Smartphone Applications, Malware and Data Theft

Lynn M. Batten, Veelasha Moonsamy and Moutaz Alazab

Abstract The growing popularity of smartphone devices has led to development of increasing numbers of applications which have subsequently become targets for malicious authors. Analysing applications in order to identify malicious ones is a current major concern in information security; an additional problem connected with smart-phone applications is that their many advertising libraries can lead to loss of personal information. In this paper, we relate the current methods of detecting malware on smartphone devices and discuss the problems caused by malware as well as advertising.

Keywords Android · Application · Malware · Advertising

1 Introduction

Although many operating systems offer applications (APPs), most are not open source. In this paper, we focus on Android as it is an open source system and therefore easier to test than a proprietary system. None-the-less, much of the discussion here applies to other operating systems too, including iOS, Blackberry and

L.M. Batten (✉)
School of IT, Deakin University, Melbourne, Australia
e-mail: lmbatten@deakin.edu.au

V. Moonsamy
Radboud University Nijmegen, Nijmegen, The Netherlands
e-mail: veelasha@cs.ru.nl

M. Alazab
Faculty of Information Technology, Isra University, Amman, Jordan
e-mail: moutaz.alazab@iu.edu.jo

M. Senthilkumar et al. (eds.), *Computational Intelligence, Cyber Security and Computational Models*, Advances in Intelligent Systems and Computing 412, DOI 10.1007/978-981-10-0251-9_2

Windows. APPs can be a great source of convenience, but they also bring problems connected with malicious code as well as data theft. We shall consider both of these aspects in this paper.

Android applications are available to users as a zipped file, identified as the Android application package, or APK file, which contains classes.dex, assets, library and resource files along with the AndroidManifest.xml which is a configuration file containing the list of permissions and the application's components including activities, services, intent receivers, and content provider, layout data and the application resources. The executable code for the APP resides in classes.dex and in the library file. In particular, the classes.dex file contains all the Java classes compiled to Dalvik Byte code. (For more details see for example http://developer.android.com/sdk/installing/studio-build.html).

The Android permission system is used to protect a smartphone's resources (see http://developer.android.com/guide/topics/security/permissions.html). Each application declares a list of permissions needed to protect access to resources (for example 'permission to access the Internet' is a common one). While the permission system helps prevent the intrusion of malware, it is not designed to detect malicious applications. In fact, several research papers have shown that some applications with no permissions at all can still access the operating system [1]. Brodeur [2] developed a no-permission application that gathers user information and forwards it to a pre-selected server, while the authors of [3] demonstrate a no-permission application that can reboot an Android device. Moreover, several papers (for example [4] and its references) consider the risk of over-privileged applications which request permissions that are not required for the application to execute. Hence, analysing such applications based on only the requested permissions can bias the analysis results.

Malware detection is an emerging topic in the study of the Android platform which relies heavily on its permission system to control access to restricted system resources and private information stored on the smartphone. However, there is no evidence providing a clear understanding of the key differences for permissions between clean and malicious applications.

Mobile phones provide tracking services for several reasons, both for convenience in assisting a person to find a location and in order to offer services such as information about a favourite restaurant which is nearby: so location identification can offer assistance with directions to a target destination and also in indicating facilities along the way. However, several publications (e.g. [5]) have been able to show that sensitive information, such as device ID and user location, is often leaked via advertising libraries.

In this paper, we present some of the recent work on malware detection on Android APPs and also on 'data leaks' from APPs; this latter refers to information taken from the smartphone without the knowledge of the user. Section 2 looks at malware while Sect. 3 considers leaky APPs. In Sect. 4, we describe what is generally required in setting up experiments and tests on APPs. This is followed by a brief summary and then references.

2 Malware

In addition to permissions, a second security mechanism for smartphones is traditional anti-malware analysis, which uses a pattern matching technique to identify malicious applications based on a byte-string that is unique to an application. Recent research work, for example [6], points out that such a signature is comprised of a package or file name that can be matched against an anti-malware database. Nevertheless, work by [7] and others, confirms that malicious applications can evade the current protection mechanisms by using one or more of the following techniques: null operation (NOP) insertion, arithmetic and branch insertion, Java reflection, Byte code encryption, Junk Code Insertion, payload encryption, native exploits or changing the package name. While the pattern matching technique is popular with anti-malware companies due to the high accuracy with which it identifies malicious applications in real time and with low run time, it has been shown to be ineffective in detecting sophisticated malware. According to a study conducted on ten anti-malware applications for Android [6], none of the evaluated applications is resistant against malware transformation techniques including poly-morphism, obfuscation and anti-reversing attacks.

