2012 IEEE 12th International Conference on Data Mining

Mining Permission Request Patterns from Android and Facebook Applications

Mario Frank, Ben Dong, Adrienne Porter Felt and Dawn Song
University of California, Berkeley

***Abstract*—Android and Facebook provide third-party applications with access to users' private data and the ability to perform potentially sensitive operations (e.g., post to a user's wall or place phone calls). As a security measure, these platforms restrict applications' privileges with permission systems: users must approve the permissions requested by applications before the applications can make privacy- or security-relevant API calls. However, recent studies have shown that users often do not understand permission requests and are unsure of which permissions are typical for applications. As a first step towards simplifying permission systems, we cluster a corpus of 188,389 Android applications and 27,029 Facebook applications to find patterns in permission requests. Using a method for Boolean matrix factorization to find overlapping clusters of permissions, we find that Facebook permission requests follow a clear structure that can be fitted well with only five patterns, whereas Android applications demonstrate more complex permission requests. We also find that low-reputation applications often deviate from the permission request patterns that we identified for high-reputation applications, which suggests that permission request patterns can be indicative of user satisfaction or application quality.**

***Keywords*-Unsupervised learning; pattern mining, Smartphones; Permissions; Android; Facebook**

I. Introduction

Open development platforms like Android and the Facebook Platform have resulted in the availability of hundreds of thousands of third-party applications that end users can install with only a few clicks. Consequently, end users are faced with a large and potentially bewildering number of choices when looking for applications. Users' installation decisions have privacy and security ramifications: Android applications can access device hardware and data, and Facebook applications can access users' profile information and social networks. As such, it is important to help users select applications that operate as the user intends.

Android and Facebook use permission systems to control the privileges of applications. Applications can only access privacy- and security-relevant resources if the user approves an appropriate permission request. For example, an Android application can only send text messages if it has the `SEND_SMS` permission; during installation, the user will see a warning that the application can "Send SMS messages" if the installation is completed. These permission systems are intended to help users avoid privacy- or security-invasive applications. Unfortunately, user research has demonstrated that many users do not pay attention to or understand the permission warnings [1], [2]. A major problem here is that users do not know what permission combinations are typical for applications.

Our work is a first step in the direction of simplifying permission systems by means of statistical methods. We propose to identify common patterns in permission requests so that applications that do not fit the predominant patterns can be flagged for additional user scrutiny. Towards this goal, we apply a probabilistic method for Boolean matrix factorization to the permission requests of Android and Facebook applications. We find that while applications with good reputations (i.e., many ratings and a high average score) typically correspond well to a set of permission request patterns, applications with poor reputations (i.e., less than 10 ratings) often deviate from those patterns.

The primary contribution of this paper is the first analysis of permission request patterns with a statistically sound model. Our technique captures the concept of identifying "unusual" permission requests. Our evaluation demonstrates that our technique is highly generalizable, meaning that the found clustering is stable over different random subsets of the data. We find that permission request patterns can indicate user satisfaction or application quality.

II. Background and Related Work

Android and the Facebook Platform support extensive third-party application markets. They use permission systems to limit applications' access to users' private information and resources.

A. Android

The Android Market is the official (and primary) store for Android applications. The Market provides users with average user ratings, user reviews, descriptions, screenshots, and permissions to help them select applications. Android applications can access phone hardware (e.g., the microphone) and private user information (e.g., call history) via Android's API. Permissions restrict applications' ability to use the API. For example, an application can only take a photograph if it has the `CAMERA` permission. Developers select the permissions that their applications need to function, and these permission requests are shown to users during the installation process. The user must approve all of an application's permissions in order to install the application.

