Open Access License Notice

Limitations of Current Android Analysis Tools

1 Introduction

There has been an exponential growth in smartphones usage for the past several years. According to Gartner Inc. smartphones dominated the overall mobile phone sales, and the fourth quarter of 2012 saw record smartphone sales which was up 38.3 percent from the same period in 2011. These devices which put the Internet, a phone, GPS tracking, a camera, a microphone and much more in the hand of the users, have gradually become an extension to the human body. Nowadays, smartphones contain more personal information (from contacts and SMS history, to passwords to different accounts and GPS records) than an average PC, which make them a lucrative target for hackers.

With the growth and popularity of smartphones usage, there has been a similar increase in the spread of mobile malware. The Android OS as the most widely used platform for the mobile phones has been the most popular target for writers of mobile malware. In the July-September 2012 quarter alone, Blue Coat Security Labs saw a 600 percent increase in Android malware over the same period last year[1]. One of the main factors contributing to the success of malware attacks by cybercriminals on Android platform more than other platforms is the unregulated app market and diversity of Android-based devices.

Given the rapid growth of Android malware, there is a pressing need to effectively mitigate or defend against them. Static and dynamic security analysis tools have always been the first line of defense and have helped users to find and remediate vulnerabilities. Static analysis tools provide the user with lots of useful information about the application, such as permissions requested and the APIs which were called. Dynamic analysis tools monitor applications in execution and analyze their behavior by looking at the system and network logs. These tools are frequently considered the silver-bullet for detecting vulnerabilities and malware in mobile apps. However, the effectiveness of most of them has not been fully determined and in some cases is very low. The undeserved confidence in these tools may lead to deployment of applications with undetected vulnerabilities.

To determine the accuracy and effectiveness of some of the commonly used static and dynamic analysis tools for Android apps, we tested twenty seven malicious Android apps on five market-leading tools. To ensure a wide range of malicious apps, we first identified eight primary types of threats and then chose two or three malicious apps which were known for that particular attack. We also identified twelve types of anti-analysis techniques used by malicious apps and categorize our chosen apps, accordingly. In this paper we present our evaluation results and discuss the limitations of some of these widely used static and dynamic analysis tools used to identify malicious behavior of mobile apps.

The rest of this paper is organized as follows: Section 2 presents the background information. Section 3 discusses the evaluation environment and the Android malware set used in this study. Section 4 presents the selected tools and Section 5 shows the evaluation result. We summarize our paper in Section 6

2 Background

Static and dynamic analysis are two primary approaches when it gets to software testing. The healthiness of Android apps is also measured by performing static and/or dynamic analysis. In this section we briefly describe these two approaches pertaining to mobile apps.

2.1 Static Analysis

For static analysis, the application's dex file can be analyzed by itself or it can be disassembled and further decompiled into Java using retargeting tools like dex2jar [2], ded [3] and Dare [4]. Standard static program analysis techniques, such as control-flow analysis and data-flow analysis, can then be performed. Automated code analysis can be also used to give a complete picture of all possible paths. Researchers have demonstrated that this approach is very effective in many cases and can be performed on a large scale of applications in a short period of time[3] [5]. However, static analysis heavily relies on the reverse engineering process, which is known to be vulnerable to code obfuscation techniques. In fact, Google suggests obfuscation using Proguard [6] for applications using its licensing service. Recently DexGuard [7] has also been introduced to optimize and obfuscate the Android apps and it even can encrypt entire classes and thoroughly remove Android logging code. Android malware may also generate or decrypt native components or Dalvik bytecode at runtime. Moreover, researchers have demonstrated that bytecode randomization techniques can be used to completely hide the internal logic of a Dalvik bytecode program [8]. While formal Java code analysis tools like Fortify SCA are useful, Enck et al. [3] found that custom tools are required to overcome analysis hurdles created by the Android middleware. For example, component IPC must be tracked through the middleware, the middleware API has many callbacks that indirectly use IPC, and APIs frequently require depend on variable state. Besides, if the malicious behavior is distributed in multiple applications or the malware payload is dynamically loaded, it is difficult for static analysis to detect it since it usually only analyzes the static code of one application at a time. Static analysis also falls short for exploit diagnosis, because a vulnerable runtime execution environment is needed to observe and analyze an exploit attack and pinpoint the vulnerability.

