Manilyzer: Automated Android Malware Detection through Manifest Analysis

Stephen Feldman
Undergraduate
University of Virginia
1980 Arlington Blvd Apt. I
Charlottesville, VA 22903
Email: feldman@email.virginia.edu

Dillon Stadther
Undergraduate
Gardner-Webb University
110 S. Main Street P.O. Box 5295
Boiling Springs, NC 28017
Email: dstadther@gardner-webb.edu

Bing Wang
Professor
University of Connecticut
371 Fairfield Way, Unit 2155
Storrs, CT 06269
Email: bing@engr.uconn.edu

***Abstract*—As the world's most popular mobile operating system, Google's Android OS is the principal target of an ever increasing mobile malware threat. To counter this emerging menace, many malware detection techniques have been proposed. A key aspect of many static detection techniques is their reliance on the permissions requested in the AndroidManifest.xml file. Although these permissions are very important, the manifest also contains additional information that can be valuable in identifying malware, which, however, has not been fully utilized by existing studies. In this paper we present *Manilyzer*, a system that exploits the rich information in the manifest files, produce feature vectors automatically, and use state-of-the-art machine learning algorithms to classify applications as malicious or benign. We apply Manilyzer to 617 applications (307 malware, 310 benign) and find that it is very effective: the accuracy is up to 90%, while the false positives and false negatives are both around 10%. In addition to classifying applications, Manilyzer is used to study the trends of permission requests in malicious applications. Through this evaluation and further analysis, it is clear that malware has evolved over time, and not all malware can be detected through static analysis of manifest files. To address this issue, we briefly explore a dynamic analysis technique that monitors network traffic using a packet sniffer.**

***Index Terms*—Android security, data mining, malware detection, manilyzer**

I. Introduction

With the expansion of computing has come the desire to have access to the world's information on the go through mobile, hand-held devices. Smartphones and tablets allow their users to engage in communication (e.g. social media, phone calls, SMS, email, etc.), entertainment (e.g. games, music, movies, news, etc.), travel (e.g. GPS, food and hotel search, etc.), and business (e.g. shopping, banking, investments, etc.). In 2013 alone, nearly 760 million devices running Google's Android OS were sold to end users [8]. Android-powered devices make up 78.4% of the smartphone market share [8]. The second most common mobile platform is Apple's iOS, with approximately 150 million units sold in the same year (15.6% of the market share) [8]. It is evident that mobile devices of all kinds are quickly becoming ubiquitous and pervasive.

While mobile devices have grown in popularity among consumers, they have also become very attractive to malicious application developers. Malicious applications, or malware, are applications which negatively affect the function of a device, steal sensitive data, or attain unconsented control of said device. The wide usage of mobile devices generates an ever increasing amount of sensitive personal data, all of which is vulnerable to loss or theft. Between 2012 and 2013, the number of unique malware samples that were detected has increased by 300%; these samples were found in nearly 4 million application installation packages [9]. The incentives for writing malware range from trivial amusement and spam to profitable data theft and ransom [11].

The safety and security of mobile devices and their data has become a dire necessity, and consequently many mobile malware detection techniques have been developed. There are two broad methods of malware detection - static analysis and dynamic analysis. Static analysis, as the name suggests, consists of methods which do not require an application to be executed (i.e. code review). In contrast, dynamic analysis is a behavioral study of an application at runtime. Each technique has its own advantages and disadvantages. While static analysis is the most energy efficient and simplest to implement, malware developers have discovered techniques to make their applications more difficult to detect using this method. On the other hand, dynamic analysis has the potential to provide more thorough detection due to its runtime analysis. It is, however, very resource intensive and is difficult to implement and automate.

For a mobile malware detection technique to be effective, it must be automated, adaptive, and energy efficient. Detection should be a persistent task; therefore, it ought not require heavy user-interaction, but rather be a series of processes which can be completed automatically. Also, the technique needs to be able to detect a breadth of various families of malware, as well as adapt to many variations of a single malware. Detection of zero-day malware is also very important, as anti-malware software must guard against previously undiscovered threats. Lastly, mobile devices have a limited battery life which can be easily wasted during malware detection; therefore, mobile malware detection techniques must require minimal resource consumption.

Though these requirements challenge the ease of creating an effective mobile malware detection system, we have developed a static analysis system which satisfies all three requirements of an effective malware detection technique. Our contributions are the following.