Several studies have examined Android applications' use of permissions. Barrera et al. surveyed the $1,100$ most popular applications and found that applications primarily

1550-4786/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDM.2012.86

request a small number of permissions, leaving most other permissions unused [3]. They used self-organizing maps (a dimensionality reduction technique) to visualize the relationship between application categories and permission requests; based on this analysis, they concluded that categories and permissions are not strongly related. Their focus was on visualization and their findings are not applicable to identifying unusual permission request patterns; they relied on the minimization of a Euclidian cost function to find a low dimensional visualization of the data, whereas we use a generative probabilistic model to learn request patterns. Felt et al. and Chia et al. surveyed Android applications and identified the most-requested permissions [4], [5]. Chia et al. also found several correlations between the number of permissions and other factors: a weak positive correlation with the number of installs, a weak positive correlation with the average rating, a positive correlation with the availability of a developer website, and a negative correlation with the number of applications published by the same developer. We expand on these past analyses, and our analysis of permission requests is by far the largest study to date.

Other research has focused on using machine learning techniques to identify malware. Sanz et al. applied several types of classifiers to the permissions, ratings, and static strings of 820 applications to see if they could predict application categories, using the category scenario as a stand-in for malware detection [6]. Shabtai et al. similarly built a classifier for Android games and tools, as a proxy for malware detection [7]. Zhou et al. found real malware in the wild with DroidRanger, a malware detection system that uses permissions as one input [8]. Although our techniques are similar, our goal is to understand the difference between high-reputation and low-reputation applications rather than to identify malware. Applications may be of low quality or act in undesirable ways (i.e., be risky) without being malware. Additionally, our approach only relies on permission requests; unlike these past approaches, we do not statically analyze applications to extract features, which makes our technique applicable to platforms where code is not available (such as the Facebook platform).

Enck et al. built a tool that warns users about applications that request blacklisted sets of permissions [9]. They manually selected the blacklisted patterns to represent dangerous sets of permissions. In contrast, we advocate a statistical whitelisting approach: we propose to warn users about applications that do not match the permission request patterns expressed by high-reputation applications. These two approaches could be complementary; human review of the found patterns could potentially improve them.

B. Facebook

The Facebook Platform supports third-party integration with Facebook. Facebook lists applications in an "Apps and Games" market alongside information about the applications, including the numbers of installs, the average ratings, and the names of friends who use the same applications. Through the Facebook Platform, applications can read users' profile information, post to users' news feeds, read and send messages, control users' advertising preferences, etc. Access to these resources is limited by a permission system, and developers must request the appropriate permissions for their applications to function. Applications can request permissions at any time, but most permission requests are displayed during installation as a condition of installation. Chia et al. surveyed Facebook applications and found that their permission usage is similar to Android applications: a small number of permissions are heavily used, and popular applications request more permissions [4].

III. Application Data Set

In this section, we report about how the data was collected[1]. For a more detailed analysis of the global statistics of the dataset that includes price and ratings distributions, please refer to the author version of this paper [10].

1) Android.: We collected information about 188,389 Android applications from the official Android Market in November 2011. This data set encompasses approximately 59% of the Android Market, which contained 319,161 active applications as of October 2011 [11]. To build our data set, we crawled and screen-scraped the web version of the Android Market. Each application has its own description page on the Market website, but the Market does not provide an index of all of its applications. To find applications' description pages, we first crawled the lists of "top free" and "top paid" applications. These lists yielded links to 32,106 unique application pages. Next, we fed 1,000 randomly-selected dictionary words and all possible two-letter permutations (e.g., "ac") into the Market's search engine. The search result listings provided us with links to an additional $156,283$ unique applications. Once we located applications' description pages, we parsed their HTML to extract applications' names, categories, average rating score, numbers of ratings, numbers of downloads, prices, and permissions.

2) Facebook.: Chia et al. provided us with a set of 27,029 Facebook applications [4]. They collected these applications by crawling SocialBakers [12], a site that aggregates statistics about Facebook applications. After following SocialBakers's links to applications, they screen-scraped any permission prompts that appeared. They also collected the average ratings and number of ratings for each application. One limitation of this data set is that it only includes the permission requests that are shown to users as a condition of installation; they did not attempt to explore the functionality of the applications to collect secondary permission requests that might occur later. As such, our analysis only incorporates the permission requests that are shown to users as part of the installation flow.