2.2 Dynamic Analysis

Dynamic program analysis is the analysis of computer software that is performed by executing programs built from that software system in a real or virtual environment. To avoid infecting the phone and to allow snapshots, the dynamic analysis of Android applications is usually performed by executing and observing the Android application on an emulator installed on a virtual machine. Contrary to static analysis, dynamic analysis overcomes anti-reverse-engineering and is able to monitor the app behavior on an actual execution path. Dynamic analysis allows users to detect not only dynamically registered broadcast receivers that need not be listed before actual execution, but started services as well. Techniques used in dynamic analysis include system hooking, dynamic taint analysis, and instrumentation. The levels upon which dynamic analysis can be performed include the application framework/java level, native library level, kernel/driver level, and the emulator/QEMU level.

Currently, there are a few dynamic analysis tools available and many of them are based on TaintDroid [9]. Enck et al. proposed TaintDroid and modified Android's Dalvik VM to perform instruction-level taint tracking to identify when applications send privacy sensitive information to network servers. Dynamic analysis is limited by scalability. As discussed by Gilbert et al. [10], generating test inputs is hard. Although researchers have proposed some solutions [10] [11] [12] to create an automated smartphone application analysis environment for dynamic detection engines, it's very hard to cover all execution paths. In the characterization of the majority of exiting Android malwares [13], most malicious behaviors of Android malwares are activated by the systemwide Android events, such as "BOOT COMPLETED" and "SMS RECEIVED". Other malicious behaviors are activated when meeting special conditions, such as location, time, SMS and OS version, or by commands from a control-and-command (C&C) server. Even with manual input and investigation, these malicious behaviors are hard to detect through dynamic analysis. Also, most dynamic analysis is performed on an emulator and can be easily thwarted by detecting whether the host device is an emulator before running.

3 Evaluation Setting

3.1 Evaluation environment

We used virtualization-based analysis approach to set up our evaluation environment – the Android applications are executed and observed on an emulator installed on a virtual machine. VMware Workstation is used to run the Linux-kernel target operating system on the host. For the Linux-kernel OS, we compared 3 options: 1) Santoku from viaForensics[14], 2) Android Reverse Engineering (A. R. E.) virtual machine [15] from the Honeynet Project, 3) Ubuntu LTS (10.04). We chose Santoku since the image is preconfigured with several mobile security tools (Apktool, Androguard, Dex2Jar) and has uninstalled unnecessary packages of Ubuntu.

3.2 Sample Selection

The malware examples used are collected from Android Malware Genome Project [16] [13] and Contagio mobile malware mini dump [17]. To ensure that we select a wide range of malware with different attack payloads, we first identified the threat categories exploited by malicious apps as shown in Table 1 and then selected at least two malicious apps for each threat type. To better understand and analyze the evaluation results we further classified the selected malicious apps into twelve anti-analysis techniques[13], as depicted in Table 2.

Our malware set includes: 1)AnserverBot[18], 2)BeanBot[19], 3)CI4[20], 4)ConnectSMS[21], 5)Copy9[22] 6)DogWars[23], 7)DroidDreamLight[24], 8)DroidKungfu2A[25], 9)DropDialer[26], 10)Exprespam[27], 11)Fakeguard[28], 12)FakeNetflix[29], 13)Geinimi [30], 14)GoldDream[31], 15)Loozfon[32], 16)LuckyCat[33], 17)MobileAttacks[34], 18)NotCompatible[35], 19)Uranico[36], 20)SndApps[26], 21)sumzand[29], 22)TapSnake[37], 23)Tascudap[38], 24)Tetus[39], 25)TigerBot[40], 26)Trojan!Extension.A[41], 27) zSone[42]

