6-3 Risk Analysis System for Android Applications

Takeshi TAKAHASHI and Tao BAN

The number of malware targeting Android terminals is growing. To cope with that, we propose a malware detection technique for Android applications. The proposed technique uses APK's metadata obtained from the online APK markets along with static analysis techniques to improve the accuracy of malware detection. Our prototype implements not only the malware detection technique but also vulnerability detection techniques to protect Android terminals in an integrated manner.

1 Introduction

The smartphone has become an indispensable basic tool in today's modern lifestyle. With a smartphone, people can not only make calls, check emails, and do web browsing. They can also do internet shopping, do their banking online, trade stocks, etc. Android phones are a typical example of smartphones, but these have now become the target of malicious third party attacks.

According to GDATA Software AG, there were already 1,575,644 Android malwares detected by the third quarter of 2015[1][2]. These malwares include strong ransomware, such as Simplocker that encrypts the user's files, and LockerPIN that sets/changes the phone's PIN lock without permission. For example, if LockerPIN changed the PIN without permission, then that changed PIN will not be sent to the user nor even to the attacker. Therefore, even if the user pays the amount demanded by the attacker, the user cannot obtain the changed PIN from the attacker because the attacker does not know it. Security threats to Android are already a pressing reality, and the damages are estimated to be huge. Hence sufficient countermeasures must be taken before an incident occurs[2].

Malware spreads in two main ways. One is where the malware is downloaded from a market and installed, and the second way (similar to a normal PC) is where the user has downloaded and installed the application from a website or email. Both ways of installing malware can occur within the scope of normal usage of an Android phone. An engineer who is always concerned about Android security may probably take action beforehand so such malware is not installed, but there is a broad base of Android users, and users are not necessarily only engineers who know IT in detail. Depending on the family there are a wide range of users, from young children to elderly. Security must be considered by taking all of them into account.

Various technologies are required for assuring Android security, but in this paper a method of determining malware particularly in an Android app (hereinafter referred to as "app") is proposed. In the proposed method, various types of metadata, such as text description of the app which can be obtained from the web, are used in conventional static analysis. When implementing the prototype, in addition to the method for determining malware, a vulnerability determining function was also implemented, and a system that can comprehensively determine risks of the app was constructed. For details, refer to the author's paper [3] and documents [2][4], since this paper is a summary of these.

2 Various approaches to Android app analysis

There are many technologies available for analyzing APK files, and there are mainly two types of analysis: static and dynamic analysis. Static analysis is the white box method wherein the contents in the APK file are analyzed, and dynamic analysis refers to the black box method wherein without analyzing the contents within the APK file, operations are performed on the Android OS and those operations are analyzed. Both types of analysis are effective, but this paper will focus on static analysis. The following is an explanation of each typical approach.

2.1 Use a blacklist

In this approach, the malware is identified based on a

blacklist. Examples of blacklists are: A blacklist of hash values of APK files known as malware, blacklist of risky communication destination IP addresses/URLs, blacklist of certificates of malware creators, etc. With these techniques, operations can be done quickly and widely, but this assumes the blacklist is created based on results that were already evaluated by someone. Hence, the reliability of newly released apps must be evaluated anew.

2.2 Quantifying malware possibility based on statistical processing

In this approach, the technique quantifies the possibility of being malware. One such method is called DroidRisk[5], in which for each permission used by the app, the expected value of risk is calculated, and the total of all those expected values is considered as the risk value of the entire app. Then, malware is determined by whether or not that risk value has reached a certain threshold value. More specifically, it calculates the probability of each requested permission being exploited by the malware, and the severity of the exploit. By multiplying these, it calculates the risk value of each permission. Then, it totals the risk values for all permissions that APK uses, and that value shall be the risk value of that APK.

Even in this method, it achieves a certain level of malware detection precision, but of course it is not perfect. Actually, the expected value of a negative influence caused by a certain permission will differ depending on the type of app, but the context of the app is not taken into consideration. For example, it is not unusual if there is request for permission for accessing the phone book in the calendar app, but it is quite unlikely that a calculator app will make a request for such a permission. Such kinds of contexts are not taken into account, so it obviously limits the precision of malware detection.

