RISKMON: Continuous and Automated Risk Assessment of Mobile Applications

Yiming Jing†, Gail-Joon Ahn†, Ziming Zhao†, and Hongxin Hu‡
†Arizona State University ‡Delaware State University
{ymjing,gahn,zzhao30}@asu.edu, hhu@desu.edu

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 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. 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.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls, Information flow controls*

Keywords

Smartphones; Android; Risk Assessment

CODASPY'14, March 3–5, 2014, San Antonio, Texas, USA.

http://dx.doi.org/10.1145/2557547.2557549.

1. INTRODUCTION

Mobile operating systems, such as Android and iOS, have tremendously supported an application market over the last few years. Google Play announced 48 billion app downloads in May 2013 [27]. Almost at the same time, Apple's AppStore reached 50 billion downloads [31]. 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 a variety of purposes. 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 and when 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 [14, 22, 30, 38, 39]. Specifically, TaintDroid [14] and Aurasium [38] inspect an application's behaviors at variable and syscall level, respectively. 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 since 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. The user needs to provide her selection of trusted applications from the installed applications on her device and her ranking of permission groups in terms of their relevancy to the corresponding application. Then, RiskMon builds the user's risk assessment baseline for different application categories by leveraging API traces of her selected applications. RiskMon continuously monitors the installed applications' 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 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.

As RiskMon interposes and assesses API calls before an application gets the results, we foresee the possibility of integrating RiskMon into an automated permission granting process as discussed in [18] and [32]. Furthermore, while we implement RiskMon on the Android platform, RiskMon is equally applicable to other platforms (e.g. Apple iOS and Microsoft Windows Phone) in assisting security experts to discover high-risk 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 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. Our approach lowers the required sophistication to conduct effective risk assessment for end-users;
- We propose a machine-learned ranking based framework that continuously monitors the runtime behaviors of mobile applications, automatically measures their risks, and intuitively presents the risks;
- 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 problem description of this paper. Section 3 provides a high-level overview of the RiskMon framework and system design by illustrating each stage of automated risk assessment. Section 4 presents prototype implementation and evaluation of our framework. Section 5 discusses the limitations of our approach. Section 6 describes related work. Section 7 concludes the paper.

2. MOTIVATION AND BACKGROUND TECHNOLOGIES

Users are concerned about security and privacy issues on mobile devices. However, in most cases they are not aware of the issues unless highlighted. Although Apple's mandatory application review process [1] and Google Bouncer [25] strive to mitigate misbehaving applications, users are still responsible for defending themselves.

2.1 Use Cases and Threat Model

A continuous and automated risk assessment framework enhances a number of use cases in the current mobile application ecosystems. In general, such a framework improves user experience of security features and promotes understanding about risks of mobile applications. This enables more users to discover misbehaving applications and possibly write negative reviews, thereby alerting and protecting other users. In addition, it complements static and dynamic analysis in ensuring appropriateness justifications by security analysts. This could be applied in both official and alternative application markets as a pre-screening mechanism to select suspicious applications for further analysis. Alternatively, a developer can evaluate her applications against those of her competitors and improve security practices if necessary. For the purposes of this paper we consider the generic scenario where a user assesses her installed applications.

Applications, as long as they are not on users' devices, do not incur any substantial risk. Once an application is installed, it starts interacting with the operating system and other applications. While the application accesses sensitive resources, it gradually builds a big picture of the system as well as the user. Each access, such as calling an API, returns a tiny fraction of the picture and incurs a small amount of risk. Once the picture is finished, it may contain a user's personal identities (e.g. contacts), device identities (e.g. manufacturer) and context identities (e.g. locations, WiFi SSIDs). Since risk assessment at the pre-installation stage does not address such threats on users' devices, we aim to provide continuous risk assessment for normal users.

2.2 Risk Assessment of Mobile Applications

Recent work has proposed mechanisms to extract risk signals from meta information on application markets such as permissions [16, 28, 33, 36], ratings [9, 10], and application descriptions [26]. Their limitation is that such information is fuzzy and fails to provide fine-grained information about how and when sensitive resources are used. For example, an application may stay in the background and keep probing a user's locations and surroundings. Moreover, a malicious application with split personalities [5] can evade screening mechanisms of application markets. We argue that users deserve the rights to understand what is happening on their own devices. Thus, continuously revealing runtime behaviors plays a vital role as a necessary defense line against rogue applications.