Table 1 Malware malicious behaviors

Threat		Malwares
Information Stealing	Phone Identifier	2, 3, 4, 5, 7, 8, 13, 15, 19, 20, 24, 25, 26
	Phone number	2, 3, 8, 10, 11, 13, 14, 19, 20, 21
	User account	7, 12, 20
	Contact List	4, 5, 7, 8, 10, 15, 17, 19, 21, 26
	SMS	5, 11, 13, 14, 17, 25
	GPS Location	5, 14, 17, 22, 25, 26
	Device Information	4, 8, 11, 13, 17, 25
	List of installed apps	5, 7, 24, 25
	Call history	5, 26
	Photos	5, 17, 25
	Files	5, 16, 17
Financial Charges	Phone Call	2, 13, 14, 26
	SMS	2, 4, 6, 9, 11, 13, 14, 17, 24, 26, 27
Microphone/camera eavesdropping		5, 25
Killing other running processes		1, 25
Bots - Remote Control, DDoS/APT attacks, TCP relay/Proxy		1, 2, 3, 5, 8, 13, 14, 16, 17, 18, 23, 25
Download/install/update/remove apps		1, 5, 8, 9, 16, 17
Infect PCs		17
Privilege Escalation		7, 8, 14

Table 2 Anti-analysis techniques

Behavior	Malwares
Heavy Obfuscation	1, 3, 8, 11, 26
Shadow Payloads	1, 3, 8, 18, 25
Encryption	1, 10, 13, 20, 26
Native program (JNI)	1, 8, 14, 25, 26
Partition Payloads	1, 9, 17, 26
Dynamic loading	1, 9, 17, 26
Anti-repacking	1
Implicit data flows*	
Verify the device before run*	
Triggered by specific condition	1, 3, 8, 12, 20
Triggered by C&C commands	1, 2, 8, 13,14, 16, 17, 18
Triggered by SMS/phone call	2, 7, 11, 14, 25, 27

4 Selected tools

The main purpose of this project is to study the effectiveness of current tools for detecting malicious apps. In this section we briefly discuss the tools we selected and their limitations.

4.1 ComDroid

Android supports inter-application communication through the use of intents. Unfortunately, intents may also make one application vulnerable to others. The contents of intents can be sniffed, modified, stolen, or replaced, which can compromise user privacy. Also, a malware can inject forged or otherwise malicious intents, which can lead to user data being breached and application security policies being violated.

ComDroid[43] is a static analysis tool that detects application communication vulnerabilities. It emits warnings for both sending- and receiving-based intent vulnerabilities and gives additional details on when the developer of an application may be using intents in an improper way.

Applications for the Android platform include Dalvik executable files that run on Android's Dalvik Virtual Machine. ComDroid first disassembles these DEX files using the publicly available Dedexer tool. Then, disassembled output from the Dedexer is parsed and potential component and intent vulnerabilities are logged.

Limitations:
It was pointed out by Enck [9] that ComDroid's control flow analysis follows all branches, which can result in false negatives during analysis. While static analysis tools can identify already well-known privacy leaks, the challenge for these tools is automatically determining if the leak was desired.

4.2 TaintDroid

Enck et al. proposed a realtime monitoring service called TaintDroid [9] and modified Android's Dalvik VM to perform instruction-level taint tracking to identify when applications send privacy sensitive information to network servers. TaintDroid uses variable-level tracking within the VM interpreter. To use TaintDroid you must flash a custom-built firmware to your device, similar to a number of popular community-supported Android ROMs or use an emulator.

Limitations:

- TaintDroid does not include Google API by default. A big issue with TaintDroid is that many Android apps with Google map features cannot be installed.

```
E/PackageManager(73): Package com.riteaid.android requires unavailable shared library com.google.android.maps; failing!
W/PackageManager(73): Package couldn't be installed in /data/app/com.riteaid.android-1.apk
```