- We developed *Manilyzer*, an automated disassembly and static analysis tool. An extensive evaluation using 617 applications (307 malware, 310 benign) indicates that the accuracy of Manilyzer is up to 90% while the false positives and false negatives are both around 10%.
- We investigated the evolution of permission usage in malicious applications using latest malicious applications. We found that the number of requested permissions is no longer a valuable indicator for malware detection. Further, the increased usage of certain permissions indicates that malware is evolving to avoid static detection.
- We proposed dynamic network traffic analysis as a supplemental approach to detect malicious applications which evade static analysis. Through small-scale experiments, we found that communications between suspicious IP addresses can be captured and analyzed using a packet sniffer, which can provide insights for dynamic malware detection.

The remainder of the paper is organized as follows. Section II provides necessary background information on Android OS and the make-up of an application file. Section III presents *Manilyzer*, our mobile malware detection system. Section IV shows our experimental results and their implications for mobile malware detection. Section V introduces and briefly explores network traffic analysis as a dynamic supplement to *Manilyzer*. Section VI describes related work. Finally, Section VII concludes the paper and presents avenues for future work.

II. Background

Android applications are built with four primary components — activities, services, receivers, and intents. Activities can be thought of as the "screens" with which the user interacts; they directly control the graphical interface. Services execute tasks in the background. Receivers accept messages broadcast from the Android OS and other sources. Intents are packaged messages sent between the aforementioned components [3].

Android applications are distributed in an application package file (APK), which is simply an archive file. The APK file is composed of the program source code, the Android manifest file, and some additional resources. The program is written in Java, translated into Dalvik bytecode, and stored as a Dalvik EXecutable (DEX) file. The AndroidManifest.xml describes the contents of the APK and includes information on the activities, services, receivers, and requested permissions of an application (which are required to access device functionality) [3].

Google has implemented several security mechanisms to help protect the Android platform from mobile malware threats. One of these security measures is mandatory application sandboxing. Every Android application executes in its own system process inside of a Dalvik virtual machine [3]. This restricts direct access to the underlying Linux kernel (which forms the core of the Android operating system) and limits most application-to-application communication [3].

Another security mechanism is Android's permission system. It protects sensitive features of the hardware, OS, and API by not allowing an application access to those features without requesting permission and receiving approval from the user. This gives the user more information about the applications they download and gives them an opportunity to judge their trustworthiness [3].

III. Methodology

The primary goal of Manilyzer is to effectively detect malware automatically through static analysis of Andriod manifest files. Its design is motivated by the analysis results of a rich set of manifest files, for both malware and benign applications. Specifically, we collected 307 malware samples and 310 benign applications. The malware samples were acquired from Contagio Mobile [4], a popular malware minidump that has gathered an extensive set of mobile malware, dating from 2011 to the present. Our benign applications were downloaded from the Google Play store, selecting at least 10 free and popular applications from each of the 27 available categories [6]. Gathering a truly representative dataset is a difficult task with many challenges. Although there is a seemingly endless supply of benign applications, there are comparatively few easily obtainable malicious applications. Further, malware is often designed to mask their malicious behavior and masquerade as a legitimate application, making the precise identification of malware types increasingly complex. On the other hand, by collecting a large number of applications, including the latest malware from Contagio Mobile and a wide variety of benign applications, we believe our dataset provides a reasonable basis for us to understand the characteristics of malware and benign applications in order to develop an effective malware detection tool.

Manilyzer is a tool designed to disassemble APK files, analyze the manifest files, and generate feature vectors describing each application's characteristics. Several third-party software packages are utilized in this tool [5], [2], [1]. Through a manual inspection, we determined that the following were potentially significant characteristics of the AndroidManifest.xml in malware samples:

- *Permission requests*: We observed that some permissions were frequently requested by malware.
- *High priority receivers*: We hypothesized that applications requesting an intent filter with a high priority may be malicious, especially when associated with SMS or boot completed receivers. High priority receivers of these types may be employed by malware to expedite the process of delivering and executing their malicious payloads.
- *Low version numbers*: We observed that malicious applications tended to have lower version numbers compared to benign applications.

- *Abused services*: For security reasons, Google's Android API Guide advises against declaring intent filters for services [3]. After observing multiple violations of this recommendation by an adware family in our dataset, we hypothesized that malicious applications might generally exhibit this characteristic.

