Vulnerability Mining Techniques in Android Platform

Wei Zhang, Chengzhi Cao, Wenqing Liu, Yiran Jin
College of Computer
Nanjing University of Posts & Telecommunications
Nanjing 210003, China
zhangw@njupt.edu.cn

***Abstract*— Android is a mainstream smart phone platform. Vulnerability mining work in android platform has become one of the most careful subjects in information security field. This paper combined the vulnerability mining research results of traditional PC platform with the features of android platform to analyze the advantages and disadvantages of traditional vulnerability mining techniques applied in android platform. This paper proposed a four-layer architecture model of vulnerability mining techniques in android platform, and then put forward its possible research directions. Finally, some case studies are given to demonstrate the effectiveness and practical significance of the mining layer, the core layer of the four-layer architecture model.**

Keywords-Android Platform; Vulnerability Mining; Information Security

I. RESEARCH BACKGROUND

With the high-speed development of mobile network, our society has entered into the mobile age. Smart phones, tablets PCs and other mobile devices have been widely join into people's daily life. Following that is the increasingly rampant mobile virus [1], including virus, worm, trojan, malicious mobile code and so on. Like the PC virus, the mobile virus can destroy the normal function of the mobile devices and affect the users. Due to the features of the mobile platform, the mobile virus is more harmful to users than PC virus. Massive private information stored in mobile devices, such as text messages, phone call history, location information, mobile traffic, mobile accounts and so on, may be stole by attackers, which can cause extremely serious consequences to users.

Most mobile virus are using mobile system vulnerabilities to attack and spread, such as Cabir, the world's first mobile virus, take use of the Bluetooth vulnerability of Symbian; Mobile hackers virus, namely Hack.mobile.smsdos, take use of the built-in MMS vulnerability; Gingermaster, a virus for android 2.3, take use of the privilege escalation vulnerability to attack android. So, in order to reduce the harm caused by mobile system vulnerabilities, passive detection and prevention against the virus is not enough. This paper takes the perspective of attackers to mine the existing vulnerabilities in mobile system which can be exploited by the mobile virus.

At present, there are four main mobile system platforms: Google android, Apple ios, Windows phone and Symbian os. According to the 'cloud security' monitoring platform of NetQin statistics [2], In 2013Q1, the number of virus that have been worldwide detected and killed by NetQin is 25140, about 353.05% growth compared to 2012, and the number of infected smart phones is 10.4 million, about 99.23% growth compared to 2012. Among them, 82% of the mobile virus concentrated in android platform. This situation has made android platform the main battlefield of virus and anti-virus. Another report from iiMedia-Research [3] shows that the number of smart phone users in China reached 420 million by the end of 2013Q1, and android's share is 71.0% with an increasing trend. According to the above statistics, this paper mainly introduces the research situation and research directions on vulnerability mining techniques in android platform.

II. ANDROID PLATFORM OVERVIEW

A. Status of the Android Platform Vulnerability

Vulnerability refers to the defects and shortages in the design and implementation of computer system's hardware, software or protocol. Broadly speaking, vulnerability refers to all the factors that threats and breaks the system's reliability, availability, confidentiality, integrity, controllability and non-repudiation. The potential source of vulnerabilities in android platform can be classified into three types [4]: Embedded operating system, runtime environment and application program. Embedded operating system vulnerability refers to the vulnerability causes by android system itself, a typical case is buffer overflow vulnerability; Runtime environment includes java, flash, .net and other support libraries. These support libraries are vulnerable, they may be abused by users and cause runtime environment vulnerability. There are many android applications (apps) in the android market and some android apps may have vulnerabilities. A typical example of application program vulnerability is SSL vulnerability which can cause man-in-the-middle (MITM) attacks.