Previous research concerning applications' runtime behaviors specifies a set of risk assessment heuristics tailored to their specific problems. For example, TaintDroid [14] considers a case in which sensitive data is transmitted over the network. DroidRanger [40] and RiskRanker [20] assume that dynamically loaded code is a potential sign of malware. While these techniques provide valuable insights about run-

time behaviors of mobile applications, they do not justify the appropriateness of the revealed behaviors. We argue that runtime behaviors are not the only factor to determine appropriateness. Another important factor is the contextual properties. For example, a location-based application has good reason 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. Also, a user's expectations are another critical factor. Even though an application is allowed to use a user's sensitive information, the user should have the capability to specify preferences for determining accesses to her own sensitive information. However, we cannot assume that all users are able to digest contextual information and system-level expectations, which is necessary for establishing a risk assessment baseline that captures appropriate behaviors. The absence of such a baseline renders current risk assessment process ineffective. Therefore, it is imperative to automate risk assessment for seamlessly helping users accommodate their preferences without requiring additional intervention.

2.3 Android Platform

Android is a computing platform for mobile devices. It implements a security architecture that adopts a sandbox and a permission framework. While system services and installed applications are isolated and confined in their respective sandboxes, they can interact and collaborate via APIs. Each permission protects a set of APIs that access some sensitive resources. A user can approve permission requests of an application at install time so that the application is allowed to use the corresponding APIs.

Permission groups: Permission group is a logical grouping of related permissions. For example, `SOCIAL_INFO` includes permissions that access a user's contacts and call logs. Given Android API level 18, Android provides 31 permission groups to cover 134 permissions. 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.

API and direction of control flow: A typical Android application's execution is orchestrated by API calls and callbacks with opposite directions of control flows. An API call initiates a synchronous control flow so that the caller application gets results immediately after the API returns. API callbacks are designed for asynchronous control flows which enable a system service to notify an application when an event occurs or a result is ready. Both API calls and callbacks are frequently used in accessing sensitive resources, such as getting a contact entry and receiving location updates.

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*. Binder includes a kernel driver and a userspace library. 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. Binder identifies a transaction with the UIDs and PIDs of sender and recipient processes as well as a command code that specifies the action to be performed in the receipt process.

3. RISKMON: OVERVIEW AND SYSTEM DESIGN

In this section, we describe our risk assessment framework that lowers the required intervention and sophistication in risk assessment of mobile applications.

IT risk assessment guidelines, such as NIST SP 800-30 [35] and CERT OCTAVE [2], illustrate general methodologies that enable organizations to understand, assess and address their information risks. For example, OCTAVE covers the following critical tasks [3]:

1. Identify critical information assets and their security requirements;
2. Consider the activities that can expose the identified assets to threats due to organizational and/or technological vulnerabilities;
3. Define risk evaluation criteria that captures operational context and organization's tolerance; and
4. Create practice-based protection strategies and risk mitigation plans.

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. Several existing state-of-the-art frameworks attempt to automatically extract a universal risk assessment baseline by mining the meta information of a large number of mobile applications (e.g. Peng et al. [28]). Compared with their approaches, RiskMon adheres to general risk assessment methodologies and considers user's security requirements and operational contexts as indispensable inputs. This design choice enables our framework to accurately capture user's expected appropriate behaviors rather than average practices of developers.

An underlying assumption of RiskMon is that a user's trusted applications could define her expected appropriate behaviors. Recent empirical analysis showed that applications of similar categories normally request a similar set of permissions [6], implying similar core functionalities. Hence, each of the user's trusted applications can be used as a reference point of appropriate behaviors for applications of similar categories. For example, Netflix application is under "Entertainment" category, and Pandora's Internet Radio application is under "Music & Audio" category. Even though they are not in the same category, each application similarly uses one of core functionalities such as the streaming service of personalized media contents from remote servers. If a user trusts Netflix application, it implicitly affirms that Pandora application may also incur commensurate risks caused by Netflix application. Thus, using Netflix application as a reference point, the deviation or "distance" of runtime behaviors between Netflix and Pandora applications indicates Pandora's additional inherent risks.