For each application, the information contained in its manifest file is represented as a feature vector. Manilyzer automatically analyzes the feature vectors of all the applications in our dataset, and classifies applications as malicious or benign. Specifically, Manilyzer uses multiple machine learning algorithms, including Naive Bayes, Support Vector Machine (SVM), K-Nearest Neighbors algorithm (KNN), and C4.5 decision tree algorithm, for classification. These algorithms are implemented using *Weka*, an open-source suite of machine learning tools [13], which also includes an automatic procedure for feature selection. In addition to classifying applications, we also use the feature vectors corresponding to our dataset to identify trends in permission usage.

IV. Results & Discussion

Our experimental results and their implications will be elaborated upon within the following subsections. Section IV-A will discuss valuable features for malware detection and classification. Section IV-B will elaborate on the effectiveness of *Manilyzer*. Section IV-C presents preliminary results on classifying malware into more specific categories. Section IV-D will examine the evolution of requested permissions.

A. Features for Malware Classification

We confirmed our hypothesis that malware tends to have lower version numbers in comparison with benign applications - 247 malicious applications were labeled with a 1.x application version number, while only 84 benign applications exhibited this characteristic. We also confirmed that SMS receivers with a high priority intent filter were closely associated with SMS malware - 88% of applications with this characteristic were malicious.

We found that malicious and benign applications frequently ignore Android's recommendation to avoid the use of intent filters within services (as previously mentioned in Section III) - thus invalidating our prediction that this characteristic of the manifest file could be valuable in identifying malware [3]. Our preliminary analysis showed a correlation between service abuse and adware malwares. However, after further investigation it became obvious that this feature is only successful in identifying a specific family of adware samples and is generally not a good indicator of adware. *Manilyzer* flagged 47 malware samples as adware with a 38% false positive rate (117/310). 55 of the false positives offered in-application purchases, in-application advertisements, or shopping markets, but all appeared to be legitimate. One of the false positives utilized an advertising network and exhibited suspicious characteristics, but the application itself appeared benign. As there is controversy over the exact definition of adware, any classification as such will be somewhat cumbersome.

As can be seen from the aforementioned results, there is no single feature or "silver bullet" that can perfectly identify malware. However, a combination of features creates a feature vector or "malicious fingerprint" which can be effectively used for malware detection. Therefore, *Manilyzer* classification is based on the feature vector as a whole rather than on one feature in particular.

B. Effectiveness of Manilyzer

In order to determine the effectiveness of *Manilyzer*, four supervised machine learning algorithms, Naive Bayes, SVM, KNN, and C4.5, were applied to the feature vectors. We choose to use the following formulas to evaluate the effectiveness of each algorithm on our dataset. For these equations, let TP (true positive) be the number of malicious applications correctly classified, TN (true negative) be the number of benign applications correctly classified, FP (false positive) be the number of benign applications incorrectly classified, and FN (false negative) be the number of malicious applications incorrectly classified.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$False\ positive\ ratio = \frac{FP}{TP + FP} \quad (2)$$

$$False\ negative\ ratio = \frac{FN}{TN + FN} \quad (3)$$

Figure 1 shows the accuracy results of four algorithms using 10 fold cross-validation.

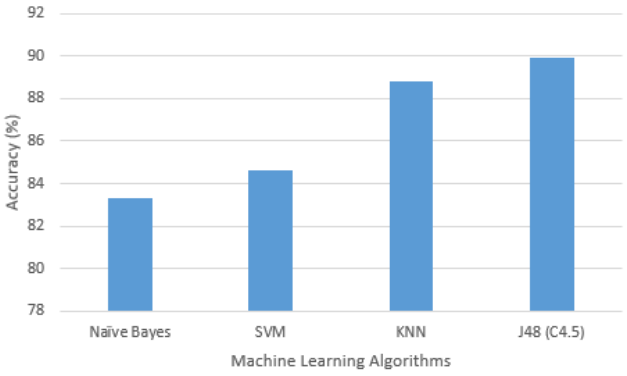


Fig. 1: Accuracy of four machine learning techniques for Android malware detection.

We found that *Weka*'s implementation of the C4.5 decision tree algorithm, J48, yielded the best accuracy with approximately 90% samples correctly classified. Further analysis shows there is a negligible difference between *Manilyzer*'s ability to correctly classify malware and benign applications. Using the formulas previously mentioned, the false positive ratio is 10.1% and the false negative ratio is 10%. While these ratios may seem practically the same, it must be noted that the ratios for benign applications are slightly better while

Classification		Ground Truth
Malicious	Benign	
276	31	Malicious
31	279	Benign

TABLE I: Confusion matrix for dataset.

also having a slightly larger sample size. Table I shows the confusion matrix for our dataset, illustrating these results.