Based on CVE, Security Focus and other well-known security vulnerability database, this paper counted 252 android platform vulnerabilities published from March 2008 to May 2013 (the total number may be different due to different security vulnerability databases). According to the android vulnerability classification methods mentioned above, we made a classification of these 252 vulnerabilities, as shown in table 1. As you can see, runtime environment and application program are the most vulnerable places, accounted for 49% and 41% respectively. Attackers can get user's sensitive information or execute DOS attack through

these vulnerabilities. The highest risk vulnerabilities are mostly come from embedded operating system, accounted for 10%. Its amount is little, but the harm is great. Attackers can execute arbitrary code and do the privilege escalation operation through these vulnerabilities. The emergence of various android vulnerabilities has largely threatened user's information security. An effective way to assure user's information security is to mine and fix the threatening vulnerabilities before they are being misused by attackers. Therefore, our work is of great practical significance to study the android platform vulnerability mining techniques.

TABLE I. ANDROID PLATFORM SECURITY VULNERABILITIES STATISTICS

Vulnerability Point	Causes	Influence	NO.	Sum
Embedded Operating System	Libpng library's vulnerability	App crashes.	6	25
	GIF library, showLog's function overflow vulnerability, etc.	Arbitrary code execution or DOS.	11	
	Samsung,HTC equipment's vulnerability	Privilege escalation, etc.	5	
	Pusher,ACRA library's integer overflow vulnerability	Buffer overflow.	2	
	Android ADB vulnerability	Allows user to overwrite any files.	1	
Runtime Environment	Android browser's integer overflow, information leaks, etc.	Arbitrary code execution, etc.	7	124
	Adobe Flash Player's vulnerability	Arbitrary code execution or DOS.	117	
Application Program	Cnectd, KKtalk and other apps have unknown vulnerability	Unknown influence.	57	103
	iLunascape, Cookpad and other apps haven't implemented WebView class correctly	Sensitive information access.	6	
	Twicca and other apps haven't limit the use of network access	Sensitive information access.	7	
	Tencent QQPhoto, Kaixin001 and other apps haven't protected data properly	Sensitive information access.	25	
	Mozilla Firefox's vulnerability	Arbitrary code execution or DOS.	4	
	Zoners, Groupon and other app's server name is as same as their domain name	Man-in-the-middle attack.	4	

B. *Android Platform Features*

Android is a Linux-based free and open source operating system, mainly used in mobile devices, such as smart phones and tablet PCs. It adopts software layer architecture. The underlying Linux kernel provides only basic function and the applications are developed independently by the third party companies. Android platform has the following features:

1) Android platform is open source: Analysis on its source code can theoretically mine all existing vulnerabilities. Anyone can use the android source code, so in recent months, a lot of secondary development versions and operator customized versions of android system emerged. On this occasion, the android virus must have a highly targeted purpose and can't be used to all android versions. For this reason, it should be easy to capture the samples of android virus.

2) Android fragmentation: Android system version updates very frequent while the old versions eliminate very slowly. By March 2013, the share of android 2.3 version is 44.2%, still take the overwhelming majority, while the share of the latest android 4.1/4.2 is only 16.5%. This situation leads to some low android versions which have vulnerabilities still have a considerable number of users. It means that a considerable number of users can be very vulnerable to be attack by the old android version's vulnerabilities.

3) Openness of android apps: Android is completely open to the third party app company, any person or team can develops android apps and releases to the app markets for users to download and install. In addition, android app is easy to be reverse analysis, so repacking type virus appears a lot in android platform.

III. TRADITIONAL VULNERABILITY MINING TECHNIQUES

According to different research objects, the traditional vulnerability mining techniques can be classified into two categories: Vulnerability mining techniques that use program as object, namely active vulnerability-mining-techniques; Vulnerability mining techniques that use vulnerability itself as object, namely passive vulnerability-mining-techniques. Figure 1 shows the classification and overview of the traditional vulnerability mining techniques.