2.3 Verifying deviations between text description of app and actual functions

In APK markets, there are metadata such as app category, app text description, etc. Research and development on analysis of whether the text description of app matches the actual operations is also reported. For example, in CHABADA[6], clusters are generated from app text descriptions, and for APKs belonging to each such cluster, APKs having characteristics that are clearly different from other APKs are judged as "apps where the actual functions deviate from the app text description".

In many malware, the actual operations differ from the descriptive text, so by slightly adjusting these methods, they can be put to good use for malware detection. However, if the purpose is to detect malware, then rather than comparing the text descriptions of apps vs. actual operations, the detection ratio will be higher by the direct method of implementing machine learning based on the characteristics information of the app.

2.4 Using machine learning

Quantification of an app's risk and the method of monitoring dubious usage of permission are both highly effective, but if the purpose is narrowed down to binary judgement of whether or not it is malware, the precision of detecting malware will be highest with the machine learning method. Support Vector Machine (SVM)[7] is one machine learning methods. In this technique, for the target data set, the characteristics of each data is mapped, and the data set is divided into two by drawing a demarcation line. This can also be used for malware analysis of apps, and for the characteristics information of the entered APK file, all the APKs files are divided into two by drawing a demarcation line between what is malware and what is not. This differs from the above mentioned risk quantification technique, and it is difficult to implement evaluation that is expressed in multiple stages such as malware identity, but it is very precise for evaluating whether or not the malware is binary.

This method inputs information which becomes the APK's characteristics, so it inputs parameters in which characteristics of malware and non-malware could appear easily, enabling achievement of a higher performance malware identification system. For example, by entering the permission request list being used in each app, a high precision of determining malware can be achieved.

The above explanation of all the methods is based on permissions, but in reality, rather than analyzing permissions, the precision of analysis is usually higher when analyzing API calls. Hence, in each of above mentioned methods, keep in mind the point that there will be improvement in precision of detection of malware by analyzing API calls instead of permissions. For the sake of simplicity, this paper omits study regarding API calls.

2.5 Detecting vulnerability

The vulnerability information related to software can be collected by searching data in the Japan Vulnerability Note (JVN)[8] managed by Japan's IPA, and the National Vulnerability Database (NVD)[9] managed by the NIST in

the USA. Currently, most of the information collected is assumed to be from PCs, and information related to Android is limited.

Therefore, a technology by which people on their own can determine the existence of vulnerabilities is also required. There are various methods, but APK files can be analyzed by reverse engineering, and whether the program contains vulnerable code can be checked. Here, the standards for unsafe coding are required, for example methods such as judging risks of violating various coding guide compliance can be considered. For example, "Android Application Secure Design/Secure Coding Guidebook"[10] released by the Japan Smartphone Security Association (JSSEC). A method can be that when code that violates guidelines determined here is detected, it can be judged as vulnerable. This guidebook already has rules that should be followed, so just by building tools that check the status of compliance with these rules, the vulnerability check will be automatically done to a certain extent.

3 Proposed method

The proposed method uses metadata of the app collected from the web. This metadata includes text description of app, category information etc. At present, only these two types of information are being used. As a specific algorithm, two types of methods are proposed.

First is the DRcategory, which is a method for extending DroidRisk. In DRcategory, in each metadata of the app, DroidRisk is implemented. DroidRisk is a method for determining malware by statistical processing, and it achieves precision by implementing statistical processing for each context of the app. We also built DRcluster, that autogenerates clusters from the text description of app, instead of category clusters, and implements DroidRisk for each cluster. But it is omitted here.

We next propose SVMcluster, a method for extending the SVM machine learning method. SVM is a method where the characteristics of an app are extracted from various parameters, and based on the parameters, the characteristics are classified into two groups, depending on whether malware is there or not. In the conventional SVM method, the characteristics were configured based on only information contained in the APK file, but in the proposed method, characteristically different information such as the metadata on the web are used for characteristics extraction. In SVMcluster, machine learning is implemented on the SVM using the cluster information generated from the text description of the app and the static analysis result.

Both of the proposed methods mentioned above are extensions and improvements of existing methods. Of these, SVMcluster has the best performance anyway, so currently we mainly use SVMcluster in the implementation of the prototype described below. Refer to document [3] for details on these methods.