We now summarize the design goals for a continuous and automated risk assessment framework:

Continuous and fine-grained behavior monitoring: Applications access sensitive resources by calling APIs to communicate with each other and system services. To ensure continuous monitoring on API calls, RiskMon interposes Binder IPC on a user's device. The risks incurred by API calls are determined by the caller, the callee, and the data. To capture such information, RiskMon opts for a fine-grained

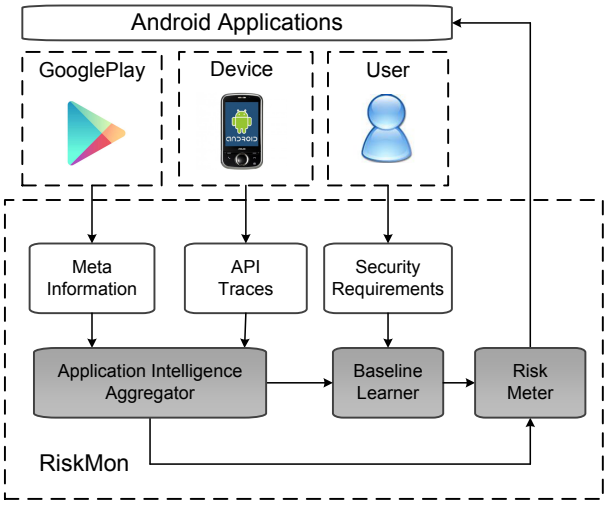


Figure 1: RiskMon Architecture for Android

scheme to capture various intelligences about applications. This provides a well-founded base for measuring the "distance" between two API calls in the space of runtime behaviors.

Simplified security requirement communication: It is a challenging task for users to specify security requirements for security tools. To tackle this problem, RiskMon adopts a simple heuristic that allows users to communicate security requirements through their coarse expectations. Although this reduces the burden on the user, we cannot entirely eliminate it. We note that acquiring a user's expectations is necessary since each user has diverse preferences on the same application. For instance, all users of Facebook application may have disparate expectations for controlling their location and camera utilities.

Intuitive risk representation: The way in which risk is presented significantly influences a user's perception and decision upon risky applications. A counterexample would be standalone risk scores, such as a risk indicator saying "Facebook incurs 90 units of risk" without proper explanation. As Peng et al. noted in [28], "it is more effective to present comparative risk information". Inspired by their approach, RiskMon presents a ranking of applications so that a user can compare the potential loss of using an application with other applications. In addition, the user can view the risk composition of an application for supporting evidences.

Iterative risk management: Risk assessment is an ongoing iterative process. As applications get upgraded and bring more functionalities, they introduce new risks that should be measured. To this end, the risk assessment baseline should evolve to continuously monitor installed applications and update the risk assessment baseline periodically. Moreover, users need to provide their feedbacks to RiskMon by adding or revising their security requirements.

We now present our risk assessment framework. Figure 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 application intelligence aggregator compiles a dataset from API traces collected on a user's device and meta information crawled from application markets. API traces cover an application's interactions with other parts of the system via API calls and callbacks. To complement API traces with contextual information, RiskMon uses meta information on application markets such as ratings, number of downloads and category which provide a quantitative representation of applications' reputation and intended core functionalities. The baseline learner combines a user's coarse expectations and aggregated intelligences of her trusted applications to generate a training set. Afterwards, the baseline learner applies a machine-learned ranking algorithm to learn a risk assessment baseline. Then the risk meter measures how much an application's behaviors deviate from the baseline. Using the deviation to provide risk information, risk meter ranks a user's installed applications by their cumulative risks and presents the ranking to the user in an intuitive way. The remainder of this section describes each component in detail.

3.1 Application Intelligence Aggregator

This component aggregates intelligences about a user's installed applications, including their runtime behaviors and contextual information. As RiskMon monitors runtime behaviors by interposing Binder IPC, we propose a set of features for API traces tailored to the peculiarity of Binder. Also, we seek contextual information from application markets and propose corresponding features to represent and characterize them. The proposed features build a space of application intelligences and enables subsequent baseline generation and risk measurement. Unless explicitly specified, all features are normalized to [0,1] so that each of them contributes proportionally.

3.1.1 Features for API Traces

Android applications frequently use APIs to interact with system services. Considering that using most APIs does not require any permission, we assume that resources protected by at least one permission are a user's assets.

We are interested in runtime behaviors, i.e. Binder transactions, that are used by APIs to reach the assets. However, APIs do not carry information about Binder transactions. To bridge this gap, we adopt existing work [4, 17] to provide mappings from permissions to APIs. Meanwhile, we analyzed the interface definitions of Android system services and core libraries to generate a mapping from APIs to Binder transactions. As a result, we extracted 1,003 permission-protected APIs, of which each corresponds to a type of Binder transactions. Each type of Binder transaction is identified by the corresponding system service, direction of control flow, and a command code unique to the service. For example, an API named `requestLocationUpdate` is identified as Binder IPC transaction `(LocationManager, callback, 1)`.