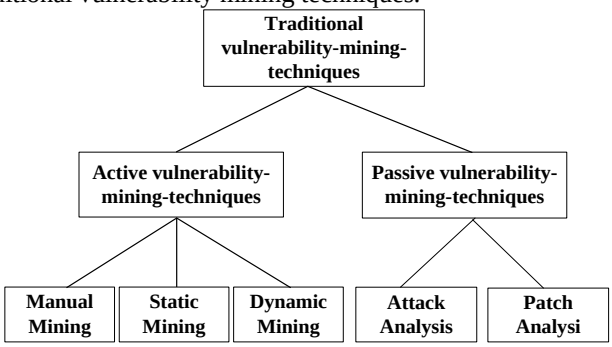

Figure 1. Classification of the traditional vulnerability mining techniques.

A. *Active vulnerability-mining-techniques*

Active vulnerability-mining-techniques uses program as its object, it actively mine the unknown vulnerabilities through the various analysis of program. This paper classified it into three types in the perspective of whether it is needed to run the program [5]: Manual vulnerability-mining-techniques, static vulnerability-mining-techniques and dynamic vulnerability-mining-techniques, as shown in Figure 1.

1) Manual vulnerability-mining-techniques [6]： For a long time, vulnerability mining is mainly by manual mining and depending on researcher's experience. Researcher manually constructed special input parameters for the target program, then observing the outputs and the target state changes in order to obtain vulnerability data. Inputs include valid and invalid inputs and outputs include normal and abnormal outputs. Abnormal outputs are the signs of vulnerability. Manual vulnerability-mining-techniques commonly used in web applications, browsers and other applications that require man-machine interaction.

Manual mining can be done independently by the researchers, so it is relatively simple to implement. But it also has many disadvantages, such as high cost, unstable results, highly depend on the ability of researchers and failing to accumulate knowledge experience, etc. After studying some android apps' vulnerabilities, it can be found that some abnormal outputs, namely crash outputs, are very useful for researchers to lighten the mining work. Therefore, we can build an interactive platform to encourage users to upload their crash information during the daily life, and take a good use of the crash information to reduce the cost of manual vulnerability-mining-techniques.

2) Static vulnerability-mining-techniques [7]: It analysis the source code or decompilation code of the target program to mine some potential vulnerabilities in the case that the program haven't been running. Its main methods include static string search, context search, etc. It is important for static mining to found incorrect function calls and function return values, especially the function calls that haven't done border checks, the functions that may cause buffer overflow, the external call functions, the memory shared functions and the function pointers, etc. To the open source program, we can mine vulnerability by detecting file structure, naming conventions and stack pointer which have violated the security rules. To the non-open source program, firstly we need to do reverse engineering in order to get the decompile code which is similar to the source code, and then analyze the decompile code.

In theory, static vulnerability-mining-techniques can find all existing vulnerabilities. But the expanding feature databases and dictionaries will cause a large set of test results, high false positive rate and other problems. This mining approach focuses only on analyzing the features of code and don't care about program's function. So there will be no vulnerability checks for the program's function and missing some potential vulnerability. The open-source android system allows everyone to study and modify its source code, it also provide convenience for static vulnerability-mining-techniques. At present, domestic and foreign researchers mine the vulnerabilities of functions, libraries and calls mainly through the static analysis of the android source code.

3) Dynamic vulnerability-mining-techniques [8]： It takes use of the program’s runtime information to mine the vulnerability. More specifically, it applies a traversal searching algorithm of the program’s state space, and then monitors whether there is something that violate the specific security attributes during the program’s running time. Dynamic vulnerability-mining-techniques includes fuzz testing, stain spread analysis and defects injection, etc. Fuzz testing is an automated program testing technology, which uses a large number of semi-valid data as program’s inputs, and take program’s abnormal outputs as a symbol to discover program’s security vulnerabilities; Stain spread analysis is a method that mine vulnerabilities in the simulation or actual attack environment; Defects injection technology injects some defective data into the program and then observe whether the program can operates normally. If the program operates abnormally, then the program may have vulnerabilities.

Compared with the other two technologies, fuzz testing technique has a simpler principle and is easier to understand. Moreover, the experience from vulnerability discovery to vulnerability exploit is easy to be reuse. But it also has advantages such as poor generality, very long cycle time for constructing a test case, etc. Currently, fuzz testing technique is used in the vulnerability mining of android Bluetooth protocol, HTTP protocol and android apps.