4 Constructing the prototype

This section describes the app risk evaluation system that was built as a prototype of SVMcluster. In this prototype, all the apps installed in the Android phone are monitored, and the risk is evaluated for each app. The risk is evaluated from both aspects of threat and vulnerability, and in threat evaluation, whether the app is malware or not is evaluated in three levels. Here, three level evaluation means "Red" "Yellow" "Unlit" evaluation in the sense of signals. When there is a probability that the app is malware, "Red" evaluation is given. When there is a possibility of malware, "Yellow" evaluation, and when a risk cannot be particularly detected by the current evaluation engine "Unlit" evaluation is given. In vulnerability evaluation, whether the app is seriously vulnerable or not is evaluated in three levels, similar to the signal format.

The evaluation engines of threat and vulnerability are independent, and various types of analysis methods can be freely incorporated in each evaluation engine. This is done because the current risk analysis method may not be optimal in the future, and a single analysis method cannot evaluate risks perfectly. Figure 1 shows a snapshot of our current "Android App Risk Analysis" app.

The figure to left of Fig. 1 shows the comprehensive risk evaluation of a certain app, and based on the evaluation of threats and vulnerability, comprehensive risk was evaluated. The configuration is such that, when the signal of the evaluation result related to threat or vulnerability is tapped, it displays detailed information which is the basis for the respective evaluation. The central figure in Figure 1 shows the threat evaluation results, and the figure to the right of Fig. 1 shows the vulnerability evaluation results.

First, the results of threat evaluation show that multiple evaluation standards are implemented. Currently, we implemented DroidRisk evaluation (DRcategory), malware evaluation based on SVM (SVMcluster), blacklist URL check, etc. When a threat is detected by the malware evaluation based on SVM or blacklist URL check, the threat

