for risk assessment.

Static and dynamic analysis. Analysis on execution semantics of applications, such as static analysis of code and dynamic analysis of runtime behaviors, can reveal how applications use sensitive information. CHABADA [38] compares an application's static API usage against its description on application markets to detect API "outliers". TouchDevelop [39] statically identifies leaked or tampered information flows to suggest privacy settings for end users. However, malware with dynamic external code loading [40] could easily evade static analysis, rendering all existing static assessment mechanisms ineffective. Regarding dynamic analysis, TaintDroid [4] uses dynamic information flow tracking to detect at most 32 types of flows that are leaked to the network. DroidRanger [15] and RiskRanker [16] combine both static and dynamic analysis to detect anomalies. Compared to RiskMon, these work do not provide a baseline that captures diverse operational contexts as well as a user's expectation. Moreover, RiskMon monitors every permission-protected API and thus provides better coverage.

Mandatory access control frameworks. RiskMon includes a lightweight reference monitor for Binder IPC. While it monitors IPC transactions for risk assessment, several frameworks mediate IPC channels as part of their approaches to support enhanced mandatory access control (MAC). SEAndroid [32] brings SELinux kernel-level MAC to Android. It adds new hooks in the Binder device driver to address Binder IPC. FlaskDroid [33] provides flexible MAC on multiple layers, which is tailored the peculiarity of the Android system. Along these lines, RiskMon captures Binder transactions with a fine-grained scheme to facilitate risk assessment on applications' runtime behaviors.

8 Conclusion

In this paper, we have presented RiskMon that continuously and automatically measures risks incurred by a user's installed applications. RiskMon has leveraged machine-learned ranking to generate a risk assessment baseline from a user's coarse expectations and runtime behaviors of her trusted applications. Furthermore, we have proposed an automated decision process that utilizes RiskMon to support granular permission revocation. Also, we have described a proof-of-concept implementation of RiskMon, along with the extensive evaluation results of our approach.

Acknowledgments

This is an extended and enhanced version of the paper [1] that appeared in ACM CODASPY 2014. This work was partially supported by the grants from National Science Foundation (CNS-0916688) and Global Research Laboratory Project through National Research Foundation (NRF-2014K1A1A2043029).

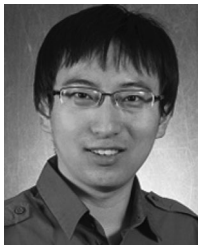
References

[1] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Riskmon: Continuous and automated risk assessment of mobile applications," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, 2014, pp. 99–110.

[2] M. Panzarino, "Google announces 900 million android activations, 48 billion apps downloaded," http://thenextweb.com/google/2013/05/15/google-announces-900-million-activations-of-android-in-total-to-date/, 2013, accessed: Jul. 2013.

[3] A. Robertson, "Apple passes 50 billion app store downloads," http://www.theverge.com/2013/5/15/4327536/apple-passes-50-billion-app-store-downloads, 2013, accessed: Jul. 2013.