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 so as 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 and system services. Thus, internal properties are represented with the following features:

- **Type of Binder transaction**: 1,003 boolean features as a bit array, where one bit is set to 1 for the corresponding transaction type and others are 0; and
- **Direction of control flow**: another boolean feature: 0 for transactions initiated from applications (API calls), 1 for transactions initiated from system services (API callbacks).

Note that we use 1,003 boolean features to represent the type of Binder transactions instead of using one integer value. This is because Binder transactions are independent from each other, and the Binder command codes are simply nominal values. By using the array of 1,003 boolean values, the distances between any two Binder transaction types are set to the same value, which is important for our learning algorithm (Section 3.2.3).

In terms of contents, parcels in Binder transactions are unstructured and highly optimized, and it is hard to restore the original data objects without implementation details of the sender and recipient. Therefore, we use length as one representative feature of parcel. 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 contacts database. Thus, 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.

3.1.2 Features for Meta Information

Although meta information on application markets cannot describe applications' runtime behaviors, it is still viable to use such information as contextual properties that capture users' and developers' opinions and complement runtime behavior information.

In terms of representing the opinions of users, we use the following features in correspondence with their counterparts of meta information on application markets:

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

These three features capture an application's popularity and reputation. The first two features are similar to number of views and comments in online social networks. Recent studies [37] demonstrated that online social networks and crowd-sourcing systems expose a long-tailed distribution. Therefore, we assume they follow the same distribution and use the 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.

[1]Number of installs is specified with exponentially increasing ranges: 1+, 5+, . . . , 1K+, 5K+, . . . , 1M+, 5M+.

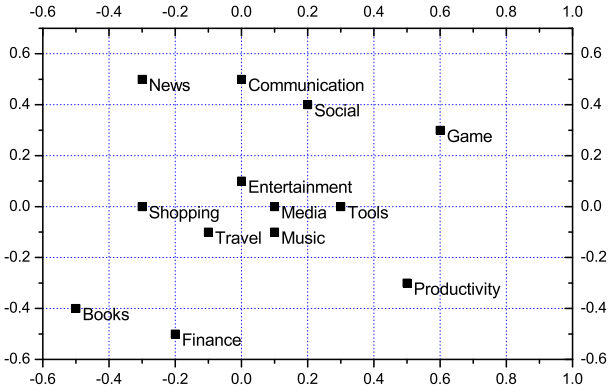


Figure 2: SOM Representation of 13 Categories

Next, we propose a feature to capture the developer's opinion:

- **Category:** a tuple of two numerical values normalized to [-0.5, 0.5].

Google Play uses an application's category to describe its core functionalities (e.g. "Communication"). As of this writing, Google Play provides 27 category types. We choose Self-Organizing Map (SOM) to give a 2-dimension representation of categories. Barrera et al. [6] demonstrated that SOM can produce a 2-dimensional, discretized representation of permissions requested by different categories of Android applications. Categories in which applications request similar permissions are clustered together. Therefore we use the x and y coordinates in the map to represent categories. Figure 2 depicts the coordinates of 13 categories as an example. It is clear to see that some categories bear underlying similarities, such as "Entertainment", "Media and Video" and "Music and Audio" in the center of the figure[2].

Clearly 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 applications. In addition, falsifying an application's meta information violates the terms of application market's developer policies and may lead to immediate takedown.

Finally, based on the scheme defined by these features, the application intelligence aggregator generates a dataset consisted of feature vectors extracted from API traces 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 which 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

[2]For more details on SOM, please refer to [6].

could be mostly acceptable by users. Krosnick and Alwin's dual path model [24] 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.*

This heuristic captures a user's expectations as security requirements by risk aversion, which implies the reluctance of a user to use a functionality with an unknown marginal utility [29]. 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.

Base 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 [12,19]. Although users tend to overestimate the scope and risk of permission groups, they are more intuitive and reduce warning fatigue [19].

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*. Figure 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. $PG = \{pg_1, pg_2, \cdots, pg_m\}$ is a set of permission groups available in a mobile operating system. $A = \{a_1, a_2, \cdots, a_n\}$ is a set of a user's installed applications. TA is a set of a user's trusted and installed applications and $TA \subseteq A$. $RequestedPG : A \rightarrow 2^{PG}$ is a function that maps an application to its requested permission groups. A user's security requirement Req is a mapping $Req : TA \times PG \rightarrow R$. $R = \{1, 2, 3, 4, 5\}$ is a set of relevancy levels, where a larger value indicates higher relevancy and less risk and vice versa.