B. *Passive vulnerability-mining-techniques*

Passive vulnerability-mining-techniques take vulnerability itself as its research objects. We can analyze and recover the vulnerability information base on the security patches and the attack samples that we captured. According to the above definition, passive vulnerability-mining-techniques can be classified into attack analysis and patch analysis.

1) Attack analysis [9]： First, we need to capture some attack samples, and use honeypots to capture attack samples is a common approach. Honeypot has a lot of mature and effective systems, such as Minos, Honey Monkeys and so on. After capturing the attack samples, the key procedure is doing reverse analysis of the attack samples. Reverse analysis is the core technology of Attack Analysis. Reverse analysis is used as a tool to translate the executable binary program into an equivalent program in a high-level language. The development of reverse analysis technology drives the development of attack analysis techniques.

Using attack analysis to mine vulnerability, the advantages are high pertinence, low false alarm rate and high timeliness, etc. But relying too heavily on reverse analysis technology makes attack analysis has significant limitations. Android apps are compiled by Java language. Java language

is easier to do reverse analysis than C&C++ language. Therefore, attack analysis technology has great practical significance in Android-platform.

2) Patch analysis[10,11]: Patch analysis is mainly depends on the public vulnerability patches. We can use the patch information to analyze vulnerability detail such as vulnerability location, vulnerability exploitation, influence, etc. Through the comparison of the patched and unpatched source code or binary code, patch analysis can locate the vulnerability location. Then we analyze the code comparison information to understand the more information of the vulnerability. Finally, we can recover the vulnerability and fix it. There are two easy methods of code comparison, comparison of source code and comparison of decompilation code. The first method is appropriate for the case that there are only few changes between the patched and unpatched vulnerability. It commonly used in the vulnerability caused by string changes and boundary value changes, etc. The second method is appropriate for the case that the targeted program can be reversed and we can discover the vulnerability caused by function parameter changes according to the decompilation code. There are also some complicated code comparison methods such as structured comparison which proposed by Halvar Flake [10] and comparison based on semantics which proposed by Debin Gao [11]. These methods can discover some unstructured changes and displayed in graph.

Patch analysis technique can locate the vulnerability rapidly and accurately, but highly dependent on the patches has restricted the development of this kind of technology. But the situation is much more optimistic in android platform. Android's open source and the frequent patches have made patch analysis a great application prospect in android platform.

IV. Model and Example of Vulnerability Mining

A. Architecture Model

According to the characteristics of android platform and the general flow of vulnerability mining, this paper presents a four-layer architecture model of vulnerability mining techniques in android platform, the four layers are base layer, mining layer, analysis layer and attack layer, as shown in figure 2.

Base layer is mainly used to solve the construction problem of vulnerability mining environment and provide theory support. As the basis of mining work, base layer constructed multiple simulation or reality analysis environment. It also added the special vulnerability environments that need hardware or other special trigger conditions. Based on the research of traditional vulnerability mining techniques, available technologies and tools can be extracted and transplant into android platform after optimization. The supportive technologies of this layer include code analysis, reverse engineering, simulation, etc.

