International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 4 678 - 682

Evaluation of Android anti-malware resistance against transformation attacks

Omkar Yeshvekar, Snehal Zende, Deepti Walvekar, Namrata Wabale, Akash Korde,
and Mrs. Nilam S. Patil
Department of Computer Engineering, D Y Patil College of Engineering, Akurdi
SavitribaiPhule Pune University, Pune, India
Email: omkaryeshvekar@gmail.com, Email: snehalzende1@gmail.com
Email: walvekardeepti@gmail.com, Email: namratawabale@gmail.com
Email: akashkorde55@gmail.com, Email: neelam v patil@rediffmail.com

***Abstract*—**Android being most popular and user-friendly is targeted by most of the malware authors. The malware authors use various transformation techniques to create different variants of malwares. Different transformation techniques such as obfuscation, repackaging, renaming are used mostly. Many anti-malwares are developed to secure the Android devices. Android does not offer file access permissions to all the applications installed. Thus anti-malwares may not provide complete security to the Android devices. In this paper, many such different techniques are presented that can be used to evaluate different anti-malwares.

Keywords *- Android, Malware, Anti-malware, Transformation*

*****

1. Introduction

Android being open-source offers many applications in An-droid official market. Many third-party applications are also available on Google play store. These applications are very vulnerable to the threats. The overall risk situation for the An-droid users depends on two main factors, firstly threat level by malware and secondly protection level by anti-malware [12]. To maintain the privacy of user data and security of applications, Android does not provide file system monitoring to the applications installed. Thus to access the file system for malware detection, anti-malware has to rely on package database and package file of installed applications. The package database consists of all the contents and locations of the directories of installed applications. The package file consists of all the log files, backup files, etc of installed applications. Thus anti-malware can access the package database to scan the files of applications installed on the device.

Malware authors use different techniques to transform the malwares into variants of malwares. The transformation techniques makes some code level and package level changes in the malwares. These changes may be minor but creates variant of malware. The changes done in the malware also change the signature of the malware thus making the variant unique. Therefore anti-malware is not able to recognize the variant of malware.

In this paper, variants of malwares are created through transformation techniques and applied to the anti-malware tools. Evaluation technique includes study, design and implementation of framework to generate variants of malwares to check the effect of anti-malware tools.

The most popular Android anti-malware products such as AVG Antivirus Free, Lookout Mobile Security, ESET Mobile Security, Dr. Web anti-virus Light, Trend Micros Mobile Security, Andro Helm, EST Soft and Zoner are selected for evaluation. The malware set consists of various kinds of malware. It includes Benign, Geinimi and BaesBridge families of malwares.

2. Related Work

2.1 ADAM

ADAM is an automated system for evaluating the detection of Android malware. ADAM is able to automatically transform a malware sample to different variants of malware using repackaging and obfuscation techniques in order to evaluate the efficiency of different anti-malware systems against malware attacks. ADAM is built by connecting different building blocks such as transformation of malware samples, scanning and analysis of malwares. These blocks help to test different anti-malwares against malware variants. ADAM is not always able to avoid anti-malware tool so, better detection mechanism is not always provided by it. Lastly, ADAM can be extensible to support new implementations of malware transformations and detection evaluations [3].

ADAM aims for the following design goals:

1. Security analysis
2. Automated transformation
3. Generic application
4. Extensibility

2.2 Automatic Code Obfuscation

Code obfuscator is used to create source code or machine code that is not easy for humans to understand. They are

programs which transform readable code using various techniques into obfuscated code. By using source message object code is created which is then obfuscated and passed to the server. The server sends it to client. The reverse operation is done to get the original source code. Although the system can trace the software pirates easily but it remains secret until the powerful de-obfuscator to be built[10].

So, obfuscated software version release should be within short period. A variety of tools exist to perform or assist with code obfuscation. There exists de-obfuscation tools that at-tempt to perform the reverse transformation. Obfuscation make a program difficult for reading, writing, reverse-engineering and time-consuming, but not necessarily impossible. Some anti-virus software, like AVG, will also alert their users when they land on a site with code obfuscated, as to hide malicious code can be one of the purposes of obfuscation. This decreases security.