3.2.2 Compiling Training Set

Next we describe how the baseline learner compiles a training set from the aggregated application intelligences and user-specified relevancy levels. For brevity, we apply the relevancy levels onto the feature vectors generated by the application intelligence aggregator to generate a set of vectors annotated with relevancy levels.

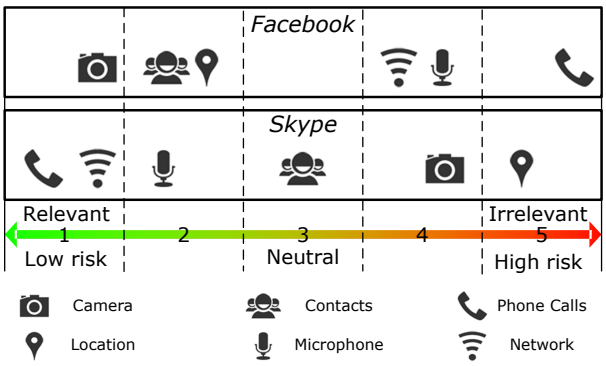


Figure 3: An Example of Specifying Relevancy for Permission Groups

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 [4,17]. 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 as follows. Algorithm 1 illustrates the process to compile the training set T.

- X is a space of features as defined by the scheme discussed in Section 3.1, $X = \{\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_l\}$, $X \in \mathbb{R}^i$, where i denotes the number of features;
- $DS = \{D_{a1}, D_{a2}, \cdots, D_{am}\}$ is a collection of sets of feature vectors, where $D_{aj} \subseteq X$ and D_{aj} corresponds to an application a_j;
- $Apd : A \times PG \rightarrow DS$ is a function that maps an application and one of its requested permission groups to a set of feature vectors; and
- $T = \{(\vec{x}_1, r_1), (\vec{x}_2, r_2), \cdots, (\vec{x}_n, r_n)\}$ is a training set consisted of annotated vectors, $r_k \in R$, $\vec{x}_k \in X$.

Algorithm 1: Compiling Training Set

```
Data: DS, TA, Req
Result: T
T ← ∅;
for a ∈ TA do
    pg ← RequestedPG(a);
    r ← Req(a, pg);
    D ← Apd(a, pg);
    for x⃗ ∈ D do
        add (x⃗, r) to T;
    end
end
return T
```

3.2.3 Generating Risk Assessment Baseline

Ranking Support Vector Machine (RSVM) [21, 23] is a pair-wise ranking method. Generally it utilizes a regular

Support Vector Machine (SVM) solver to classify the order of *pairs of objects*. Next we explain how we apply RSVM to learn a risk assessment baseline.

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

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

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:

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

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 $(\vec{x}_i, \vec{x}_j)$.

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

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

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

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 in the application intelligence dataset:

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

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 those are trusted by the user. Note that (6) gives a signed distance. We use the absolute value to represent the deviation and risk. The risks incurred by an application a_i are the cumulative risks of its runtime behaviors:

$$\sum_{\vec{x} \in D_{ai}} |f_{\vec{w}^*}(\vec{x})| \tag{7}$$

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. Note that the risk meter does not provide any technical explanation at this level.

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 which 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.

API calls and callbacks: The evidences presented at this level are intended for experienced security analysts who are familiar with the security mechanisms under the hood of Android. This is the raw data generated by the risk scoring function. An analyst can inspect values of features to reconstruct the semantic view of runtime behaviors.

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. 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 two case studies. We conclude our evaluation with the usability and performance of our system.

4.1 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 placing hooks inside the Binder userspace library. The hooks tap into Binder transactions and log the parcels with zlog[3] which is a high-performance logging library. In addition, we implemented automated risk assessment based on SVMLight[4] and its built-in Gaussian radial basis function kernel.

We designed and conducted a user study to evaluate the practicality and usability of RiskMon. We hand-picked 10 applications (Table 2) that were mostly 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 intelligences to generate the baselines. We also randomly selected 4 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