  To use Google API, you have to use a cracked device and install Google apps through a flash method, which will void the warranty of the device.
- TaintDroid can only be used to detect information stealing (IS) threats.
- TaintDroid only tracks data flows (i.e., explicit flows) and does not track control flows (i.e., implicit flows) to minimize performance overhead. Therefore, a malicious developer can use implicit flows within an application to "scrub" taint markings from variables. However, such actions are likely identifiable using static analysis and will draw attention of developers for attempting to hide their tracks. Gilbert et al. [10] extend TaintDroid to track specific types of implicit flows and discuss approaches for automating application analysis. However, their proposal has no implementation.
- Taint Source limitation: Only tracks information leakage on phone number, ICC-ID, IMEI, contact list, location, microphone, camera, SMS, and does not track IMSI, the list of installed applications, call history, files, or in-app data.
- Taint Sink limitation: only tracks information leakage to network (leave the phone), and does not track information leakage to files, database, Logs or via IPC.
- Message-level taint propagation for IPC leads to false positives.
- Taint source and sink placement is limited to variables in interpreted code, IPC messages, and files.
- TaintDroid-based dynamic analysis tools are more focused on detecting privacy violation and ignored security violations and vulnerabilities.
- The leakage is reported as an alert in the device. There is no easily readable report generated on the host machine. Although we can use the logcat tool to capture the log, it is not easy to find the privacy leakage information in the log.

4.3 DroidBox

Droidbox [44] is an open source project based on TaintDroid. It uses dynamic taint analysis and system hooking techniques at the application framework/Java level to monitor actions. Droidbox is written in the Python programming language. The analysis is currently not automated except for installing and starting packages in an Android emulator. It analyzes the manifest file of the application and starts the launch activity of this app. Ending the analysis is simply done by pressing Ctrl-C or specifying a time argument for the total running time.
The improvement of DroidBox over TaintDroid includes detecting: information leaks via SMS and file; network IO and file IO; cryptography operations performed using Android API; SMS and phone calls; started services and loaded classes through

DexClass Loader; broadcast receivers; enforced or bypassed permissions. Moreover, it collects the log information and generates readable reports in text and graph format. Besides the limitations of TaintDroid, Droidbox also has its own limitations.

Limitations:

- The Android OS version of TaintDroid is already 4.1, and the Android OS version of DroidBox is still 2.3. Some malware's target SDK is higher than 10 and may cause problems running on DroidBox.
- Since Droidbox can only be used on an emulator and is limited to one version of Android, it will fail when the apps check the phone's feature before running, such as the OS or some hardware feature.
- There are some obvious bugs need to be fixed.
  - The DroidBox will automatically install the app using the Monkeyrunner[45] tool only when the app contains a MAIN Activity. Some malware will hide their existence and only contains Service or Broadcast Receivers to perform sneaky actions.
  - The DroidBox will automatically launch the first activity with an Action tag in the manifest file. However, the first activity may not be the correct launcher and will cause problems if other activities depend on the launcher activity. The correct way is to use "android.intent.category.LAUNCHER" category tag.
  - The kernel of the emulator is modified by the Droidbox and it cannot open browser view.

```
I/ActivityManager(71): Process com.android.browser (pid 472) has died.
E/InputDispatcher(71): channel '40853ea0 com.android.browser/com.android.browser.BrowserActivity
(server)' ~ Consumer closed input channel or an error occurred. events=0x8
```

4.4 Anubis

Anubis [46] is a tool for the analysis of the behavior of Windows PE-executables with a specialized focus on malware. Users submit their .apk files (normal or malware) to a website, which in turn generates a report. The report provided by Anubis gives the human analyst insight into various behavioral aspects and properties of the submitted application. To achieve comprehensive results, Anubis employs both static and dynamic analysis approaches by leveraging existing open source projects: DroidBox, TaintDroid, apktool[47] and Androguard [48], and generates a readable report in HTML and XML format.

The static analysis report contains activities, services, required permissions, used features, URLs which are accessed by the application, and information related to the intent-filters declared by these components. The report also displays both external libraries that are necessary to run the app as well as specific hardware features the app requires. Furthermore, the permissions the user has to grant at installation-time are compared with those actually used by the application.

The dynamic analysis is based on DroidBox, TaintDroid, and it fixes some issues and generates a more readable report than DroidBox. Besides the operations reported by DroidBox, all traffic transmitted during the sandbox operation is captured and provided as a pcap file. Dynamically loaded code, both on the Dalvik VM level (DEX-files) and on the binary level, is reported on.

