EVALUATION OF SECURITY IN ANDROID ANTI-MALWARE APPLICATIONS

R. Tamilarasi[#1], B. Dharani Manoharan[*2]

[#]*PG Scholar, Department of Computer Science and Engineering, Christian College of Engineering and Technology, Dindigul, Tamilnadu - 624619, India.*
tamilarasirobert@gmail.com
**Assistant Professor, Department of Computer science, Christian College of Engineering and Technology, Dindigul, Tamilnadu - 624619, India.*
dharani18@gmail.com

***Abstract*— Mobile devices such as smart phones have gained great popularity in response to vast repositories of applications. Most of these applications are created by unknown developers who may not operate in the user's best interests, leading to malware. Earlier work used DroidChameleon, the DroidChameleon is a systematic framework with various transformation techniques which is used to evaluate the anti-malware products. Those transformations are classified as trivial transformations, DSA-Detected by Static Analysis and NSA-Non-Detected by Static Analysis. However comprehensive evaluation using a much larger number of malware samples and anti-malware tools are not performed. To deal with this, the proposed system extends DroidChameleon with various detection techniques such as Taint tracing, sensitive API monitoring and Kernel-level monitoring for detecting malware in android. The proposed system is experimentally evaluated the anti-malware applications.**

***Keywords*— Android,anti-malware tools, transformation attacks.**

I. INTRODUCTION

Mobile phones are very popular among people today. The most popular operating systems used in mobile phones are Android, iPhone, and Windows mobile. This popularity of using much apps especially in android mobile phone attracts the malwares to attack the different kind of apps[7]. A malware instance is a program that has malicious intent. Examples of such programs include viruses, Trojans, and worms. A malware detector is a system that attempts to identify malware. A virus scanner uses signatures and other heuristics to identify malware, and thus is an example of a Malware detector [4].

In order to protect the mobile phones from various attacks and malwares, many ant-malware products are introduced [7]. But all anti-malware products are not fully secured and high resistance against the Malware samples.

Evaluating anti-malware products of android based mobile phones is a non-trivial issue, especially with the challenge that there are a wide variety of smart phone operating system available now-a-days[1][2]. The Android is a Linux based operating system that runs java based applications. The android anti-malwares are evaluated through various kinds of transformation techniques.

The DroidChameleon techniques are used to appraise the android anti-malware products [1][2]. The term transformation denotes semantics preserving changes to a program. In the system different types of transformations are applied to the malware samples, Each malware samples should undergoes to the transformations as a step by step process, those results are collected as a next generation solution [1][2]. In addition to this some of new techniques are introduced to improve the performance of the evaluation technique by adding much larger malware transofrmations. The tactical detection technique includes Taint tracing, sensitive API monitoring and Kernel-level monitoring.

The prototype of droidchameleon is used to evaluate fifteen popular anti-malware products of android with various transformation attacks and malware samples.

II. BUILDING AN ANDROID APPLICATION

An Android application is mainly written in Java source code. The build process of an Android application is to compile and package a Java source code project into an .apk file that can run on a Smartphone device or emulator. We now summarize the key steps of the build process as follows.[3]

1. Preparation: An Android project contains Java source code (and possibly some other native code), as well as metadata such as resources and programming interfaces. The build process first converts the metadata information into Java code or interfaces.

2. Compilation: All Java source code files as well as the converted metadata are compiled together into .class files, which contain Java bytecode.

3. Bytecode conversion: All Android applications run on the Dalvik Virtual Machine (DVM), which is a runtime environment similar to the Java Virtual Machine (JVM) but is designed for mobile devices that generally have limited hardware resources. The build process converts all .class files into .dex files, which contain the Dalvik Executable bytecode.

4. Building: All resource files, including both non-compiled and compiled files, as well as the .dex files are then packaged (i.e., zipped) into a single .apk file.

5. Signing: The .apk file needs to be digitally signed before it can be published in well-known sites (e.g., Google Market). It is typical that the .apk file is signed with the application

developer's private key, rather than by a centrally trusted authority

6. Alignment: To optimize the performance of the Android program (e.g., reducing memory usage), the .apk file can be aligned along the byte boundaries with the zip align tool. Note that some integrated development environment (IDE), such as Eclipse with the ADT plug-in, will automatically zip align the .apk file after signing the file with the developer's private key[3].

III. FRAMEWORK DESIGN OVERVIEW

The Droid Chameleon, a systematic framework is used to evaluate the anti-malware tools by introducing various transformation attacks and techniques and also increasing the number of Malware samples. Each and every Malware samples should undergo various transformation attacks step by step while preserving their malicious behavior.