[1]The data is available at http://www.mariofrank.net/andrApps/index.html

IV. Pattern Mining Technique

Our goal is to infer statistically significant permission request patterns from the set of all applications' permission requests. Let N be the number of applications, and let D be the total number of possible permissions. We can then represent the dataset of applications' permission requests as a binary matrix $\mathbf{x} \in \{0,1\}^{N \times D}$. The entry $x_{id} = 1$ means that application i requests permission d. The row $\mathbf{x}_{i*} \in \{0,1\}^{D}$ represents all permission requests of application i.

Given this matrix as an input, we want to find the two following binary matrices: the matrix $\mathbf{u}$ that encodes which permissions are frequently requested together (the permission request patterns), and the matrix $\mathbf{z}$ that encodes which applications share the same permission request patterns. Let K be the number of patterns found, then $\mathbf{z} \in \{0,1\}^{N \times K}$ and $\mathbf{u} \in \{0,1\}^{K \times D}$.

These two matrices represent an approximate factorization of the input matrix via the Boolean matrix product, where the Boolean product $\mathbf{c} = \mathbf{a} \otimes \mathbf{b}$ of two matrices $\mathbf{a} \in \{0,1\}^{N \times K}$ and $\mathbf{b} \in \{0,1\}^{K \times D}$ is defined as (see [13]): $c_{id} = \bigvee_{k=1}^{K} (a_{ik} \wedge b_{kd})$.

Having introduced these concepts, we can rephrase our goal as finding a factorization $(\mathbf{z}^*, \mathbf{u}^*)$ that approximates the permission request matrix $\mathbf{x}$ up to such residuals that exhibit no statistically significant pattern, i.e., $\mathbf{x} \approx \mathbf{z}^* \otimes \mathbf{u}^*$.

We employ a probabilistic model for binary matrix factorization [13]. This model takes a binary matrix $\mathbf{x}$ and computes the likelihood that a given factorization $(\mathbf{z}, \mathbf{u})$ represents the statistically significant patterns of $\mathbf{x}$. We then tweak the entries of $(\mathbf{z}, \mathbf{u})$ to maximize the likelihood of $\mathbf{x}$. The outcome is a factorization $(\mathbf{z}^*, \mathbf{u}^*)$ that does not necessarily provide an exact representation of $\mathbf{x}$. This model was originally derived as an approach to the role mining problem [14], [15], where the goal is to identify roles to configure role-based access control (RBAC) [16]. Sets of permission requests can be equated to roles.

The likelihood function in [13] explicitly models the data as a probabilistic mixture of signal and noise. In the context of mining permission requests, signal corresponds to significant patterns of permission requests, and noise corresponds to the residuals when fitting these patterns to the data. Each permission request of each application is assumed to follow either the signal distribution $p_S(x_{id} \mid \mathbf{z}, \boldsymbol{\beta})$ with probability $(1-\epsilon)$, or a random Bernoulli distribution $p_N(x_{id} \mid r)$ with probability ϵ. The signal distribution is

$$p_S(x_{id} \mid \mathbf{z}, \boldsymbol{\beta}) = \left[1 - \prod_{k=1}^{K} \beta_{kd}^{z_{ik}}\right]^{x_{id}} \left[\prod_{k=1}^{K} \beta_{kd}^{z_{ik}}\right]^{1-x_{id}}$$

Here, the Boolean permission requests u_{kd} assigning permission d to pattern k are modeled by the probability that they are 0, i.e. $\beta_{kd} := p(u_{kd} = 0)$. Assuming that the individual entries x_{id} are independent, the model is a modified Bernoulli distribution $B(x_{id}|q_{id})$ with Bernoulli parameter $q_{id} = \prod_{k=1}^{K} \beta_{kd}^{z_{ik}}$.

The noise distribution is a Bernoulli distribution $p_N(x_{id} \mid r) = r^{x_{id}} (1-r)^{1-x_{id}}$. This means that if an application's permission request x_{id} is generated from the noise distribution, then it is sampled from a biased coin flip and with probability r application i requests permission d.