According to our evaluation, the report generated by Anubis is more accurate than DroidBox and TaintDroid. However, it still suffers from a number of limitations and failed to detect many malicious behaviors.

Limitations:

- The static analysis does not report any vulnerability. For each Broadcast Receiver, only one action is reported, which is incomplete.We can find in the manifest files that some Receivers can be started by several actions. Like in GoldDream, the

receiver can be started by booting, receiving SMS or incoming/outgoing phone call. Anubis only reported the booting action, so the analyzer may be misled and did not send SMS and make phone call to the emulator to trigger possible malicious behaviors.
- It still does not include Google API.
- It fails to track implicit flow and cannot detect in-app data leakage or data leakage to database.
- When the payload is written in native code, or dynamically loaded, or partitioned to several parts, the detection rate is very low.

4.5 APIMonitor

Android is upgrading at a fast pace. To avoid endless porting of DroidBox, instead of hooking systems, a new trend of dynamic analysis is to interpose APIs in APK files and insert monitoring code. APIMonitor [49] is such a tool, developed by the same group as Droidbox. It includes an APK instrumentation library, which can parse smali files into tree structure, and implement some instrumentation API for monitoring Android API specified. Smali is an IR (Intermediate Representation) of Dalvik Bytecode and is an assembler for the dex format used by Dalvik. Smali's syntax is loosely based on Jasmin's syntax (Jasmin is an assembler/IR for the Java Virtual Machine). By using smali to do instrumentation, they avoid furthering decompiling dex to Java bytecode and compiling Java back to dex files. APIMonitor first uses code borrowed from Androguard to do reverse engineering to discover the smali code of APK files. Next, it parses smali files of APK to a tree tree-based structure, and then injects monitor code to the smali tree. In the end, it repackages APK to monitor arbitrary APIs. The users can specify APIs to monitor themselves just by putting their set of method signatures in the config file of the API list. By running the repackaged APK, we can get API call logs and understand the APK's behavior. The biggest advantage of APIMonitor is that you can run the repackaged App on any devices, and not rely on the modified kernel.

Limitations:
- From the Logcat report generated by running the repacked APK modified with APIMonitor, we can only detect suspicious behaviors, like reading device's identifier, writing to file, and writing to network. But we can hardly detect if there is information leaking.
- When an app is heavily obfuscated or encrypted, the APIMonitor fails to inject all required monitoring code.
- Some malware includes Native program (JNI). APIMonitor cannot inject code there.
- APIMonitor can only inject monitoring code on static code parts of the app while the payload can be dynamically loaded or partitioned to several apps.
- APIMonitor needs to repack the app, and some malware, like AnserverBot, prevents the infected app from being repackaged again.

It is important to note that all these four dynamic analysis tools share the same limitation we specified in section 2.2.

5 Evaluation Result

As shown in **Table 3**, Anubis performs better than TaintDroid and Droidbox in detecting information leakage on phone number, phone identifier and contact list. It also reports an "otherDB" information leakage and we are unsure what is contained in "otherDB". Because of the bugs we mentioned in the limitation of DroidBox, it performs the worst since it either failed to install the package (Uranico, CI4) or cannot open browser view (DroidKungfu2A, GoldDream). For APImonitor, since we can define which API is to be monitored, we can detect more sensitive information gathering (tagged with * in **Table 3**) behaviors. However, we cannot easily detect whether the sensitive information has been leaked or not. Through APImonitor, we can see that some malware only collects phone numbers in the contact list, and some malware, like Loozfon, will also collect email

addresses in order to send spam to contacts. And some malware, like Copy9, will collect sensitive information and save it to a local database, which cannot be detected by TaintDroid, DroidBox and Anubis.