A different approach to identifying malicious applications has been to leverage information from system calls dynamically. However, the authors of [8] argue that monitoring and intercepting system calls is inefficient in Android because system calls are basic interfaces provided by an operating system, and they are the only entrance to kernel mode from user mode; nevertheless, some malware do not necessarily make use of the system call interface and so can evade this analysis method [9]. A second problem is that it is difficult to identify behaviour with system calls. Thirdly, monitoring and intercepting system calls in real Android devices is not possible, as the kernel of such a device cannot use loadable kernel modules. In the literature, techniques used to identify malicious applications include permission-based detection, signature-based detection and system calls-based detection. There have even been attempts to apply machine learning algorithms on the smartphone devices. Some of these algorithms are mentioned below and more information about them can be found in [10].

Google deploys an automated system, known as "*Bouncer*" to test uploaded applications for malicious code. Once a developer has uploaded an application to the Google market, Bouncer compares uploaded applications with known malware and then runs the uploaded application in a virtual environment in order to identify any potential malicious behaviour. Nevertheless, some malicious applications targeting the Android market can evade Bouncer according to a report published by TrendMicro in 2012 [11].

A study by Amamra et al. [12] investigated the effectiveness of machine learning classifiers in detecting malware. The authors collected a dataset of 100 free applications from the Android market and 90 malicious applications from the Contagio mobile dump. They focused on leveraging information from system calls to identify malicious applications, and initially evaluated their framework using the

following algorithms: Logistic Regression, support vector machine (SVM), artificial Neural Network and Naive Bayes. They achieved 92.5 % detection accuracy with an 8.5 % false positive rate using the SVM classifier.

Zhao et al. [13] developed a tool they call RobotDroid to detect smartphone malware during runtime. RobotDroid logs the intent issued and system resources accessed by applications; it then categorizes the logs and sorts them based on the timestamp. Their tool employs the SVM classifier to find those support vectors which are best able to identify malicious applications. The tool was tested using three malware families, lGeinimi, DroidDream and Plankton, and obtained 93.3, 90 and 90 % detection accuracy respectively.

Sahs and Khan in [14] collected a dataset of 2081 benign applications and 91 malicious applications and extracted permissions and Control Flow Graphs using Androguard to train a classification SVM. The authors mention (Section VII) that their system is limited to just permission and control flow graphs, and that other information-rich features can be extracted from the code itself, including constant declarations and method names.

Aung and Zawi [15] applied machine learning algorithms to the information retrieved from permissions and events. The authors applied information gain [16] on the given features in order to improve the detection accuracy and efficiency. In order to test their methodology, they included 3 machine learning algorithms: J48, Random Forest and CART, and obtained 89.36, 91.67 and 87.88 % accuracy respectively.

Amos et al., the authors of [17], developed a framework called STREAM to enable large-scale validation of mobile malware machine learning classifiers. They extract information using dynamic analysis about the battery, binder, memory, network, and permissions from a dataset of 408 benign applications and 1330 malicious applications. For the purpose of testing their framework, they include 6 machine learning algorithms: Random Forest, Naive Bayes, Multilayer Perceptron, Bayes net, Logistic and J48, and obtained 70.31, 78.91, 70.31, 81.25, 68.75 and 73.4 % detection accuracy respectively.

In [18], the authors are the first to examine behaviour in malicious applications using DroidBox. Using a dataset comprising samples that were collected from publicly available sources, each malicious application is executed for 60 s in a sandboxed environment and the log files generated are collected at the end of execution. Droidbox also generated two types of graphs (behaviour graphs and treemap graphs) for each sample. Both graphs helped the authors analyze the activities performed during run-time and also assisted in establishing patterns between variants from the same malware family. These graphs illustrate how some benign applications might leak data connected to short message service (SMS) texting and other features of the applications. The authors note that, while one would expect to see encrypted code in malware, not a single malicious application in their (small) sample set invoked a cryptographic activity, while several of the clean APPs did so.

Finally, it is worth noting that according to the authors of [19, 20], the detection accuracy and efficiency of any machine learning system are influenced by three main factors: the features used to represent the instances; the algorithm used to generate the classifier; and the parameter values of a classifier.

3 Applications Which Leak Data

3.1 Malware and APPs

Our 2012 paper [18] explains that the Android applications market has been infected by numerous malicious applications and that rogue developers are injecting malware into legitimate market applications which are then installed on open source sites accessible to consumers. We thus consider the situation of malware à propos APPs in this sub-section.