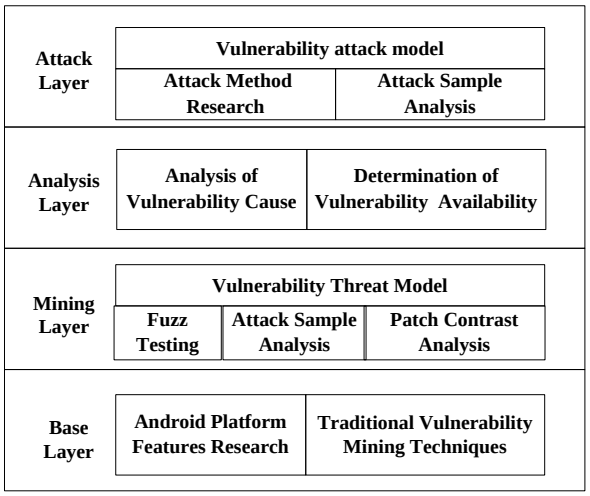

Figure 2. Architecture model

Mining layer is mainly used to solve the problem that what kind of technologies can be used to mine the target object. In particular, this paper presented three technologies to mine the android vulnerabilities, they are fuzz testing, attack sample analysis and patch contrast analysis. Finally, a vulnerability threat model was formed according to the rules and vulnerability information. The threat model accumulated a lot of experience and skills of vulnerability mining work, so it has great significance to the vulnerability mining in android platform. The supportive technologies of this layer include android patches comparison technology, attack samples analysis technology and fuzz testing technology.

Analysis layer is mainly used to confirm if the suspected vulnerability is an exploitable vulnerability. And then debug the vulnerability and monitor its implementation process in order to analyze the causes of it. The supportive technologies of this layer mainly include data tracking, control flow analysis and abnormal monitoring, etc.

Attack layer is mainly used to evaluation its harmfulness, stability and reliability after confirming the vulnerability, and then study its exploit methods. Finally, a vulnerability attack model was formed based on the research of the existing exploit methods and reverse analysis of attack samples. The attack model can provide a lot of exploit methods for android platform. The supportive technologies of this layer mainly include shellcode construction, simulation and permission attack, etc.

B. Cases Study

Based on the above four-layer architecture model of vulnerability mining techniques and the three key technologies of the mining layer, we will study the android platform vulnerability mining techniques with practical examples in this section.

1) Source code comparison based on the android vulnerability patches：

Since the function call relationship is very complex in android, Google or other third-party companies hope to make minimal changes to fix the vulnerability. Through the analysis of the numerous vulnerability patches, we found that

the changes that vulnerability patches caused are not too large and the program's execution process barely changed. Under this kind of patch strategy, these patches may bring new vulnerabilities.

In this paper, we name the vulnerable code location as V-point (Vulnerability Point), and the patched code location as P-point (Patch Point). In android platform, V-point is generally a single vulnerability point, but P-point may be a set of one or multiple patched points. According to the relative position relationship of V-point and P-point, this paper classified the possible conditions into the following three cases:

a) P-point overlap with V-point: That means the patches have directly modified the codes at the vulnerable location. To be specific, the patches have replaced the basic block or unsafe functions at the vulnerable location or directly modified the logical conditions of the vulnerability. However, this kind of patched method does not take the complex real-world situations into account. It lacks an overall consideration. In practical situations, vulnerabilities of the same or similar attributes as the patched one may still exist. What's worse, the patches have exposed the location and attributes of the vulnerability, so researchers can use the exposed information to dig out other unknown vulnerabilities.

b) P-point and V-point are in the same function: That means the vulnerability point code and the patch point code are in the same function. In this case, the patches only consider the vulnerability's context environment and may not consider the influence of the patches on global variables or logical conditions. If there is a path that can bypass the P-point to directly trigger the V-point. That means the patches have caused new vulnerabilities.

c) P-point and V-point are in the different function: That means the vulnerability point code and the patch point code are in the different function. This kind of patches seems like very covert and is hard for reverse analysis. In fact, it is most likely to cause new vulnerabilities in this condition. Function call relationship of android is quite complex, so once the patches have errors in checking each function's parameters, it is likely to cause new vulnerabilities.

Source code comparison technology that based on the android vulnerability patches can quickly locate the position of the patched vulnerability. And with the help of some auxiliary means like dynamic tracking and code analysis, we can understand the principles of the vulnerability and know how to exploit it. This technology makes vulnerability mining work more targeted. Here we take the android adb setuid vulnerability, a kind of privilege escalation vulnerabilities, as an example to illustrate its feasibility.