Table 3 Detection rate of analysis tools on each threat

Threat		**Detection Rate**			
		TaintDroid	Droidbox	Anubis	API Monitor
Information Stealing	Phone Identifier	4/13	3/13	7/13	10/13
	Phone number	0/10	2/10	5/10	9/10
	User account	0/3	0/3	0/3	1/3*
	Contact List	7/10	5/10	4/10	5/10
	SMS	7/6 *	0/6	0/6	1/6
	GPS Location	1/6	0/6	1/6	3/6
	Device Information (OS, APN)	0/6	0/6	0/6	2/6*
	List of installed apps	0/4	0/4	0/4	2/4*
	Call history	0/2	0/2	0/2	1/2*
	Photos	0/3	0/3	0/3	0/3
	Files	0/3	0/3	0/3	0/3
Financial Charges	Phone Call	0/4	0/4	1/4	1/4
	SMS	0/11	1/11	4/11	2/11
Microphone and camera eavesdropping		0/2	0/2	0/2	0/2*
Killing other running processes		0/2	0/2	0/2	0/2
Bots - Remote Control, DDoS/APT attacks, TCP relay/Proxy		See discussion below			
Install/remove apps, update APK, download files		0/6	0/6	0/6	2/6
Infect PCs		0/1	0/1	0/1	0/1
Privilege Escalation		0/3	2/3	0/3	0/3

The testing version of TaintDroid is the most recent version for Android 4.1 and we use the recommended settings on the emulator. One thing we noticed in our evaluation is that TaintDroid failed to detect any phone number information leakage and reported false positive SMS leakage on several malwares. For example, for DroidKungFu2A, we used the same sample as used in the SANS Institute [25] and they did not report any SMS leakage. However, TaintDroid reported SMS leakage on it and several other malwares (tagged with * in **Table 3**).

On the other hand, DroidBox and Anubis did not report any SMS leakage. According to the malware report of GoldDream, when a SMS message is received or there is an incoming/outgoing phone call on an infected phone, GoldDream will collect the information of this SMS message or the phone call and write it into local files for later use. When we tested GoldDream using DroidBox, it did detect the information leakage to the file zjsms.txt and zjphonecall.txt. However, they are marked as TAINT_CONTACTS. DroidBox also reported a bypassed permission on "READ_CONTACTS" on GoldDream and DroidDreamLight, which is not included in their required permissions and can indicate privilege escalation. DroidDreamLight is also started every time a phone call is initiated or received. Anubis failed to detect these malicious behaviors since the analyzer did not send SMS and make phone call to the emulator.

Whether these tools can detect any GPS location leakage is limited by the emulator. Even though we did set GPS location on the emulator, TapSnake still cannot get the location information since "requested provider network doesn't exist". However,

we successfully detected location information leakage on GoldDream using TaintDroid, Anubis and APImonitor. DroidBox failed to detect it since it cannot open browser view to start the app correctly.

Almost half of the testing malwares are botnet malwares. Dynamic analysis tools perform poorly for botnet malwares since these bots only perform malicious behaviors once receiving instruction from command-and-control (C&C) servers. Unless the tool monitors the malicious app for a long time and the C&C server is active, it may fail to detect all malicious behaviors the malware can do.

Many recent malwares use anti-analysis techniques and we can see in **Table 4**, these techniques have various impact on the detection rate of these analysis tools based on our evaluation.

APIMonitor is sensitive to obfuscation and encryption. Droidbox is sensitive to shadow payloads due to its bugs. When the payload is dynamic downloading, loading and partitioned, all these analysis tools performed poorly.

Some malware hide their malicious behaviors in native code. From the static analysis report of Trojan!Extension.A, we can see that it requests lots of permissions and only used several of them. We also see native library load behavior in dynamic report. So we can suspect that many operations are done in native code and are hard to detect with these analysis tools.

For some malware, malicious behaviors are triggered by specific conditions. For example, AnserverBot communicates with C&C server every two hours. SndApps sleeps for three hours after booting, then it wakes up and steals some information. The Trojan CI4 sends private information to the server addresses specified in Twitter accounts that have been established by the attacker. We can also develop malwares which will be triggered when near specific locations. Dynamic analysis tools may not detect the malicious behaviors since they may not fulfill these conditions.