Finally, the complete likelihood function is a mixture of the noise probability distribution and the signal probability distribution:

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\beta}, r, \epsilon) = \prod_{i=1}^{N} \prod_{d=1}^{D} (\epsilon p_N(x_{id}|r) + (1-\epsilon) p_S(x_{id}|\mathbf{z}, \boldsymbol{\beta}))$$

The parameters $(\mathbf{z}, \boldsymbol{\beta}, r, \epsilon)$ of this distribution can be optimized by an annealed expectation-maximization algorithm [13]. This algorithm alternates between updating the individual parameters and, after each iteration, reducing the computational temperature, ultimately forcing the probabilistic parameters $\beta_{kd} \in [0,1]$ towards 0 or 1. The algorithm terminates when the parameter updates become negligible or when a predefined temperature is reached.

The number of patterns K must be provided as an input of the algorithm. There are several related heuristics for this model-order selection problem [17], [18] and we approach it by carrying out an instability analysis [17]. An introduction to this method and the results of our analysis can be found in the author version of this paper [10]. Following the results of this analysis, we selected $K = 30$ patterns to fit the Android dataset and $K = 5$ patterns to fit the Facebook dataset.

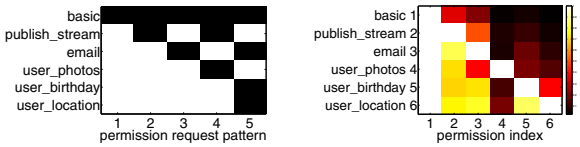
V. Experiments

We mine patterns of permission requests from high-reputation Android and Facebook applications. Section V-A details our methodology and provides an overview of the permission request patterns that we found. We then consider how permission request patterns differ between high- and low-reputation applications (Section V-B) and find that the patterns can be used as part of a risk metric for new, unknown applications.

A. Overview of the Permission Request Patterns

We apply the techniques described in Section IV to the permission requests of high-reputation Android and Facebook applications.

1) Methodology.: We train our model on *high-reputation applications*, defined as applications with average ratings of 4 or higher and at least 100 user ratings. Our reputation metric combines average ratings and the number of reviews because the average rating by itself is an unreliable measure. This quality criterion yields 11,554 Android applications and 1,998 Facebook applications. For Android, we reserved $2,000$ applications to use as a test set and trained on the remaining 9,554applications. For Facebook, we reserved 400 test applications and trained on 1,598 applications.



(a) PRPs (b) Pairw. Cond. Probablities

Figure 1: PRPs and pairw. cond. probabilities for Facebook. Each column in Fig. 1a is a PRP; a black entry in the matrix means that the permission is part of the PRP; the patterns are sorted from left to right with decreasing popularity. Fig. 1b: bright colors indicate a high conditional probability.

2) Results.: The Facebook dataset has a simple and clean structure. We found five permission request patterns (PRPs) for Facebook, which suffice to cover the requested permissions of most Facebook applications. Figure 1a depicts the five PRPs. This simple structure might be due to the fact that the permissions requested after installation are not accounted for in the data, as explained in Section III.

The Android dataset is more complex: our mining technique identified 30 PRPs for Android. The most common PRPs for Android are shown in Figure 2. Figures 1a and 2 are the (transposed) matrices $\mathbf{u}^*$ that assign permission patterns to permissions. We sorted the patterns by frequency such that PRP1 is the most common pattern, PRP2 is the second most-requested pattern, etc.

PRPs are not disjoint: permissions can be members of multiple patterns, and applications can request multiple patterns. Consequently, most patterns only include a small number of permissions. A PRP with only one permission reflects the fact that the permission is requested very frequently, but not always together with the same other permissions.

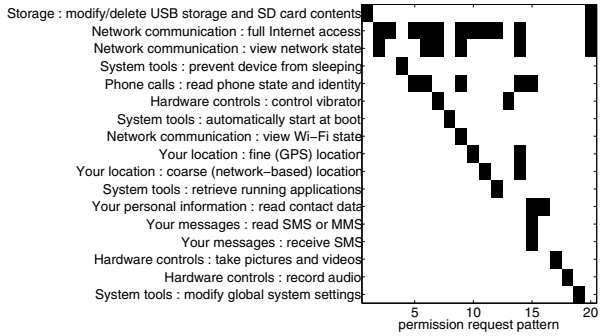


Figure 2: The permissions included in the 20 most popular Android permission request patterns (PRPs). The fraction of applications requesting the PRP decreases from left to right.