2.3 Effective and Efficient Malware Detection at the End Host

This technique analyzes a malware that characterizes its behavior and build a model. Then, extract the slices of program responsible for such information flows. For detection, execute these slices to match with these models of an unknown program against the runtime behavior. The main limitation is that it cannot generate system call signatures for the slicing process. The goal of this system is to effectively and efficiently detect malicious code at the end host [7]. There is a challenging problem in crafting the malware samples and malware authors to have full freedom for this activity. To attack this problem, system operates by generating detection Models based on the observation of the execution of malware programs. It extracts the behavior that characterizes the execution of this program. The behavior is then automatically translated into detection models operating at the host level.

2.3 Crowdroid

This is a behavioral based malware detection system. Using detector that is embedded in the entire framework for a collection of traces from unlimited real users based on crowdsourcing. There is a central server in which the system analyzed collected data is stored using two types of data sets artificially created malwares and real malwares[8]. It is an effective method of isolating the malware and alerting the users about the downloaded malwares. Crowdroid collects system calls of running apps and applies clustering algorithms to differentiate between benign and malicious apps on mobile devices.

Behavior-based malware detection System framework:

This framework having several components which provide resources and mechanisms to detect mal-ware on the Android platform. First a lightweight client called Crowdroid which can be downloaded and installed from Google Market is developed. This application is in charge of monitoring Linux Kernel system calls and sending them preprocessed to a centralized server. According to a crowdsourcing philosophy, users will help with sending non-personal, but behavior-related data of each application they use.

TABLE I
COMPARISON OF EVALUATION TECHNIQUES

Methods of Evaluation	Transformation Technique	Limitations
ADAM (Automated Detection of Android Malware)	Repackaging and Obfuscation Technique	Not always capable of providing better detection mechanism
Automatic Code Obfuscation	Code Obfuscator Tool	Obfuscator remains secret until deobfuscator built
Effective and Efficient malware detection at the End Host	Novel Malware Detection Approach	Not capable of generating system call signatures
Crowdroid	Based on Crowd Sourcing and Clustering	Not suitable of large set of malwares and security issues
Scalable and Zero-day Android Malware Detection	Risk-Ranker Automatic System	Results in extra overhead in processor draining battery faster

2.4 Scalable and Accurate Zero-day Android Malware Detection

This technique does not depend on malware samples and their signatures. It is an automated system to scalable analyze whether a particular app exhibits dangerous behavior, this system is called as RiskRanker. It checks and translates potential security risks into corresponding detection modules of two orders of complexity[9]. The first-order modules by evaluating the risks in a straightforward manner handles non-obfuscated apps . The second-order modules capture certain behaviors to detect malware.

3. System Design

In this paper, we focus on the evaluation of anti-malware tools for Android Operating System. We attempt to deduce the kind of signatures that these products use to detect malware and we also find out that how resistant and robust these signatures are against changes in the malware binaries. Different types of transformations are developed that can be

applied on the malware samples that can preserve their malicious behavior.

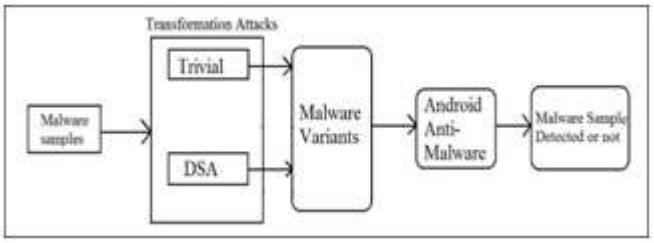

Fig. 1. Evaluating Anti-malware

3.1 Trivial Transformations

This transformation technique does not include any code level changes. Repacking, Disassembling and Reassembling, Changing package name are the Transformations included in this technique.