The transformations are classified as trivial transformations and DSA, NSA.

TABLE-I
TRANSFORMATIONS

Code	Technique
P	Repack
A	Disassemble &Assemble
RP	Remote Package
EE	Encrypt native exploit or Payload
RI	Remote Identifiers
ED	Encode strings and array data
CR	Recorder Code
CI	Call Indirection
JN	Insert Junk Code
CB	Call Blocker
MF	Message Filter
SB	Safe Browsing
SP	Sensitive API Monitoring
TT	Taint Tracing
KM	Kernel Level Monitoring
RF	Rename Files

A. *Trivial Transformation:*

Trivial transformations are defined as those which does not affect any code level changes.

1) Changing Package Name: The package name is unique to each and every application. Those names are defined in package's Android Manifest. In the given malicious application we change the package name to another name.

2) Disassembling and Reassembling: The compiled Dalvik bytecode in classes.dex of the application package may be disassembled and then reassembled back again. The various items (classes, methods, strings, and so on) in a dex file may be arranged or represented in more than one way and thus a compiled program may be represented in different forms. Signatures that match the whole classes.dex are beaten by this transformation.

3) Repacking: Once repacked, applications are signed with custom keys (the original developer keys are not available). Detection signatures that match the developer keys or a checksum of the entire application package are rendered ineffective by this transformation.

B. Transformation Attacks Detectable by Static Analysis (DSA):

All kinds of static analysis cannot be break by the application of DSA transformations. For examples data flow are possible.

1) Taint Tracing: Taint tracing defines that it trace and tracks privacy-sensitive information leakage. This is implemented by slightly modified version of Taint Droid, an open-source, high-performance taint-tracing system for Android.

2) Sensitive API monitoring: It monitors a few system APIs for detecting possibly malicious functionality. The SMS API is one of the most exploited API in Android. Malicious apps use it to send text messages to premium rate numbers without user's awareness.

3) Kernel-level monitoring: It provides kernel-level tracking to identify known root-exploits.

4) Identifier Renaming: The classes and methods and field identifiers in the byte code can be removed.

5) Data Encoding: The dex files contain all types of strings and the array data that are used in the code. These strings and arrays are used to develop signatures against malware.

6) Call Indirections: The call indirection is the simple way to manipulate call graph of the application to defeat automatic matching. This transformation may be seen as trivial function outlining.

*7) Code Reordering***:** This transformation records the instructions in the method of the program.

8) Junk Code Insertion: The junk code insertion has the code sequences which are executed but do not affect the rest of the program. Detections based on the analyzing instruction (or opcode) sequences are defeated by the junk code insertion. Junk code constitutes simple nop sequences or the most sophisticated sequence and branch that actually have no effects on the semantics.

9) Encrypting Payloads and Native Exploits: In Android, native codes are usually made available as library accessed via JNI. However, some malware such as Droid Dream also pack the native code exploits meant to run from a single command line in non-standard location in the application packages. All those files are stored encrypted in the application package and they can be decrypted at the runtime. The Malware Droid Dream also carries payload application that can install once the system compromised. These payloads are also be stored encrypted.

9) Other Simple Transformations: There are some other transformations as well, specific to Android. Debug

informations, such as source file names and local and parameter variable name, and source line number may be stripped off.

*10) Composite Transformations***:** Any of the above transformations are combined with one another to generate stronger obfuscations. While compositions were not commutative, anti-malware detection results should be agnostic to the order of applications of transformations in all cases discussed above.

TABLE-II
LIST OF ANTI-MALWARE PRODUCTS

S.No	Anti-Malware
1)	AVG
2)	Symantec
3)	Lookout
4)	ESET
5)	Dr.Web
6)	Kaspersky
7)	Trend M
8)	ESTSoft
9)	Zoner
10)	Security Manager
11)	Webroot
12)	Avira
13)	Mobile Security

C. Transformation Attacks Non-Detectable by Static Analysis (NSA)

This transformation breaks all kinds of static analysis. The transformation attacks non-detectable by static analysis are as follow as.

1) Reflection: Java reflection API allows the program to invoke a method by using the name of the methods. We may convert any method call into a call to that method via reflection.

2) Byte code Encryption: Code encryptions tries to make the code unavailable for static analysis. The relevant piece of the application code is stored in an encrypted form and is decrypted at runtime via a decryption routine.