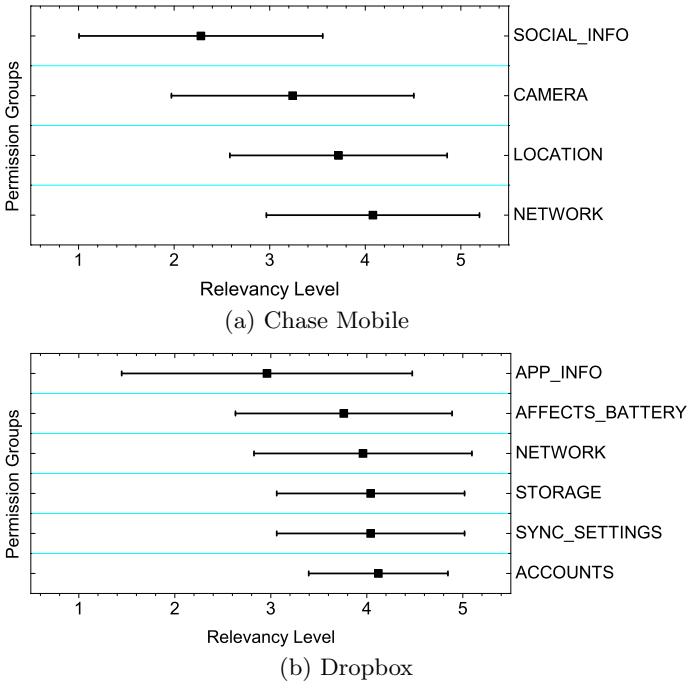
[3] `https://github.com/HardySimpson/zlog`
[4] `http://svmlight.joachims.org/`

Table 1: 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%)

Table 2: 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



(a) Chase Mobile

(b) Dropbox

Figure 4: Average Relevancy Levels Specified by the Participants for Chase Mobile and Dropbox

(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 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. 33 users participated in the study and Table 1 lists the demographics of them.

4.2 Empirical Results

4.2.1 Security Requirements

From our user study shown in Table 2, we highlight the results of Chase Mobile and Dropbox because they both request some ambiguous permission groups that are hard to justify for users. Figure 4 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 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. We can also observe that `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 (e.g. `SOCIAL_INFO` of Chase Mobile) always gets the highest risk scores for both trusted and distrusted applications.

4.2.2 Application Risk Ranking

Figure 5 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 highest risk scores.

Note that both Pandora and CNN are renowned applications developed by well-trained developers. Seemingly, they should use sensitive information appropriately. Hence, we verified them by manually dissecting their API traces. We found that they both stayed in the background and attempted to keep connected to remote servers. To this end, they kept polling ConnectivityManager for a fine-grained state of the current network connection. This is an unexpected practice for both privacy and performance perspectives and the official Android documents suggest developers register `CONNECTIVITY_CHANGE` broadcasts[5] to get connectivity updates accordingly 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.

4.3 Case Studies

In this section we evaluate the effectiveness of our approach. Note that there is no ground truth of user's expected appropriate behaviors. Thus, we opt for two case studies on two applications, SogouInput and PPS.TV. We specified the relevancy levels for 10 trusted applications and generated a risk assessment baseline. Then, we verified their identified risk composition with manual analysis.

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. Table 3 demonstrates the risk composition of SogouInput and PPS.TV by their requested permission groups. First, the unusually large portion of `PHONE_CALLS` indicates significant use of capabilities related to making phone calls and reading unique identifiers. We verified the corresponding API traces and revealed that it attempted to read a user's subscriber ID and device ID. Second and more notably, `SOCIAL_INFO` contributed 4.02% of the total risks incurred by SogouInput. We verified the corresponding API traces 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, SogouInput attempted to dump the whole contacts data 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.

[5] `http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html`

Table 3: Risk Composition by Permission Groups 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%)

Table 4: 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

The results leave much room for imagination: how come an input method and a video streaming application need capabilities related to `PHONE_CALLS`, `LOCATION` and `SOCIAL_INFO`? Possibly users get personalized services by disclosing these information. However it comes with a price of privacy. RiskMon highlights the risks so that users can weigh the benefit and relevant cost by themselves.

4.4 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 risks as presented by RiskMon.

After using RiskMon, an exit survey was presented to collect users' perceived usability of RiskMon. In the survey we asked users questions on *likeability* (e.g. "indicate how much you like using your trusted apps to set a baseline"), *simplicity* (e.g. "do you agree that RiskMon requires less mental efforts in risk assessment"), and *risk perception* (e.g. "do you feel the increased awareness of the risks of your installed applications"). Questions were measured with a five-point Likert scale. A higher score indicates a positive opinion or agreement, while a lower score indicates a negative one or disagreement. Then scores were adjusted to [0,1] for numerical analysis.

We analyzed a 95% 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 4 shows that an average user asserts 79.7% positively on likeability, 64.5% on simplicity and 75.1% on risk perception. The results show usability of RiskMon with the above-average feedback.