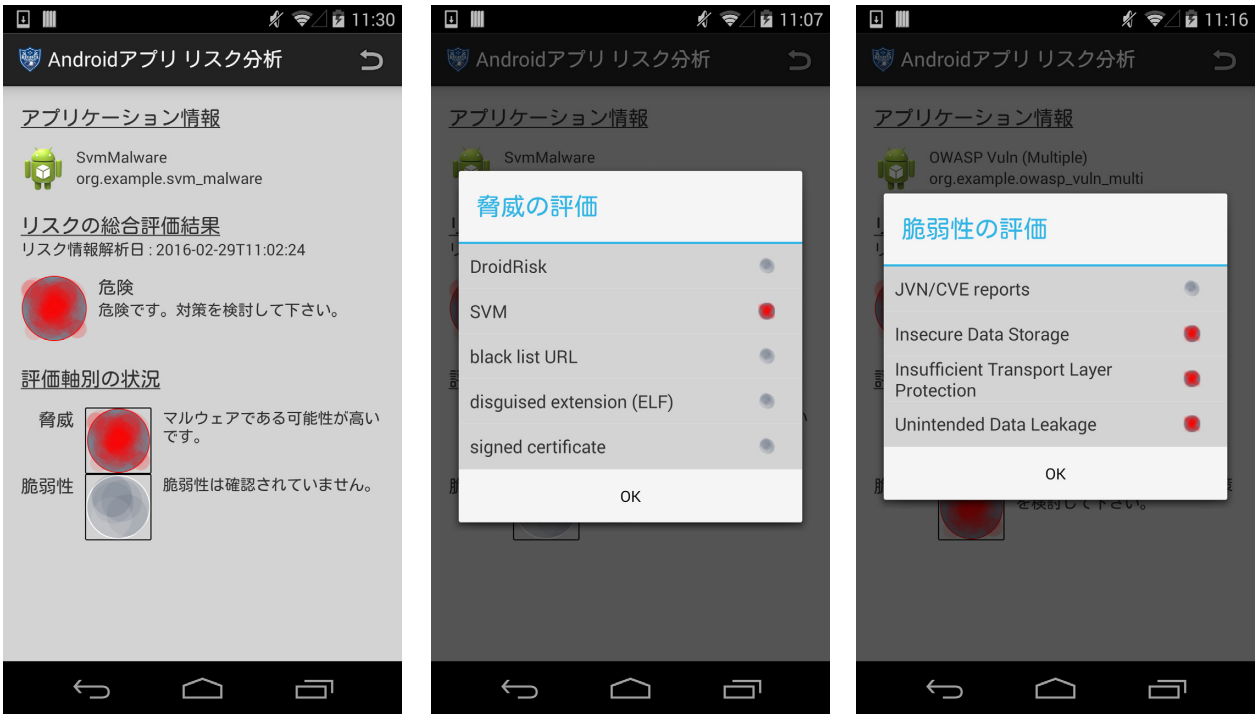

Fig. 1 Snapshot[s] of "Android App Risk Analysis" app

evaluation will be "Red", otherwise when a threat is detected by DroidRisk evaluation, the threat evaluation will be "Yellow". In this example, the threat was detected by both DroidRisk and SVM methods, so the threat evaluation will be "Red". Similarly, the results of vulnerability evaluation show that although there is no vulnerability information registered in JVN, for items that are in violation of the coding guide, the signal evaluation will be red.

For organizations with limited resources, it is difficult from the point of view of human resources to prepare everything with their own resources. However, one can make effective technologies and tools by using techniques that can auto-analyze like in threat analysis algorithms, by reusing open information such as JVN, and by cooperating with other organizations. Actually, we are collaborating with Taiwan's Institute for Information Industry (III) for vulnerability analysis, and they are studying the coding guides of not only Taiwan and Japan, but also those of other major countries, and from those coding guides more than six thousand rules have been extracted. This cooperation with them has also made it possible to cross checking against these rules and implement vulnerability checks.

5 Conclusion

Techniques for analyzing apps seem to have partly reached a level of maturity in the field of research and development, but considering that the environment is always continuously changing, continuous development is desired. Also, when these techniques are actually used in society, there will still be some problems to be resolved. For example, the accuracy of the evaluation results in actual operations must be assured by the operation. Also, when collecting/analyzing data sets, one must consider points such as legal restrictions that cannot be crossed, and the difficulty of deciding criteria for defining what is malware, etc[4].

Acknowledgments

We wish to extend our heartfelt gratitude to Dr. Koji Nakao, Senior Researcher, and Dr. Kazumasa Taira, Research Center Director, for their support in conducting this research.

References

1 G Data, "G DATA RELEASES MOBILE MALWARE REPORT FOR THE THIRD QUARTER OF 2015," 17 12 2015. [Online]. Available: https://www.gdata-software.com/g-data/newsroom/news/article/g-data-releases-mobile-malware-report-for-the-third-quarter-of-2015.

2 Takeshi Takahashi, "Android Security —— step-by-step review starting from application to user issues,"(translated title) 10 3 2016. [Online]. Available: http://www.atmarkit.co.jp/ait/articles/1603/10/news011.html. (in Japanese)

3 T. Takahashi, T. Ban, T. Mimura , K. Nakao, "Fine-Grained Risk Level Quantification Schemes based on APK Metadata," The 2015 International Data

Mining and Cybersecurity Workshop, 2015.
4 Takeshi Takahashi, Tao Ban, "Techniques for identifying malware and application vulnerabilities of Android applications,"(translated title) atmark IT, 28 3 2016. [Online]. Available: http://www.atmarkit.co.jp/ait/articles/1603/28/news002.html. (in Japanese)
5 Y. Wang, J. Zheng, C. Sun , S. Mukkamala, "Quantitative security risk assessment of android permissions and applications," Proceedings of the 27th International Conference on Data and Applications Security and Privacy XXVII, 2013.
6 A. Gorla, I. Tavecchia, F. Gross , A. Zeller, "Checking App Behavior Against App Descriptions," ICSE'14: Proceedings of the 36th International Conference on Software Engineering, 2014.
7 Wikipedia, "Support vector machine," [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Last access: 25 4 2016].
8 JPCERT/CC and IPA, "Japan Vulnerability Notes," [Online]. Available: http://jvn.jp/. [Last access: 1 2014].
9 National Institute of Standards Technology, "National Vulnerability Database Version 2.2," [Online]. Available: http://nvd.nist.gov/. [Last access: 1 2014].
10 Japan Smartphone Security Association, "Android Application Secure Design/Secure Coding Guidebook" 2 2016. [Online]. Available: http://www.jssec.org/dl/android_securecoding_en.pdf.

Takeshi TAKAHASHI, Ph.D.
Senior Researcher, Cybersecurity Laboratory,
Cybersecurity Research Institute
Cybersecurity, Network Security

Tao BAN, Dr. Eng.
Senior Researcher, Cybersecurity Laboratory,
Cybersecurity Research Institute
Cybersecurity, Network Security