In order to explain why particular permissions appear in the same pattern, we can consider the pairwise conditional probabilities. For two permissions s and t, the conditional probability is empirically estimated as

$$p_{st} := p(x_s = 1 | x_t = 1) = \left(\sum_{i=1}^{N} x_{it} \right)^{-1} \sum_{i=1}^{N} x_{is} x_{it}$$

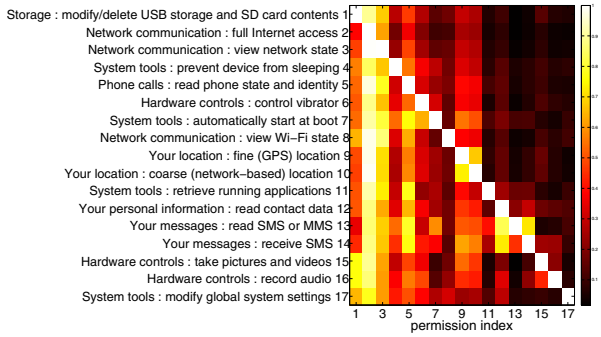


Figure 3: Pairwise conditional probabilities for the Android permissions from the 20 most popular patterns.

If $p_{st} = 1$, then s is requested whenever t is requested. This score is not symmetric, i.e., $p_{st} = p_{ts}$ is not necessarily true. We plot these scores for Android and Facebook in the heatmaps in Figure 1b and Figure 3, respectively. The brighter the color, the higher the conditional probability. It is apparent that pairs of permissions with a high conditional probability often end up in the same pattern. We found that clusters of permissions with a high pairwise conditional probabilities cannot emerge by chance [10].

3) Evaluation.: We evaluate how well the permission request patterns generalize and fit the data by considering the false positive and false negative rates. Ideally, the patterns should yield low false positive and false negative rates for both the training sets and the test sets. A *false positive* occurs when an application is assigned to a PRP without having all of the permissions associated with the PRP, and a *false negative* occurs when an application's permission requests are not covered by any of the PRPs that the application is assigned to. Consequently, we define the false positive rate f_p as the average number of permissions that are incorrectly assigned to applications, and the false negative rate f_n as the average number of permissions per application that are not covered by PRPs. We depict the cumulative false positive and false negative rates in Figures 4 and 5. (Please note that we cut off the y-axis at different values to best resolve the dynamic range for each experiment.) Our evaluation focuses on the error rates for the training set and high-reputation test set in Figures 4 and 5; we discuss the error rates for the low-reputation test set in Section V-B.

Among the Facebook applications, 2% of high-reputation applications have at least one false positive ($f_p > 0$), and just under 20% of all high-reputation applications have false negatives ($f_n > 0$). Thereby, the residuals are very small: there are almost no applications with more than one false positive ($f_p > 1$) and only 7% with more than one false negative ($f_n > 1$). The error rates are higher for Android applications, which reflects the greater complexity of the Android dataset. Approximately 20% of high-reputation applications have at least one false positive, and 35% have false negatives. For both platforms, the false negative rate is significantly higher

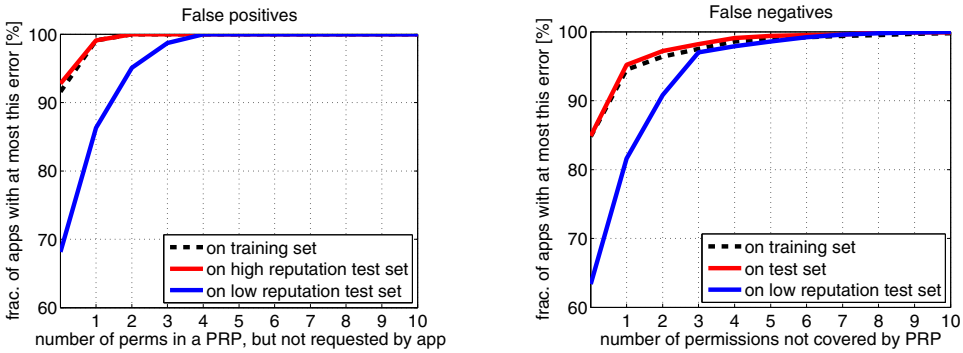


Figure 4: Fraction of Facebook applications with a particular error rate.