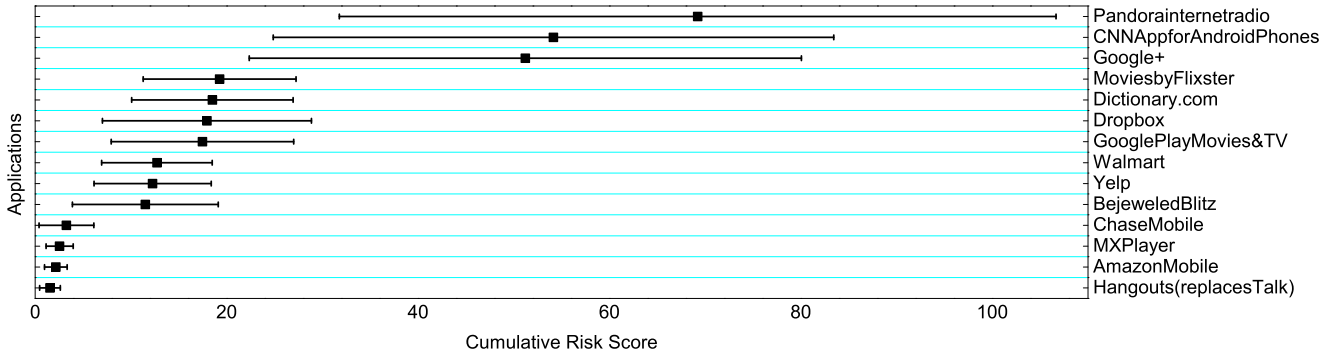


Figure 5: Average Cumulative Risk Scores Measured by the Participants' Risk Assessment Baselines

Table 5: 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

4.5 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 1.2GHz dual-core ARM CPU. The phone runs Android v4.2.2 and RiskMon built on the same version. Table 5 shows the average results.

Feature extraction: The application intelligence aggregator extracted feature vectors from the raw API traces of 33,368,458 IPC transactions generated by 14 applications in one day. We measured the CPU-time used by parsing the API traces 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 SVM-Light.

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.

5. 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% of the respondents in our user study claimed that they often test drive applications on their devices. 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[6]) or data shadowing (e.g. [22]). However, it is far from perfect for running untrusted applications on trusted operating systems.

RiskMon requires users to specify security requirements through permission groups. While most of the frequently requested permission groups are self-descriptive (e.g. `LOCATION` and `CAMERA`), some are ambiguous (e.g. `APP_INFO`) and contain low-level APIs only known to developers. 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 for users. Note that we choose permission groups only to demonstrate the feasibility of our approach of security requirement communication. As our future work, we plan to develop a systematic and intuitive taxonomy of sensitive resources on mobile devices to facilitate more effective requirement communication. Moreover, generating a risk assessment baseline is a compute-intensive task that does not quite 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 does not address: (1) interactions between third-party applications; and (2) interactions that do not utilize Binder. This indeed illustrates potential attack vectors that can bypass RiskMon. Unauthorized accesses on resources of third-party applications [11] might be possible because such resources are not protected by system permissions. Also, two or more malicious applications can collude via local sockets or covert channels and evade the Binder-centric reference monitor in RiskMon. For our future work, we will extend our framework to maximize the coverage of attack vectors in our approach.

6. RELATED WORK

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. Kirin [16] provides a conservative certification technique that enforces policies to mitigate applications with risky permission combinations at install time. Sarma et al. [33] propose to analyze permissions alongside with application categories in two large application

[6]`http://www.samsung.com/global/business/mobile/solution/security/samsung-knox#con02`

datasets. Peng et al. [28] 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. [10] and Chen et al. [9] performed large-scale studies on application popularity, user ratings and external community ratings. In particular, Pandita et al. proposed WHYPER [26] which automatically infers an application's necessary permissions from its description in natural languages. However, meta information does not accurately describe the actual behaviors of applications. RiskMon uses meta information to provide contextual information so as 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. Stowaway [17] extracts API calls from a compiled Android application and reveals its least privilege set of permissions. Enck et al. [15] developed a decompiler to uncover usage of phone identifiers and locations. Pegasus [8] checks temporal properties of API calls and detects API calls made without explicit user consent. TaintDroid [14] uses dynamic information flow tracking to detect sensitive data leaking to the network. Regarding malware analysis, DroidRanger [40] and RiskRanker [20] are systematic and comprehensive approaches that combine both static and dynamic analysis to detect dangerous behaviors. DroidScope [39] reconstructs semantic views to collect detailed execution traces of applications. These work focuses on fundamental challenges for assessing actual risks incurred by applications. However, they do not provide a baseline to capture the appropriate behaviors under diverse contexts of different applications. Thus, their approaches are more intended for security analysts rather than end users.