1. Repacking- Applications are signed with custom keys. When unzipped and repacked ,the custom keys are changed. Thus generating new variant of malware.
2. Disassembling and Reassembling- The bytecode stored in classes.dex file of the application package can be disassembled and reassembled. Also the contents of the bytecode can be rearranged. It also affects the signatures of the package.
3. Changing package name- The package name is stored in the AndroidManifest and is unique to the application. Package name is changed by any malicious application to another name.

3.2 Transformation Attacks Detectable by Static Analysis (DSA)

This Transformation technique includes the code level changes.

1. Identifier Renaming- Class, method, and field identifiers in bytecode mostly can be renamed. Identifier renaming is pro-vided by several free obfuscation tools such as ProGuard[15].
2. Data Encoding- The array data and strings in the dex file are stored in the encoded form. It affects the signatures of the package.
3. Call Indirections- It is a simple way to manipulate call graph of the application to defeat automatic matching. Given a method call, the call is converted to a call to a previously non-existing method that then calls the method in the original call. This can be done for all calls, those going out into framework libraries as well as those within the application code.
4. Code Reordering- The instructions in the method of a code are reordered. goto instructions are inserted to preserve the runtime execution sequence of the instructions.
5. Junk code Insertion- Code sequences that does not affect the rest of the code is inserted. Junk code does not affect the semantics of the code.
6. Remove Debug Information- Debug information, such as source file names, local and parameter variable names, and source line numbers can be removed.

4. Implementation

Android studio is used for implementation which provides android environment for emulator. Android Studio is an IDE for developing Android Applications and it is based upon IntelliJIDEA[13]. On top of the capabilities you expect from IntelliJ, Android Studio offers:

-Flexible system for Gradle-based build -Multiple apk file generation

-Rich layout editor

-Compatibility with various versions -ProGuard and app-signing capabilities

Moreover, APKTool is used for disassembling and assembling transformations. It is a tool for reverse engineering 3rd party, closed, binary Android apps[14].

It decodes resources to nearly original form and after making some modifications rebuild them; it makes possible to step by step debug smali code. Also it makes working with an app easier because of project-like file structure and automation of some repetitive tasks like building apk, etc.

Features:

-Disassembling resources to nearly original form (including classes.dex, resources.arsc, XMLs, and 9.png.)

-Rebuilding decoded resources back to binary APK/JAR -Handling and organizing APKs

-Debugging of Smalifiles(to be removed in 2.1.0 in favor of IdeaSmali)

-Helping with repetitive tasks

Another tool named Keytool is also used for implementation purpose. Keytool stores the keys and certificates in what is called a keystore[14]. By default the keystore is implemented as a file. It protects private keys with a password. It is a certificate and key management utility. It enables users to administer their own public/private key pairs and for use in self-validation (where the user authenticates himself to other users and services) or integrity of data and service authentication and validation using digital signatures. Also the keytool command enables users to cache the public keys (in the form of certificates) of their communicating peers.

Code level transformations are implemented in Smali/Baksmali language. Smali/Baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation[11]. We perform some combinations of transformations on malware samples as shown in fig 2. There is no any pre-planned order for applying those transformations.

Code	Technique
P	Repack
A	Dissassemble & assemble
RP	Rename package
EE	Encrypt native exploit or payload
RI	Rename identifiers
RF	Rename files
ED	Encode strings & array data
CR	Reorder code
CI	Call indirection
JN	Insert junk code

Fig. 2. Set of Transformations

5. Results

Malware samples should show changes in their behaviour after transformation along with their signatures. The Table II shows the evaluation of few anti-malware system against Basebridge, Geinimi and Benign malware samples. We have analysed that the anti-malware system gets evaded by re-spective malwares, performing some particular transformations or a combination of those. For example Basebridge evades Lookout by just performing Repack (P) transformation. Our results includes whether anti-malware detects the transformed malware sample or fails to detect it.

The following fig. shows results of Basebridge and Geinimi malware sample against Dr.Web.