MobileAttacks is a malware discovered on 1/22/2013 on Google Play and is designed to infect PCs. This malware can download three files from C&C server at the instruction of the master and infect PCs once the phone is connected to them. Right now none of the dynamic analysis tools can detect this malware and Anubis does not generate any dynamic analysis report for it.

Table 4 Impact of Anti-analysis techniques on analysis tools

Anti-analysis Techniques	**Impact on tools**				
	Dynamic Analysis				Static Analysis
	TaintDroid	Droidbox	Anubis	API Monitor	ComDroid
Heavy Obfuscation	Low	Low	Low	High	Low
Shadow Payloads	Low	High	Medium	Low	Low
Encryption	Low	Low	Low	High	Low
Native program (JNI)	High	High	High	High	High
Partition Payloads	High	High	High	High	High
Dynamic loading	Medium	Medium	Medium	High	High
Anti-repacking	Low	Low	Low	High	High
Implicit data flows	High	High	High	Medium	No
Verify the device before run	High	High	High	No	No
Triggered by specific condition	High	High	High	High	No
Triggered by C&C commands	High	High	High	High	No
Triggered by SMS/phone call	High	High	High	High	No

6 Conclusion

There is no doubt that mobile malware is on the rise, especially on Android, and increasingly the users and developers rely on automated tools for malware detection. A number of static and dynamic analysis tools have been developed aiming at preventing and detecting malicious behavior. However, one of the main challenges is to determine the reliability and effectiveness of these tools. In this paper we presented the results of evaluating twenty seven existing malware designed to exploit eleven types of vulnerabilities and demonstrated that none of the existing tools is able to detect all the malware types. We also discussed the impact of anti-analysis techniques adopted by malicious apps on the tools.

References

[1] Blue coat systems 2013 mobile malware report. Blue Coat. [Online]. Available: http://www.bluecoat.com/sites/default/files/-documents/files/BC_2013_Mobile_Malware_Report-v1d.pdf

[2] Dex2jar. [Online]. Available: http://code.google.com/p/dex2jar/

[3] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX security symposium*, no. August, 2011.

[4] D. Octeau, S. Jha, and P. McDaniel, "Retargeting android applications to java bytecode," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 6. [Online]. Available: http://siis.cse.psu.edu/-dare/index.html

[5] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: Detecting privacy leaks in android applications," Tech. rep., UC Davis, Tech. Rep., 2011.

[6] Proguard. [Online]. Available: http://proguard.sourceforge.net

[7] Dexguard. [Online]. Available: http://www.saikoa.com/dexguard

[8] Dynamic, metamorphic (and opensource) virtual machines. [Online]. Available: http://archive.hack.lu/2010/-Desnos_Dynamic_Metamorphic_Virtual_Machines-slides.pdf

[9] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, Oct. 2010, pp. 1–6. [Online]. Available: http://appanalysis.org/-tdroid10.pdf

[10] P. Gilbert, B. G. Chun, L. P. Cox, and J. Jung, "Vision: automated security validation of mobile apps at app markets," in *Proceedings of the second international workshop on Mobile cloud computing and services*, ser. MCS '11. New York, NY, USA: ACM, 2011, pp. 21–26. [Online]. Available: http://dx.doi.org/10.1145/1999732.1999740

[11] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications."

[12] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications," in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2012, pp. 93–104.

[13] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 95–109.

[14] Santoku. ViaForensics. [Online]. Available: https://santoku-linux.com/

[15] Android reverse engineering (a.r.e.) virual machine. Honeynet Project Redmine. [Online]. Available: https://-redmine.honeynet.org/projects/are/

[16] Android malware genome project. North Carolina State University. [Online]. Available: http://www.malgenomeproject.org/

[17] Contagio mobile malware mini dump. [Online]. Available: http://contagiominidump.blogspot.com/

[18] An analysis of the anserverbot trojan. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/pubs/AnserverBot_Analysis.pdf

[19] Beanbot sms trojan. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/BeanBot/

[20] Ci4 - android.twikabot. Symantec. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2012-062614-5813-99