Although *Manilyzer* correctly classifies approximately 90% of the dataset, it is less effective than other work [16], [14]. Our evaluation and survey of previous research indicates that examining requested permissions is supplemented and enhanced by inspecting the API calls associated with each permission [16], [14]. The drawback from not including API calls is the inability to connect a requested permission to the API calls requiring them.

C. Further Classification of Malware

We recognized that *Manilyzer* could be used to effectively classify malware samples into more specific categories, such as adware, spyware, and SMS malware (malware which abuses SMS capabilities of the device). For our dataset, we cannot easily determine their type without manually analyzing each malicious sample in our dataset. Given the large size of the dataset, this is a very time consuming task. As a preliminary study, we opt to use a rule-based approach to classify malware into sub-categories. We were most successful in identifying features generally exclusive to SMS malware through the following rules (representing combinations of features), instead of using classification algorithms:

1) RECEIVE_SMS && WRITE_SMS [10]
2) SEND_SMS && WRITE_SMS [10]
3) High Priority SMS Receivers
4) High Priority SMS Receivers && (Rule 1||Rule 2)

If an application contained the characteristics described in one of the aforementioned rules, it was flagged as SMS malware. Using these rules, we were able to flag 72 malware samples with a 4% (12/310) false positive ratio. The false positives fell into three distinct categories of applications - messaging, voice command, and system utility applications. We noted that the legitimate system utility applications requested a high number of permissions within a given range, therefore we identified an upper and lower threshold in order to exclude applications of this type. By establishing these thresholds, we reduced the false positive ratio to 2%, while maintaining the detection of the 72 malware samples previously found. Although there are benign applications which utilize a high priority SMS receiver, the aforementioned data shows that in practice these are relatively few in number and with some additional care can be safely overlooked. We cannot provide a detection ratio, because we do not know the specific categorization of the malware in our dataset. Further study on this topic is left as future work.

D. Evolution of Requested Permissions

Through our study of permission usage, several trends are documented and discussed in the following subsections. These trends display notable differences from previous research [18].

1) Permission Density: As shown in Figure 2, the average number of permissions requested by malware is 9 while the average number for benign applications is 11. We observed a negligible difference between the number of permissions requested by malware and benign applications, with malware requesting slightly fewer. This finding is a significant change from [18], which reported the average number of requested permissions to be 4 for benign applications and 11 for malware. We conclude that given the evolution of Android applications in the last several years, the number of permissions requested by an application is no longer a valuable indicator for malware detection.

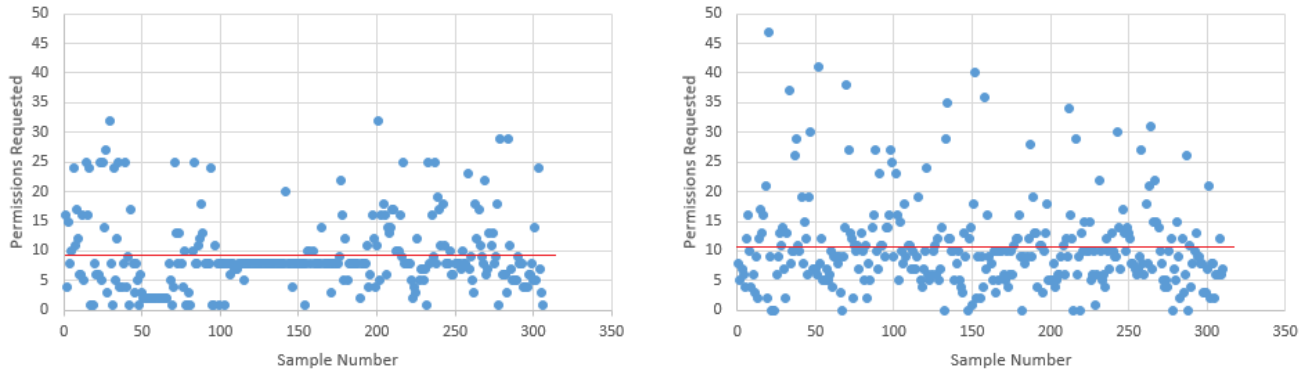
2) Common Permissions: Figure 3 displays the most requested permissions for our data set. While many of the most commonly requested permissions remained consistent with an earlier study [18], our study identified evolutionary trends of requested permissions, differing from [18] in that our malware samples range from their initial release in early 2011 to the present. Among the most requested permissions were some which were only requested by malware. In particular, the "Install/Delete Packages" permissions were requested by approximately one-third of the malware samples and not at all by benign applications. Additionally, we observed that Android's documentation specifically notes that many permissions are "not for use by third-party applications" [3]. In spite of this, many malicious applications and even some which are benign ignore this warning. We conclude that requests for the permissions "Install/Delete Packages", which allows malware to silently install malicious code (perhaps via an "update" mechanism), are on the rise.