TABLE II

RESULTS OF BASEBRIDGE, GEINIMI AND BENIGN MALWARE SAMPLESEVADING RESPECTIVE ANTI-MALWARE. 'X' SHOWS THAT ANTIMALWARE HAVE DETECTED ALL TYPES OF TRANSFORMATIONS.

	Basebridge	Geinimi	Benign
AndroHelm	P,A	P,A	A
ESTSoft	P,A	P	RP
AVG	X	X	X
ESET	X	X	X
Dr. Web	X	X	X
Trend M	P,A	X	RP
Zoner	RP	X	P
Lookout	P	A	X

Fig. 3. Basebridge malware sample detetcted

Fig. 4. Geinimi malware sample detetcted

6. Conclusion

Antimalwares are becoming robust and they are now quite efficient to face the various malware attacks. Various malware samples were undegone through Trivial and DSA transformation and the results that whether they evade the anti-malware system or not is shown in Table II . The Table II shows which malware sample evades the anti-malware through respective transformation. These transformations can be refered through Fig. 2 which shows the short form of the various transformations.

7. Acknowledgement

We express our sincere gratitude to Mrs. M. A. Potey, Head of Department, Computer Engineering, for her assistance, persuasion, and an incent to work better.
Our deepest gratitude goes to our project guide, Mrs. N. S. Patil, for her guidance, ideas, help, encouragement, interpre-tations and suggestions which helped us in the realization of our objective and coordinate as a team.
We are extremely thankful to our project incharge, MrsShanthi Guru for her comments, introspections and support which helped us throughout our work.

REFERENCES

[1] VaibhavRastogi, Yan Chen, and Xuxian Jiang, " Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks", IEEE transactions on information forensics and security VOL. 9, NO. 1, Jan 2014.
[2] V. Rastogi, Y. Chen, and X. Jiang, DroidChameleon: Evaluating Android anti-malware against transforma-tion attacks, in Proc. ACM ASIACCS, May 2013, pp. 329334.

[3] M. Zheng, P. Lee, and J. Lui, ADAM: An automatic and extensible platform to stress test Android anti-virus systems, in Proc. DIMVA, Jul. 2012, pp. 120.
[4] C. Collberg, C. Thomborson, and D. Low, A taxonomy of obfuscating transformations, Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997.
[5] M. Christodorescu and S. Jha, Testing malware detectors, in Proc. ACM SIGSOFT Int. Symp. Softw. Test. Anal., 2004, pp. 3444.
[6] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, Synthesizing near-optimal malware specifica-tions from suspicious behaviors, in Proc. IEEE Symp. SP, May 2010, pp. 4560.
[7] C. Kolbitsch, P. Comparetti, C. Kruegel, 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. 351366.
[8] Iker Burguera and UrkoZurutuza, SiminNadjm-Tehrani, Crowdroid: Behavior-Based Malware Detection System for Android, in ACM, October 17, 2011, pp. 1-11.
[9] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, RiskRanker: Scalable and accurate zero-day android mal-ware detection, in Proc. 10th Int. Conf. Mobile Syst., Appl., Services, 2012, pp. 281294.
[10] Seemadevi M. Shelake, Prof. Prakash. B. Dhainje, Dr. Deshmukh Pradeep K., Evaluating efficiency of Anti-Malware using Transformation Attacks, in International Journal of Advanced Research in Computer Science and Software Engineering,Volume 5, Issue 3, March 2015, pp. 773-777.
[11] (2016,24 Jan) Smali - https://www.quora.com/What-issmali-in-Android
[12] RafealFedler, Julian Schutte, Marcel Kulicke, On the Ef-fectiveness of Malware Protection on Android an Evalua-tion of Android Antivirus Apps, in Fraunhofer Research Institution For Applied And Integrated Security, April 2013, pp. 1-36.
[13] (2016,12 Jan) Android Studio - http://develop-er.android.com/sdk/index.html
[14] (2016,12 Jan) Apktool - https://apkudo.com/?p=775
[15] ProGuard - http://proguard.sourceforge.net/.