[4] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. 9th USENIX Conf. Oper. Syst. Des. Implementation*, vol. 10, 2010, pp. 255–270.
[5] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 639–652.
[6] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in *Proc. 21st USENIX Secur. Symp.*, 2012, p. 29.
[7] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," in *Proc. 3rd ACM Conf. Data Appl. Secur. Privacy*, 2013, pp. 209–220.
[8] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 235–245.
[9] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in *Proc. 17th ACM Symp. Access Control Models Technol.*, 2012, pp. 13–22.
[10] 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," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 241–252.
[11] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative security risk assessment of android permissions and applications," in *Proc. 27th Int. Conf. Data Appl. Secur. Privacy XXVII*, 2013, pp. 226–241.
[12] P. H. Chia, Y. Yamamoto, and N. Asokan, "Is this app safe?: A large scale study on application permissions and risk signals," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 311–320.
[13] Y. Chen, H. Xu, Y. Zhou, and S. Zhu, "Is this app safe for children?: A comparison study of maturity ratings on android and IOS applications," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 201–212.
[14] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *Proc. 22nd USENIX Conf. Secur. Symp.*, 2013, pp. 527–542.
[15] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," in *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2012.
[16] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: Scalable and accurate zero-day android malware detection," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Serv.*, 2012, pp. 281–294.
[17] G. Stoneburner, A. Goguen, and A. Feringa, "Risk management guide for information technology systems," *Nist Special Publication*, vol. 800, no. 30, pp. 800–830, 2002.
[18] C. Alberts, A. Dorofee, J. Stevens, and C. Woody, "Introduction to the octave approach," The Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2003.
[19] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 217–228.
[20] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 627–638.
[21] D. M. Wilkinson, "Strong regularities in online peer production," in *Proc. 9th ACM Conf. Electron. Commerce*, 2008, pp. 302–309.
[22] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010, pp. 73–84.
[23] J. A. Krosnick and D. F. Alwin, "An evaluation of a cognitive theory of response-order effects in survey measurement," *Public Opinion Quarterly*, vol. 51, no. 2, pp. 201–219, 1987.
[24] M. Rabin, "Risk aversion and expected-utility theory: A calibration theorem," *Econometrica*, vol. 68, no. 5, pp. 1281–1292, 2000.
[25] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proc. Eighth Symp. Usable Privacy Secur.*, ACM, 2012, p. 3.
[26] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Proc. 8th Symp. Usable Privacy Secur.*, 2012, p. 1.
[27] T. Joachims, "Optimizing search engines using clickthrough data," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 133–142.
[28] P. Eckersley, "Google removes vital privacy feature from android, claiming its release was accidental," https://www.eff.org/deeplinks/2013/12/google-removes-vital-privacy-features-android-shortly-after-adding-them, 2013, Feb. 2014.
[29] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending android permission model and enforcement with user-defined runtime constraints," in *Proc. 5th ACM Symp. Inform., Comput. Commun. Secur.*, 2010, pp. 328–332.
[30] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Trust and Trustworthy Computing*. New York, NY, USA: Springer, 2011, pp. 93–107.
[31] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android," in *Proc. 19th Netw. Distrib. Syst. Secur. Symp*, 2012.
[32] S. Smalley and R. Craig, "Security enhanced (se) android: Bringing flexible mac to android," in *Proc. 20th Netw. Distrib. Syst. Secur. Symp.*, 2013.
[33] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and fine-grained mandatory access control on android for diverse security and privacy policies," in *Proc. 22nd USENIX Secur. Symp.*, 2013.
[34] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proc. 20th USENIX Conf. Secur.*, 2011, pp. 21–21.
[35] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," in *Proc. USENIX Workshop Hot Topics Secur.*, 2012.
[36] S. Shekhar, M. Dietz, and D. S. Wallach, "Adsplit: Separating smartphone advertising from applications." in *Proc. USENIX Security Symposium*, 2012, pp. 553–567.
[37] K. Duh, "Ranking vs. regression in machine translation evaluation," in *Proc. Workshop Stat. Machine Trans. Association Comput. Linguistics*, 2008, pp. 191–194.
[38] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions" *in ICSE*, 2014, pp. 1025–1035.
[39] X. Xiao, N. Tillmann, M. Fahndrich, J. De Halleux, and M. Moskal, "User-aware privacy control via extended static-information-flow analysis," in *Proc. IEEE/ACM Int. Conf. Automated Software Engi.* ACM, 2012, pp. 80–89.
[40] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications," in *Proc. Netw. Distrib. Syst. Security Symposium*, 2014.



Yiming Jing received the BS degree from Shanghai Jiao Tong University, China, in 2010. He is currently working toward the PhD degree in the School of Computing, Informatics, and Decision Systems Engineering, Ira A. Fulton School of Engineering, Arizona State University. His current research interests include access control models and mechanisms, security and privacy in mobile computing, and secure software engineering.

Gail-Joon Ahn received the PhD degree in information technology from George Mason University, Fairfax, VA, in 2000. He is a professor in the School of Computing, Informatics, and Decision Systems Engineering, Ira A. Fulton Schools of Engineering and the director of Security Engineering for Future Computing Laboratory, Arizona State University. His research has been supported by the US National Science Foundation, National Security Agency, US Department of Defense, and US Department of Energy. He is a senior member of the IEEE.

Ziming Zhao received the PhD degree in computer science from Arizona State University, Tempe, AZ, in 2014. He is a postdoctoral scholar in the School of Computing, Informatics, and Decision Systems Engineering, Ira A. Fulton School of Engineering, Arizona State University. His research interests include hardware security, system security, and usable security. He is a student member of the IEEE.

Hongxin Hu received the PhD degree in computer science from Arizona State University, Tempe, AZ, in 2012. He is an assistant professor in the School of Computing at Clemson University. His current research interests include software-defined networking security, security and privacy in social networks, security in cloud and mobile computing, and secure software engineering. He is a member of the IEEE.