Android OS does not grant third-party applications the ability to install or delete software packages even when requesting their corresponding permissions, unless the application has gained root access to the device [3]. Rooting a device is a process which provides the user elevated administrative-level control and allows access to the root, or kernel, of the OS [7]. An application may request the permissions "Install/Delete Packages" in order to install malicious code at runtime and thereby avoid static detection. Therefore, the use of these permissions provide a further indication of malicious intent.

V. Network Traffic Analysis: A Supplement to Manilyzer

As seen through the evolution of permission requests, malware is becoming more difficult to detect through static analysis techniques. This trend shows the growing importance for dynamic analysis as a supplement to static analysis.

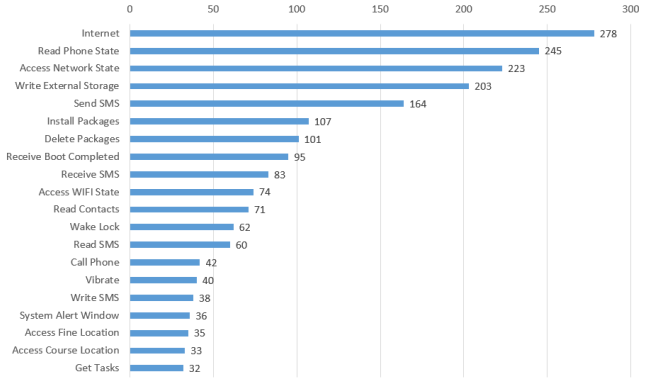
Through initial experimentation, we have concluded that network traffic analysis may be a promising dynamic detection technique. Packet sniffers, such as *Wireshark* or the *Shark for Root* Android application, can be used to capture and analyze network data sent and received from a device. Such a system



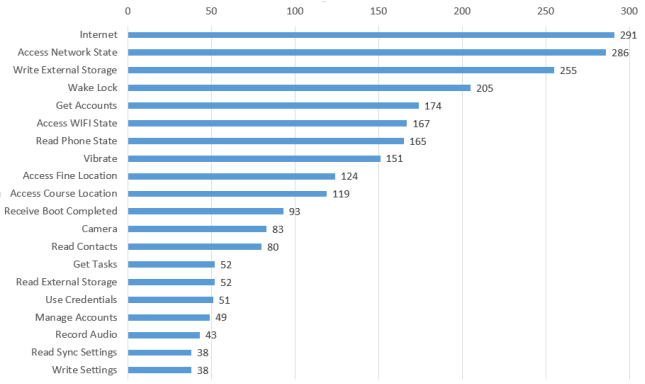
(a) Malware

(b) Benign Applications

Fig. 2: Number of permissions requested. The red line denotes the approximate average number of permissions requested. One malware sample with 68 permissions was omitted to improve visual clarity.



(a) Malware



(b) Benign Applications

Fig. 3: Top permissions requested in malware and benign applications in our dataset.

would act as a supplement to *Manilyzer* by adding additional features to the feature vector for classification.

We conducted several case studies using *Wireshark* and the official Android Emulator to examine the behavior of three malicious applications - HGSpy, Simplocker, and a Minimob variant. As seen through these studies, network traffic analysis may be an effective supplement to *Manilyzer* for malware detection. The following subsections provide descriptions of each case study and their preliminary results.

A. *HGSpy*

HGSpy is spyware, SMS, and bot malware which, once executed, cannot be removed from the infected device. This application did not generate much network traffic. It established a connection to an IP, sent an HttpPost, and terminated the connection. Analysis of this HttpPost showed that its payload contained encrypted line-based text data of the media type "application/x-www-form-urlencoded".

B. *Simplocker*

Simplocker is file-encrypting, tor-based ransomware. Upon execution, the application encrypts files on the host device's memory card and demands a ransom to be paid in order to decrypt them. Little can be gathered from its network traffic, aside from one HttpPost to check.sanasecurity.com, containing device information of the media type "application/x-www-form-urlencoded". Therefore, we began to explore individual IP addresses. Analysis of the source and destination IP addresses revealed numerous tor-related addresses, as well as the known malicious addressed used for the HttpPost.