In [18], we demonstrated that Droidbox [21] can be a useful tool both in classifying malicious Android applications and in determining weaknesses in benign Android applications; weaknesses include the leaking of private data caused by such functionalities as location services and advertising and we consider these in the next sub-section. DroidBox can track sensitive data originating from the phone's database and add and modify output channels to detect leaks via outgoing SMS and to disclose full details of the network communication. Android applications can perform phone calls or send SMS to premium rate numbers that are declared by the attacker. DroidBox can disclose these operations, and is able to track sensitive data originating from the phone's database and detect leaks via outgoing SMS as well as disclose full details of the network communication.

Some malicious Android applications can evade anti-virus software by performing obfuscation and changing themselves during run-time [15]. DroidBox is designed to detect applications which attempt obfuscation by using cryptographic keys to encrypt or decrypt data; however, as noted at the end of the previous section, the use of cryptography is not necessarily an indication that an APP is malicious.

3.2 Tracking Services

We turn to a discussion of two types of tracking features for smartphones; these are: (i) Location Services and (ii) Advertising. For (i), smartphone owners can either turn on or turn off location tracking to prevent installed applications from discovering their physical locations. As for advertising, users are allowed to either turn on or 'limit' tracking by advertising libraries embedded in applications.

The Android permission-based model is used to restrict access to privileged system resources and to a user's private information. This is achieved by requiring the user to grant access to all permissions requested by the application in order for it to be successfully installed. Consequently, any advertising libraries embedded in an APP receive the same privileges as the APP that requested the permissions.

Pearce et al. [4] proposed a framework that can separate an advertising library from its main application. They introduced a new Application Programming Interface (API) as well as two additional permissions and applied a method known as privilege separation, which extracts the advertising component from the main functionality component of the application; this ensures that the advertising library does not inherit the same permissions assigned to its home APP. In [22], Shekhar et al. presented their method for separating applications and advertisements in the Android platform: a framework that can take as input an APP with embedded libraries and rewrite it so that the main functionality of the APP and the advertising libraries run as different processes. The authors also verified that, in the rewritten version of the APP, all the permissions requested by it were indeed required for the APP to function properly.

Stevens et al., in [23], performed a thorough analysis of third party advertising libraries to understand if they are unnecessarily accessing private information stored on users' smartphones. Additionally, the authors presented several vulnerabilities that attackers can exploit whilst being connected on the same network as the victim. Grace et al. [24] observed that some third-party advertising libraries employ unsafe mechanisms to retrieve and execute code from the Internet. Such behaviour renders a user's private information vulnerable to external attacks that can be carried out via the Internet.

The authors of [25] investigated tracking services on the Android and iOS smartphone platforms and described a simple and effective way to monitor traffic generated by tracking services to and from the smartphone and external servers. As part of the testing, they dynamically executed a set of Android and iOS applications, collected from their respective official markets. Their results indicate that even if the user disables or limits tracking services on the smartphone, applications can by-pass those settings and, consequently, leak private information to external parties. On the other hand, when testing the location 'on' setting, the authors notice that generally location is not tracked.

Two of the authors of [25] collaborated with additional researchers to investigate the same problem on other smartphone operating systems [26]. Using the experimental software platform Mallory, which was also used in [25], the authors investigated the 'tracking off' settings on the Blackberry 10 and Windows Phone 8 platforms in a manner similar to that used for Android and iOS and with similar results. The conclusion is that tracking settings on all four smartphone operating systems Android, iOS, Blackberry and Windows cannot be trusted to operate as proposed.

4 Experimental Work

In this section, we give a general explanation of how APP samples can be collected and tested for malware and for leaks. Any APP should first be tested to determine whether it is benign or malicious; this can be done using an existing free online service, as described in the next sub-section. The existence of this service also permits us to define malicious and clean APPs rigorously.

4.1 VirusTotal

VirusTotal (https://www.virustotal.com/), a subsidiary of Google, is a free online service that analyzes files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners. It may also be used to detect false positives, that is, benign code detected as malicious by one or more scanners.

VirusTotal stores a list of identifiers (signatures) of known malware, which are contributed by many antivirus companies; this list is updated about every 15 min as signatures are being developed and distributed by antivirus companies. An APP can be uploaded onto their website where scanning is done via API queries to the approximately 50 different companies providing VirusTotal with information.

We define a *malicious APP* as one which is identified to be so by *at least one* of VirusTotal's antivirus products. We define a *clean* or *benign APP* to be one not identified as malicious by *any* of the VirusTotal antivirus products.

4.2 Setting up the Testing Environment

This section describes the experimental setup for most of the work done by the authors in the various publications [2, 7, 19–22, 26].