In android, adb process starts running with root permission in order to complete some initialization work. If attackers can prevent adb process's permission reduced from root to shell, then adb process will continue to be run with root permission, which means we can achieve the root permission without asking the system. Table 3 shows a comparison between the core code of android adb setuid vulnerability before and after patched.

TABLE II. COMPARISON OF ADB SETUID VULNERABILITY

Before patched	After patched

android_src/system/
core/adb/adb.c
/* then switch user and
group to "shell" */
setgid(AID_SHELL);
setuid(AID_SHELL);
	android_src/system/core/adb/adb.c
	/* then switch user and group to "shell" */
	if (setgid(AID_SHELL) != 0) {
	exit(1); }
	if (setuid(AID_SHELL) != 0) {
	exit(1); }

After comparing the code before and after patched, we can obtain the following results: This vulnerability's cause is that the code before patched has not check the return value of setuid( ) function. When the setuid( ) function execution failed, there is no errors alarm, so attackers can go on running the program and eventually get root permission. The return value of functions is always be neglected by programmers. Programmers think setuid( ) function will not fail because the getuid( ) function will never fail, so they ignore to check the return value. After the comparison between the core code before and after patched, we analyzed the exploit program and get more information about the vulnerability: Adb process starts running with root permission, and after that, in a normal case, adb process will call setuid( ) function to reduce its permission from root to shell. But attackers can construct a large number of zombie processes to filled the whole program and make setuid( ) function call failed. Then, attackers start a large number of adb subprocess to force the reboot of the adb process. Again, adb process starts running with root permission, but this time, setuid( ) function call failed, so adb process can still be running with root permission. That is how an insignificant function return value caused a privilege escalation vulnerability.

2) Attack sample analysis based on Java reverse engineering:

For the maximum benefits, attackers often use the latest vulnerabilities to attack the android platform. So we can discover the unpublished 0day vulnerability by attack sample analysis. Reverse engineering technology and code analysis technology are the core technology of attack sample analysis.

Android apps' setup files are APK format. The APK format files are actually ZIP format files, but the suffix name was changed to APK. After unzip the APK files, we can see the Dex files. Dex file is the android dalvik executive program. We take HelloWorld program as an example to look at the file structure of android app, as shown in table 4.

Java source file compiled by javac to generate the class file, then generate the binary file that can run in the Dalvik virtual machine through dx.bat. Therefore, the key to decompile APK file is to reverse class.dex files to Java code files. We can see the class.dex files after unzipping the APK file. Through tool dex2jar, we can transform the classes.dex files into the classes_dex2jar.jar files, just open the tool and run dex2jar.bat classes.dex in the command line interface. Then we can use tool JD-GUI to open the classes_dex2jar.jar files, and you can see the java code of class.dex files in the tool's window. After these steps, we have reversed the android attack sample to Java code for further analysis.

TABLE III. HELLOWORLD PROGRAM'S FILE STRUCTURE

Directory Name	Description
Src	Directory of Java source code, generate class.dex file after compiling.
Gen	Automatically generated directory, including the famous R.java file.
Android 2.3.5	Android sdk package that this app depends on.
Assets	Store the raw files, files in this directory can be packaged in this app.
Bin	The path of Java compiler output.
Res	Store the resource files, including menus, lists, images, etc.
AndroidManifest. xml	Provide the basic information about this app, especially the permissions.
Proguard-project.txt	Android code proguard, use to increase the difficulty of reverse engineering.
Project.properties	Define some basic attributes of the android project.

Researchers of the North Carolina State University have launched a project named Android Malware Genome Project [1]. The project aims to share malware samples and reverse analysis results in android platform. The project have collected 1260 android malware samples in 49 different malware families and identified a series of android malware characteristics, such as use root-level exploits to fully compromise the android security, turn the android phones into a botnet and send out background short messages, etc. Among them, around one third of the malware samples use privilege escalation vulnerabilities to get the top permission of android system, posing the highest level of threats to users' security and privacy. They have identified seven kinds of known privilege escalation vulnerabilities through the analysis of attack samples. Therefore, attack sample analysis that based on Java reverse engineering technology can effectively discover the vulnerabilities in android malware, especially the privilege escalation vulnerability.