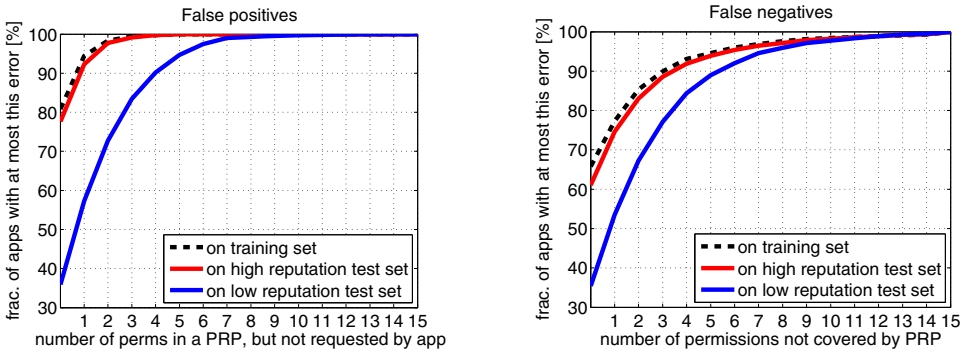


Figure 5: Fraction of Android applications with a particular error rate.

than the false positive rate. The false negative rate can be partially attributed to the very large number of infrequently-requested permissions; these unpopular permissions cannot be placed in PRPs, so the applications that request them cannot be fully fitted with PRPs.

For both platforms, the error rate for the high-reputation test set closely matches the error rate for the high-reputation training set. This demonstrates that we have likely discovered the true statistical structure of the permission requests for high-reputation applications.

B. Reputation and Risk

Our goal is to determine whether permission request patterns can help identify low-quality applications that users should be cautious about installing. We trained on high-reputation applications with the assumption that high-reputation applications are trustworthy, high-quality applications. If this is true, PRPs could be incorporated into a risk metric for new applications: new applications could be considered riskier or lower-quality if their permission requests cannot be fitted to high-reputation PRPs.

We can evaluate the suitability of PRPs for a risk metric by comparing the false positive and false negative rates for the high-reputation training set, the high-reputation test set, and a low-reputation test set that includes applications with fewer than 10 user ratings (regardless of the score). Figures 4 and 5 display all three datasets. We find that low-reputation applications significantly differ from high-reputation applications in how they request permissions, as evidenced by the false positive and negative rates. While the reputable test applications have an error rate that is close to the training error, the unpopular applications have significantly higher residuals, both false positives and false negatives. This does not indicate that the low-reputation applications are fraudulent, but it does suggest that permission requests can be used to help classify an application as high- or low-reputation. Consequently, we recommend that permission request patterns be used as part of a risk metric for newly-uploaded applications. For example, this could be incorporated into a search result ranking algorithm.

VI. Discussion

How could this information be conveyed to users? Applications that do not fit high-reputation PRPs could be ranked lower in search results, unless they receive enough favorable reviews to override the risk metric. Another possibility is to alter the permission request UI so that permissions that deviate from PRPs are highlighted, so that the user's attention is focused on the "unusual" permission requests. This would help provide the user with a relative notion of risk. However, such a design would need to be subject to user research.