Bytecode encryption is accomplished by moving most of the application in a separate dex file (packed as a jar) and storing it in the application package in an encrypted form. When one of the application components (such as an activity or a service) is created, it first calls a decryption routine that decrypts the dex file and loads it via a user defined class loader. In Android, the DexClass Loader provides the functionality to load arbitrary dex files. Following this operation, calls can be made into the code in the newly loaded dex file. Alternatively, one could define a custom class loader that loads classes from a custom file format, possibly containing encrypted classes. We note that classes which have been defined as components need to be available in classes.dex (one that is loaded by default) so that they are available to the Android middleware in the default class loader. These classes then act as wrappers for component classes that have been moved to other dex files.

The Table I consists of list of transformation attacks and the Table II consists of list of anti Malware products to be evaluated [5].

IV. Related Work

A. *Evaluating Anti-malware Tools by ADAM:*

ADAM, an automated system for evaluating the detection of Android Malware. ADAM is an automated, generic and extensible technique that evaluate the effectiveness of the android anti malware through different transformation techniques. ADAM can be extensible to support new implementation of Malware transformations and detection techniques.

Obfuscated Malware Detection:

Obfuscation resilient detection is based on the semantics rather than syntac. The works of Christodorescu et al [6] present one such technique. Christodorescu et el.[6] and Fredrikson et al. attempt to generate semantics based signatures by mining malicious behavior representations such as data dependence graphs and information flow between system calls.

V. **Conclusion**

The Droid Chameleon, Systematic framework is used to evaluate 15 anti-malwares with various transformation attacks which include taint tracing, sensitive API Monitoring and kernel level Monitoring. Different Malware samples are used to evaluate all anti-malware products and many succumb to even trivial transformations not involving code-level changes.

VI. REFERENCES

[1] V. Rastogi, Y. Chen, and X. Jiang, "Catch me if you can: Evaluating Android anti-malware against transformation attacks", January 2014.

[2] V. Rastogi, Y. Chen, and X. Jiang, "Droid Chameleon: Evaluating Android anti-malware against transformation attacks", in *Proc. ACM ASIACCS*, May 2013, pp. 329–334.

[3] M. Zheng and J. Lui, "ADAM: An automatic and extensible platform to stress test Android anti-virus systems", in *Proc. DIMVA*, Jul. 2012, pp. 1–20.

[4] M. Christodorescu, S. Jha, D. Song, and R. Bryant, "Semantics-aware malware detection," in *Proc. IEEE Symp. Security Privacy*, May 2005, pp. 32–46.

[5]AV Comparitive Test, Anti-Virus Comparitive Performance Test (AV Products)-May 2014,www.av-comparitives.org.

[6] M. Christodorescu, S. Jha, "Testing malware detectors", in *Proc. ACM SIGSOFT Int. Symp. Softw.Test. Anal.*, 2004, pp. 34–44.

[7] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution", in *Proc. IEEE Symp. Security Privacy*, May 2012, pp. 95–109.

[8] M. Christodorescu, and C. Kruegel, "Mining specifications of malicious behavior", in *Proc. 6th Joint Meeting Eur.Softw. Eng. Conf., ACM SIGSOFT Symp. Found. Softw. Eng.*, 2007,pp. 5–14.

[9] C. Kolbitsch, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. 18th Conf. USENIX Security Symp.*, 2009, pp. 351–366.

[10].*DroidKungFu* [Online].Available: http://www.csc.ncsu.edu/faculty/jiang/

DroidKungFu.html
[11]R. Whitwam.Circumventing Google's Bouncer, Android's Anti-Malware System [Online].

[12]M. Zheng, and J. Lui, "ADAM: An automatic and extensible platform to stress test Android anti-virus systems," in Proc. DIMVA, Jul. 2012, pp. 1–20.

[13]C. Collberg, C. Thomborson, D. Low, "A taxonomy of obfuscating transformations," Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997.

[14]S. B. Needleman, C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins." J. Molecular Biol., vol. 48, no. 3, pp. 443–453, Mar. 1970.

[15]T. F. Smith, M. S. Waterman, "Identification of common molecular subsequences," J. Molecular Biol., vol. 147, no. 1, pp. 195–197, 1981.

[16]A. Agrawal and X. Huang, "Pairwise statistical significance of local sequence alignment using multiple parameter sets and empirical justi- fication of parameter set change penalty", BMC Bioinformat., vol. 10,

[17]A. Agrawal and X. Huang, "Pairwise statistical significance of local sequence alignment using sequence-specific and position-specific sub- stitution matrices", IEEE/ACM Trans. Comput. Biol. Bioinformat., vol. 8, no. 1, pp. 194–205, Jan./Feb. 2011.