C. *Minimob*

Minimob is a repackaged version of BlueArt (a live wallpaper application) which spams the user with advertisements. Unlike the previous two applications, this application can be executed without harming the device. Network analysis showed numerous HttpGet requests, each corresponding to an intrusive advertisements.

VI. Related Work

Android malware detection techniques are an increasingly popular topic of research. Many of these rely on permissions requested in the AndroidManifest.xml file. DroidMat [16]

focuses on using attributes of the manifest to trace API calls requiring permissions. [10] proposes a rule-based security mechanism designed to prevent malware at install-time. [14] explores the use of machine learning algorithms for malware detection using permissions and API calls. Our research differs from these works due to its focus on feature vector construction using attributes of manifest, receiver, and service tags, in addition to permission requests, to detect malware.

Other research focuses on using permissions to shed insight on the behavior of applications. [18] discusses various characteristics of Android malware, including their use of permissions. Permlyzer [17] studies the usage of permissions through static and dynamic analysis, without regard to their malicious potential. However, our work studies the use of permissions for malware detection, while also uncovering evolutionary trends in permission usage among malicious applications.

The studies in [12], [19], and [15] focus on efficient, scalable, and accurate malware detection on large Android markets. DroidRanger [19] implements a dual static and dynamic detection system, utilizing permissions as well as receivers. The study in [15] also offers a network traffic monitor (NTM) tool to flag suspicious behavior during runtime and shows that network traffic detection is an effective malware detection technique. Our work, which also demonstrates efficient, scalable, and accurate malware detection, identifies current trends of malicious applications.

VII. Conclusions & Future Work

In this paper, we presented *Manilyzer*, an automated disassembly and static analysis tool for Android applications. Our system accurately classifies up to 90% of the applications in our sample. Based on a survey of prior research in the field of Android malware detection, this is the most recent study involving the evolution of permission usage by malicious applications. Lastly, we explored the use of network traffic analysis as a supplement to *Manilyzer*.

Manilyzer's detection capabilities would be improved with the addition of static analysis using API calls. The ability to detect evolving malware threats may be enhanced with the inclusion of the previously proposed network traffic analysis. Finally, a study regarding malwares ability to avoid detection by *Manilyzer* would be valuable.

Acknowledgments

The first two authors were funded by the NSF sponsored Trustable Computing Systems REU program at the University of Connecticut (UConn). We would like to thank Dr. Luqiao Zhang, UConn visiting scholar, for his suggestions and helpful discussion on our project. We would also like to thank the anonymous reviewers for their insightful comments.

References

[1] 7-zip.
[2] Androguard.
[3] Android developer documentation. Accessed 13 July 2014.
[4] Contagio mobile.
[5] dex2jar.
[6] Google play app store.
[7] Rooting your android. Accessed 16 July 2014.
[8] Gartner says annual smartphone sales surpassed sales of feature phones for the first time in 2013, 2014. Accessed 15 July 2014.
[9] Mobile malware evolution: 3 infection attempts per user in 2013, February 2014. Accessed 17 July 2014.
[10] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proc. of the 16th ACM conference on CCS*, pages 235–245. ACM, November 2009.
[11] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proc. of ACM CCS Workshop on Security and Privacy in Smarphones and Mobile Devices (SPSM)*, pages 3–14. ACM, October 2011.
[12] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. In *Mobile Systems, Applications, and Services (MobiSys)*, pages 281–294. ACM, June 2012.
[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *ACM Knowledge Discovery and Data Mining (SIGKDD) Explorations*, 11, 2009.
[14] N. Peiravian and X. Zhu. Machine learning for android malware detection using permission and api calls. In *Proc. of IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 300–305. IEEE, November 2013.
[15] X. Su, M. Chuah, and G. Tan. Smartphone dual defense protection framework: Detecting malicious applications in android markets. In *Mobile Ad-hoc and Sensor Networks (MSN)*, pages 153–160. IEEE, December 2012.
[16] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia JCIS*, pages 62–69. IEEE, August 2012.
[17] W. Xu, F. Zhang, and S. Zhu. Permlyzer: Analyzing permission usage in android applications. In *Proc. of IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 400–410. IEEE, November 2013.
[18] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proc. of IEEE Symposium on Security and Privacy (SP)*, pages 95–109. IEEE, May 2012.
[19] 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 *Network and Distributed System Security Symposium (NDSS)*, February 2012.