Smartphone malicious applications can be collected from several open source sites such as Contagion, Offensive and VXHeavens, while benign applications can be chosen from obvious sources such as: adobe_flash_player, official APP markets, antivirus, facebook, googlemaps, mobi and youtubedownloader. In all cases APPs should be checked by VirusTotal as described above.

For all of our testing, we used Windows supported by Linux and set up a virtual machine environment to separate the application from the network as mobile malware can be spread from mobile device to PC and vice versa.

In order to identify APPs and detect any changes during the testing, we used HashMyFiles installed in the Windows host to generate a unique identifying hash value. In doing dynamic testing (that is, executing the application to determine what it does), a decision about the running time has to be made. With a small set of

samples, a tester may be able to allocate a longer time to each sample than available when using a large set. Whatever the time chosen, if the APP is designed to behave in a malicious way after the allocated run time has expired, this behaviour will not be detected in the testing.

4.3 Performance Evaluation

The standard measure of success in machine learning is the overall accuracy [27], which is defined as the percentage of all applications classified correctly. Research papers studying malware usually work with multi-classes in which each category has many more benign than malicious applications. In this (imbalanced) case, the accuracy measure may not be an adequate performance metric [28]. For example, if a classifier correctly identifies the entire dataset as benign, the classifier achieves high accuracy results while failing to detect the malicious applications.

In order to adequately reflect categorization performance, there are more accurate metrics that can be used in imbalanced cases, including 'recall' and 'precision', which can be combined into 'F-measure' [29, 30]. These metrics would normally be calculated on each class separately and then combined together to provide a weighted average.

5 Summary

Advertising has developed as the solution to the lack of a business model associated with the provision of free APPS for smartphones, as it allows application developers to offer free applications to the public while still earning revenue from in-application advertisements. Although location services are primarily used for purposes related to navigation, advertising companies tend to exploit this functionality in order to increase their revenue. Thus, advertising is unlikely to disappear from smartphones in the near future.

In this paper, we have explained how malware can be installed on APPs offered through the unofficial APP markets, and subsequently downloaded on to many smartphones with results such as theft of user identity and contacts, and fraudulent use of the smartphone for expensive calls. We have also explained how advertising can be used to capture and track user identity.

Based on these observations, the authors of [25] suggest the following (paraphrased) recommendations for three relevant parties:

1. *Novice Smartphone Users*. Download applications only from the official markets as they are less likely to be malicious than APPs from unofficial markets. While one cannot guarantee that all applications found on official markets are

clean, there is always a chance for any malicious applications to be deleted from the market when reported to the designated authorities.
2. *Device Manufacturers*. Smart-device manufacturers can provide users with pre-installed applications so that they have more control of their private information, instead of relying on the smartphone operating system.
3. *Academia/Industry*. Researchers from academia and industry within the field form an open-source research community to develop open-source applications that will help to compensate for the security vulnerabilities found in existing applications offered by the official application markets.

References

1. Moonsamy, V., Batten, L.M.: Zero permission android applications—attacks and defences. In: 3rd Applications and Technologies in Information Security, pp. 5–9. School of Information Systems, Deakin University Press, Australia (2012)
2. Brodeur, P.: Zero-permission android applications Part 2. Publication of Leviathan Security Group; accessed August 7, 2015, http://www.leviathansecurity.com/blog/zero-permission-android-applications-part-2/ (2012)
3. Lineberry, A., Richardson, D.L., Wyatt, T.: These aren't the permissions you're looking for, 2010. https://www.defcon.org/images/defcon-18/dc-18-presentations/Lineberry/DEFCON-18-Lineberry-Not-The-Permissions-You-Are-Looking-For.pdf
4. Pearce, P., Felt, A. P., Nunez, G., Wagner, D.: Android: privilege separation for applications and advertisers in android. In: 7th ACM Symposium on Information, Computer and Communications Security, pp. 71–72. ACM Digital Library, Arizona, USA (2012)
5. Moonsamy, V., Alazab, M., Batten, L.M.: Towards an understanding of the impact of advertising on data leak. Int. J. Secur. Networks, **7**(3), 181–193. Inderscience Publishers, London, England (2012)
6. Rastogi, V., Chen, Y., Jiang, X.: DroidChameleon: Evaluating android anti-malware against transformation attacks. In: 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2013), pp. 329–334. ACM Digital Library, Arizona, USA (2013)
7. Park, Y., Lee, C. Lee, C., Lim, J., Han, S., Park, M. Cho, S.: RGBDroid: a novel response-based approach to android privilege escalation attacks. In: 5th USENIX conference on Large-Scale Exploits and Emergent Threats (LEET'12), 8 pp. Berkeley, California, USA (2012)
8. Peng, G., Shao, Y., Wang, T., Zhan, X., Zhang, H.: Research on android malware detection and interception based on behavior monitoring. Wuhan Univ. J. Nat. Sci. **17**, 421–427 (2012)
9. Egele, M., Scholte, T., Kirda E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. **44**(2), 6. Arizona, USA (2012)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explorations, **11**(1). http://www.cs.waikato.ac.nz/ml/weka/ (2009)
11. TrendMicro, Repeating History. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-repeating-history.pdf (2012)
12. Amamra, A., Talhi, C., Robert J.-M., Hamiche, M.: Enhancing Smartphone Malware Detection Performance by Applying Machine Learning Hybrid Classifiers. In: Kim, T.-H., Ramos, C., Kim, H.-K., Kiumi, A., Mohammed, S., Ślęzak, D. (eds.) Computer Applications