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 [34] brings SELinux kernel-level MAC to Android. It adds new hooks in the Binder device driver to address Binder IPC. Quire [13] provides IPC provenance by propagating verifiable signatures along IPC chains so as to mitigate confused deputy attacks. Aurasium [38] uses libc interposition to efficiently monitor IPC transactions without modifying the Android platform. FlaskDroid [7] 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.

7. 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. Also we have described a proof-of-concept implementation of RiskMon, along with the extensive evaluation results of our approach.

8. ACKNOWLEDGEMENTS

This work was supported in part by the NSF grant (CNS-0916688). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. We would also like to thank the anonymous reviewers for their valuable comments that helped improve the presentation of this paper.

9. REFERENCES

[1] App review - apple developer. `https://developer.apple.com/support/appstore/app-review/`, 2013.

[2] C. Alberts, A. Dorofee, J. Stevens, and C. Woody. Introduction to the octave approach. *Pittsburgh, PA, Carnegie Mellon University*, 2003.

[3] C. J. Alberts and A. Dorofee. *Managing information security risks: the OCTAVE approach*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

[5] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, and G. Vigna. Efficient detection of split personalities in malware. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2010.

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

[7] S. Bugiel, S. Heuser, and A.-R. Sadeghi. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *22nd USENIX Security Symposium (USENIX Security 2013). USENIX*, 2013.

[8] K. Z. Chen, N. Johnson, V. D'Silva, S. Dai, K. MacNamara, T. Magrino, E. Wu, M. Rinard, and D. Song. Contextual policy enforcement in android applications with permission event graphs. 2013.

[9] 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 *Proceedings of the 22nd international conference on World Wide Web*, pages 201–212. International World Wide Web Conferences Steering Committee, 2013.

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

[11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011.

[12] E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring user confidence in smartphone security and

privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 1. ACM, 2012.

[13] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: Lightweight provenance for smart phone operating systems. In *USENIX Security Symposium*, 2011.

[14] 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 *OSDI*, volume 10, pages 255–270, 2010.

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

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

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

[18] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, D. Wagner, et al. How to ask for permission. In *Proc. USENIX Workshop on Hot Topics in Security*, 2012.

[19] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM, 2012.

[20] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 281–294. ACM, 2012.

[21] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Neural Information Processing Systems*, pages 115–132, 1999.

[22] 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 *Proceedings of the 18th ACM conference on Computer and communications security*, pages 639–652. ACM, 2011.

[23] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[24] J. A. Krosnick and D. F. Alwin. An evaluation of a cognitive theory of response-order effects in survey measurement. *Public Opinion Quarterly*, 51(2):201–219, 1987.

[25] H. Lockheimer. Android and security - official google mobile blog. `http://googlemobile.blogspot.com/2012/02/android-and-security.html`, 2012.

[26] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the 22nd USENIX conference on Security symposium*. USENIX Association, 2013.

[27] M. Panzarino. Google announces 900 million android activations, 48 billion apps downloaded, 2013.

[28] 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 *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM, 2012.

[29] M. Rabin. Risk aversion and expected-utility theory: A calibration theorem. *Econometrica*, 68(5):1281–1292, 2000.

[30] V. Rastogi, Y. Chen, and W. Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220. ACM, 2013.

[31] A. Robertson. Apple passes 50 billion app store downloads, 2013.

[32] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 224–238. IEEE, 2012.

[33] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Android permissions: a perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 13–22. ACM, 2012.

[34] S. Smalley and R. Craig. Security enhanced (se) android: Bringing flexible mac to android. In *Proc. of the 20th Network and Distributed System Security Symposium (NDSS 2013), San Diego, CA*, 2013.

[35] G. Stoneburner, A. Goguen, and A. Feringa. Risk management guide for information technology systems. *Nist special publication*, 800(30):800–30, 2002.

[36] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala. Quantitative security risk assessment of android permissions and applications. In *Data and Applications Security and Privacy XXVII*, pages 226–241. Springer, 2013.

[37] D. M. Wilkinson. Strong regularities in online peer production. In *Proceedings of the 9th ACM conference on Electronic commerce*, pages 302–309. ACM, 2008.

[38] R. Xu, H. Saïdi, and R. Anderson. Aurasium: Practical policy enforcement for android applications. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[39] L. K. Yan and H. Yin. Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[40] 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 *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012.