3) Fuzz Testing for Intents:

Intent is a unique mechanism in android to exchange data between android processes. The android apps are composed of four components as follows: activity, service, content provider and broadcast receiver. These four components are independent, and Intents help them communicate with each other. Intent is responsible for the description of the app's action, data, etc. Then according to the description, android can find the corresponding component and transfer Intents to it, and finally complete the component call.

A fuzzer is a testing tool that sends unexpected or incorrect inputs to an application in an attempt to cause it to fail. Intent Fuzzer [12] is a fuzzer for Intents. It sends a large number of invalid inputs to the application to cause errors. Errors are crashing bugs, performance issues, etc. And then find the reason why application crashed. The tool can fuzz either a single component or all components. It works well on broadcast receivers but offers less coverage for services. But only single activities can be fuzzed, not all them at once, which limits the tool's range of application and efficiency.

We test Intent Fuzzer in android simulator. We select the desktop clock application and do a single fuzz test to the broadcast of the desktop clock. After the above operations, Intent Fuzzer will construct a large number of invalid Intents to the broadcast receiver of the desktop clock. Soon, we see some errors of the desktop clock, which means that the invalid Intents have crashed the desktop clock program. Then we analysis the crash logs to find the crash reasons, which often represent the vulnerabilities. Therefore, fuzz testing for Intents can quickly and efficiently obtain the error logs and further to mine the potential vulnerabilities.

V. CONCLUSION

The research of android platform vulnerability mining work is just at its beginning, our present study mainly concentrated in mining layer, the core layer of the four-layer architecture model. We are trying to use the traditional vulnerability mining techniques to solve some certain types of vulnerability, like the privilege escalation vulnerabilities. In the future, we need to focus our work on all the four layers of the four-layer model at the same time. Also, the research of android's unique vulnerability and the automatic mining of android platform vulnerability should be a hot topic.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (61272422).

REFERENCES

[1] Zhou, Yajin, and Xuxian Jiang. "Dissecting android malware: Characterization and evolution." Security and Privacy (SP), 2012 IEEE Symposium on. IEEE, 2012.

[2] Beijing NQ Mobile Inc. 2013Q1 global mobile security report: http://cn.nq.com/safety-report/.

[3] iiMedia Research. 2013Q1 China smart phone market monitoring report: http://www. iimedia.cn/36650.html.

[4] Gui Jiaping, Zhou Yongkai. "Research on prevention model of malicious code in smart phone." Computer Technology and Development 20.1 (2010): 163-166.

[5] Ernst, Michael D. "Static and dynamic analysis: Synergy and duality."WODA 2003: ICSE Workshop on Dynamic Analysis. 2003.

[6] Weissman, Clark.System security analysis/certification methodology and results. System Development Corporation, 1973.

[7] Bush, William R., Jonathan D. Pincus, and David J. Sielaff. "A static analyzer for finding dynamic programming errors."Software-Practice and Experience 30.7 (2000): 775-802.

[8] Cadar, Cristian, et al. "EXE: automatically generating inputs of death." ACM Transactions on Information and System Security (TISSEC) 12.2 (2008): 10.

[9] Serjantov, Andrei, and Peter Sewell. "Passive attack analysis for connection-based anonymity systems." Computer Security–ESORICS 2003. Springer Berlin Heidelberg, 2003. 116-131.

[10] Flake, Halvar. "Structural comparison of executable objects." Proc. of the International GI Workshop on Detection of Intrusions and Malware & Vulnerability Assessment, number P-46 in Lecture Notes in Informatics. 2004.

[11] Gao, Debin, Michael K. Reiter, and Dawn Song. "Binhunt: Automatically finding semantic differences in binary programs." Information and Communications Security. Springer Berlin Heidelberg, 2008. 238-255.

[12] iSECpartners. intent-fuzzer: https://www.isecpartners.com/tools/mobile-security/intent-fuzzer.aspx.