[21] Trojan.rus.sms."systemsecurity" - toll fraud / connectsms. [Online]. Available: http://contagiominidump.blogspot.com/2012/12/-trojanrussmssystemsecurity-toll-fraud.html
[22] Copy9: Phone spy app. [Online]. Available: www.copy9.com
[23] Android.dogowar. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2011-081510-4323-99
[24] T. Wyatt. (2011, May) Update: Security alert: Droiddreamlight, new malware from the developers of droiddream. Lookout Mobile Security. [Online]. Available: http://blog.mylookout.com/blog/2011/05/30/security-alert-droiddreamlight-new-malware-from-the-developers-of-droiddream/
[25] Dissecting andro malware - droid kungfu 2-a. SANS Institute. [Online]. Available: http://www.sans.org/reading_room/-whitepapers/malicious/dissecting-andro-malware_33754
[26] Android.dropdialer identified on google play. [Online]. Available: http://www.symantec.com/connect/blogs/androiddropdialer-identified-google-play
[27] Malware authors create android.exprespam after prosecutors drop case. [Online]. Available: http://www.symantec.com/connect/-blogs/malware-authors-create-androidexprespam-after-prosecutors-drop-case
[28] Android/fakeguard.a!tr.spy. [Online]. Available: http://www.fortiguard.com/av/VID4366818
[29] Security alert: Fake netflix app aids phishing. [Online]. Available: https://blog.lookout.com/blog/2011/10/13/security-alert-fake-netflix-app-aids-phishing/
[30] T. Wyatt. (2010, December) Security alert: Geinimi, sophisticated new android trojan found in wild. Lookout. [Online]. Available: http://blog.mylookout.com/blog/2010/12/29/geinimi_trojan/
[31] Golddream. [Online]. Available: http://www.csc.ncsu.edu/faculty/jiang/GoldDream/
[32] Loozfon. [Online]. Available: http://www.symantec.com/connect/blogs/loozfon-malware-targets-female-android-users
[33] Adding android and mac os x malware to the apt toolbox. Trend Micro. [Online]. Available: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_adding-android-and-mac-osx-malware-to-the-apt-toolbox.pdf
[34] Mobile attacks - android with windows malware downloads. [Online]. Available: http://www.securelist.com/en/blog/805/-Mobile_attacks
[35] (2012, May) Update: Security alert: Hacked websites serve suspicious android apps (notcompatible). Lookout. [Online]. Available: http://blog.mylookout.com/blog/2012/05/02/security-alert-hacked-websites-serve-suspicious-android-apps-noncompatible/
[36] Android uranico - infostealer. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2012-052803-3835-99
[37] Androidos.tapsnake: Watching your every move. [Online]. Available: http://www.symantec.com/connect/blogs/-androidostapsnake-watching-your-every-move
[38] Android.tascudap. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2012-121312-4547-99
[39] Android tetus - infostealer. [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2013-012409-4705-99
[40] Security alert: New android malware — tigerbot — identified in alternative markets. [Online]. Available: http://research.nq.com/-?p=402
[41] Trojan!extension.a ╠ complex malware escapes av detection. [Online]. Available: http://blog.trustgo.com/trojanextension-a-complex-malware-escapes-av-detection/
[42] Security alert: Zsone trojan found in android market. [Online]. Available: https://blog.lookout.com/blog/2011/05/11/security-alert-zsone-trojan-found-in-android-market/
[43] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 239–252. [Online]. Available: http://www.comdroid.org/
[44] Droidbox. Honeynet Project. [Online]. Available: http://code.google.com/p/droidbox/
[45] monkeyrunner. [Online]. Available: http://developer.android.com/tools/help/monkeyrunner_concepts.html
[46] Andrubis: A tool for analyzing unknown android applications. [Online]. Available: http://blog.iseclab.org/2012/06/04/andrubis-a-tool-for-analyzing-unknown-android-applications-2/ http://anubis.iseclab.org/
[47] Apktool. [Online]. Available: http://code.google.com/p/android-apktool/
[48] Androguard. [Online]. Available: http://code.google.com/p/androguard/
[49] Apimonitor. [Online]. Available: http://code.google.com/p/droidbox/wiki/APIMonitor