Is this technique suitable for detecting malware? We do not aim to provide a malware detector. Applications

may be untrustworthy, low-quality, buggy, or otherwise risky without being malware. We found that applications that receive favorable reviews from large numbers of users request permissions differently than applications with few ratings. Consequently, we designed our approach to identify user satisfaction rather than application maliciousness. However, recently Peng et al. have successfully turned a similar model-based approach into a binary classifier by simply thresholding their risk metric [19]. While our risk metric is the generalization error of predicted permissions of new applications, they use the negative log-likelihood of the requested permissions given the learned model parameters.

How indicative for the quality of apps are permissions? The difference between high-reputation and low-reputation applications is significant, which we attribute to a difference in application quality. However, other factors could at least partially influence the results. The low-reputation applications could be newer applications, and permission request patterns might change over time. This could be accounted for by computing PRPs for chronologically-similar applications. Some of the low-reputation applications might become high-reputation applications in the future; if they were excluded, the difference between the two sets might be more pronounced. Additionally, low-reputation applications might be highly specialized, leading both to exceptional permission requests and few interested users. However, we believe that a real difference in applications is the most likely and predominant reason for the disparity in ratings.

VII. Conclusion

We used a probabilistic model to mine permission request patterns from Android and Facebook applications. For both platforms, we found a set of patterns that fits well to the data. We found that the permission requests of low-reputation applications differ from the permission request patterns of high-reputation applications. This indicates that permission request patterns can be used as part of a risk metric or soft prediction of the quality of new applications. For Android, we find that there is a relationship between permission request patterns and categories. In future work, we will extend the analysis with application categories in order to achieve greater precision.

Acknowledgment

This research was supported by Intel through the ISTC for Secure Computing and by the Swiss National Science Foundation (SNSF), grant no. 138117.

References

[1] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," Tech. Rep.

[2] J. King, A. Laminen, and A. Smolen, "Privacy: Is there an app for that?" in *Proceedings of the Symposium on Usable Privacy and Security*, ser. SOUPS, 2011.

[3] D. Barrera, H. G. u. c. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *ACM CCS '10*.

[4] 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 World Wide Web Conf.*, ser. WWW, 2012.

[5] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the USENIX Conf. on Web Application Development*, ser. WebApps, 2011.

[6] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. Bringas, "On the automatic categorisation of android applications," in *Proceedings of the 9^{th} IEEE Consumer Communications and Networking Conf. (CCNC)*, 2012.

[7] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *Proceedings of the 2010 International Conf. on Computational Intelligence and Security*, 2010.

[8] 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 *NDSS'2012*.

[9] W. Enck, M. Ongtang, and P. McDaniel, "On Lightweight Mobile Phone Application Certification," in *ACM CCS'09*.

[10] M. Frank, B. Dong, A. P. Felt, and D. Song, "Mining permission request patterns from android and facebook applications (extended author version)," *CoRR*, vol. abs/1210.2429, 2012.

[11] L. Horn, "Report: Android market reaches 500,000 apps," http://www.pcmag.com/article2/0,2817,2395188,00.asp.

[12] SocialBakers – Apps on Facebook, http://www.socialbakers.com/facebook-applications.

[13] M. Frank, A. P. Streich, D. Basin, and J. M. Buhmann, "Multi-assignment clustering for Boolean data," *Journal of Machine Learning Research*, vol. 13, pp. 459–489, Feb 2012.

[14] M. Kuhlmann, D. Shohat, and G. Schimpf, "Role mining – revealing business roles for security administration using data mining technology," in *ACM SACMAT'03*, pp. 179–186.

[15] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: A formal perspective," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, pp. 27:1–27:31, Jul. 2010.

[16] D. F. Ferraiolo and D. R. Kuhn, "Role Based Access Control," in *15th National Computer Security Conf.*, 1992, pp. 554–563.

[17] T. Lange, V. Roth, M. L. Braun, and J. M. Buhmann, "Stability-based validation of clustering solutions," *Neural Computation*, vol. 16, pp. 1299–1323, 2004.

[18] M. Frank, M. Chehreghani, and J. M. Buhmann, "The minimum transfer cost principle for model-order selection," in *ECML PKDD '11*. Springer, 2011, vol. 6911, pp. 423–438.

[19] 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 *ACM CCS*, 2012.