for Software Engineering, Disaster Recovery, and Business Continuity, CCIS, vol. 340, pp. 131–137. Springer, Heidelberg (2012)
13. Zhao, M., Zhang, T., Ge, F., Yuan, Z.: RobotDroid: A lightweight malware detection framework on smartphones. J. Networks **7**, 715–722 (2012)
14. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: Intelligence and Security Informatics Conference (EISIC), pp. 141–147. Odense, Denmark (2012)
15. Aung, Z., Zaw, W.: Permission-based android malware detection. Int. J. Sci. Technol. Res. **2**(3), 228–234 (2013)
16. Kent, J.T.: Information gain and a general measure of correlation. Biometrika **70**, 163–173 (1983)
17. Amos, B., Turner, H. White, J.: Applying machine learning classifiers to dynamic Android malware detection at scale. In: Wireless Communications and Mobile Computing, pp. 1666–1671 (2013)
18. Alazab, M., Moonsamy, V., Batten, L. M., Tian, R., Lantz, P.: Analysis of malicious and benign Android applications. In: 32nd International Conference on Distributed Computing Systems, pp. 608–616. IEEE, Los Alamitos, California, USA (2012)
19. Abawajy, J., Beliakov, G., Kelarev A., Yearwood, J.: Performance evaluation of multi-tier ensemble classifiers for phishing websites. In: 3rd Applications and Technologies in Information Security, pp. 11–16. School of Information Systems, Deakin University Press, Australia (2012)
20. Lu, Y., Din, S., Zheng, C., Gao, B.: Using multi-feature and classifier ensembles to improve malware detection. J. CCIT **39**, 57–72 (2010)
21. Lantz, P.: An android application sandbox for dynamic analysis. Master's thesis, Department of Electrical and Information Technology, Lund University, Lund, Sweden (2011)
22. Shekhar, S., Dietz, M., Wallach, D.: Adsplit: Separating smartphone advertising from applications. In: 20th USENIX Security Symposium, pp. 553–567. USENIX, Bellevue, USA (2012)
23. Stevens, R., Gibler, C., Crussell, J., Erickson, J., Chen. H.: Investigating user privacy in android adlibraries. In: IEEE Mobile Security Technologies (MoST 2012), 10 pp. California, USA (2012)
24. Grace, M.C., Zhou, W., Jiang, X., Sadeghi A.: Unsafe exposure analysis of mobile in-app advertisements. In: 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 101–112. ACM, Arizona, USA (2012)
25. Moonsamy, V., Batten, L.M., Shore, M.: Can smartphone users turn off tracking service settings?' In: MoMM, 9 pp. ACM Digital Library, Arizona, USA (2013)
26. Rahulamathavan, Y., Moonsamy, V., Batten, L.M., Shunliang, S., Rajarajan, M.: An analysis of tracking settings in Blackberry 10 and Windows Phone 8 Smartphones. In: ACISP 2014, LNCS vol. 8544, pp. 430–437. Springer, Heidelberg (2014)
27. Chong, I.-G., Jun, C.-H.: Performance of some variable selection methods when multicollinearity is present. Chemometr. Intell. Lab. Syst. **78**, 103–112 (2005)
28. Barber, B., Hamilton, H.: Parametric algorithms for mining share frequent itemsets. J. Intell. Inf. Syst. **16**(3), 277–293 (2001)
29. Christen, P.: Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection. Springer, Heidelberg (2012)
30. Mani, I. Zhang, I.: kNN approach to unbalanced data distributions: a case study involving information extraction. In: ICML'03 Workshop on Learning from Imbalanced Data Sets, 7 pp. Washington, DC, USA (2003)