Black Box Analysis of Android Malware Detectors

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Guruswamy Nellaivadivelu

May 2017

The Designated Project Committee Approves the Project Titled

Black Box Analysis of Android Malware Detectors

by

Guruswamy Nellaivadivelu

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2017

Dr. Mark Stamp	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science

ABSTRACT

Black Box Analysis of Android Malware Detectors

by Guruswamy Nellaivadivelu

One of the challenges of combating mobile malware in Android devices is attributed to code obfuscation. Malware writers obfuscate the source code of their programs by employing various techniques that attempt to hide the true intent of the program. Malware detectors use a handful of features to classify a program as a malware. If the malware detector uses a feature that is obfuscated, then the malware detector will fail to classify it as a malicious software. In this survey, we look at works of literature that discuss code obfuscation in malware and the challenges faced by malware detectors to combat the code obfuscation.

ACKNOWLEDGMENTS

I would like to express my deep gratitude to Dr. Stamp for his valuable guidance and supervision throughout the project work. I am also grateful to my committee members for reviewing my work and providing constructive feedback.

TABLE OF CONTENTS

CHAPTER

APPENDIX

LIST OF FIGURES

CHAPTER 1

Introduction

Android malware proliferation is rising exponentially. In the second quarter of 2016, 3.5 million Android malware were detected [1]. This rapid increase in Android malware has placed the focus on Android security and made it imperative to develop more efficient defensive tools for combating malware. One of the challenges faced in this area is the use of code obfuscation techniques. Code obfuscation is a method of altering a source code to hide its actual purpose. There are many ways of obfuscating a source code in an Android environment. Several software applications that are available off the shelf can be used to achieve different levels of code obfuscation [2]. In order to address the problem of strengthening malware detector's strength, there are two fundamental questions that need to be addressed, as highlighted by Christodorescu et al. [3]:

1. Question 1: How resistant is a malware detector to obfuscations or variants of known malware?

2. Question 2: Can using limitations of a malware detector in handling obfuscations determine its detection algorithm?

The two questions above can be used as a way to gauge how good a malware detector will perform against obfuscated code.

1.1 Code Obfuscation

Code obfuscation is the process by which source code is manipulated to hide its true intentions. Code obfuscation is increasingly becoming a common tool to avoid detection by traditional malware detectors.

There are many different types of code obfuscation. The most basic type of code obfuscation involves the encryption of all the strings that are used in the code.

This overrides the detection mechanism of most of the traditional malware detectors. Some advanced malware detectors account for this encryption and are able to identify malware files. There are a host of other obfuscation techniques that can be employed by malware writers. Some of these include the obfuscation of function calls, permission hiding, and insertion of dead code.

1.2 Challenges

The challenges associated with code obfuscation primarily deal with the problem of maintaining the core functionality of the code, while making it difficult for malware detectors to detect their true purpose. This challenge becomes easier for malware writers when dealing with Android malware. The reason for this is associated with the permission levels of applications running on Android platform. Unlike anti-virus programs that run on computers, the Android system provides the same set of permission levels to the anti-virus application and the application that is being scanned. This is a major limitation for malware detector writers.

With the advent of more sophisticated tools for the encryption of source code for malware files, it is becoming increasingly difficult to differentiate instances where obfuscation is used for a genuine security reason and instances where it is being used with a malicious intent.

1.3 Objective

The primary objective of this project is to make malware detectors more responsive to the code obfuscation techniques employed by malware writers. The objectives can be broken down into the following two points:

1. 1.Identify the malware features that are used by a malware detector by encrypting it.

2. 2.Modify an existing malware detector to overcome the limitations of code

obfuscation.

The first step in implementation will be the identification of the factors in a malware that are taken into consideration by a malware detector. To achieve this, we will begin by encrypting various parameters of a malware and running it through a malware detector [4]. By following this approach, we can identify the exact scenario when a malware is no longer classified as a malware by our malware detector. Once we identify the features that are required by a malware detector, we will use this information to make the malware detectors process the obfuscated part of the code as well. This will make our malware detector more robust and improve their performance.

CHAPTER 2

Literature Survey

2.1 Introduction

In this chapter, we present the results of a literature survey that was performed to identify the current state of obfuscation mechanisms and their impact to the field of code obfuscation.

2.1.1 Code Obfuscation and Malware Detectors

The efficiency of malware detectors against code obfuscation has been a point of discussion amongst malware researchers for a very long time. A lot of research has been done on the robustness of malware detectors against high levels of obfuscation [3]. The issue of malware detector's strengths against obfuscated malware had been discussed as early as 1996, as can be seen in the quote by S.Gordon and R.Ford [4]:

'The evaluation of anti-virus software is not adequately covered by any existing criteria based on formal methods. The process, therefore, has been carried out by various personnel using a variety of tools and methods.'

2.1.2 Program Obfuscation

There has been a lot of theoretical research on the different aspects of obfuscation and on ways to improve it. Most of this research has been successful in arriving at a conclusion on the efficiency of the cryptographic problems of encryption, authentication and protocol [5]. But the problem of program obfuscation has remained an area within cryptography in which theoretical research has been inadequate. In their seminal paper on program obfuscation, Barak et al. [5] propose to represent program obfuscation as below: An obfuscator O is said to be an efficient compiler if it takes as input a program P and produces a program O(P) and satisfies the following two conditions:

1. Functionality: O(P) computes the same function as P

2. 'Virtual Black Box'property: Anything that can be efficiently computed from

O(P) can also be computed by P.

The paper by Christodorescu et al. [3] lists various ways to test and achieve program obfuscation in general. A detailed analysis of the various obfuscation methods is also discussed in the paper. One interesting angle explored by the paper deals with assigning mathematical equations to measure the effectiveness of the individual obfuscators. This lets us quantify the different obfuscators and rank them against each other. One of the evasion methods employed in malware obfuscation is polymorphism. It is a method by which a program evades various detection tools by mutating into different forms. In the paper by Rastogi et al. [6], the authors develop and propose a framework called 'DroidChameleon 'that provides a way to transform Android applications into different forms with minimal user involvement.

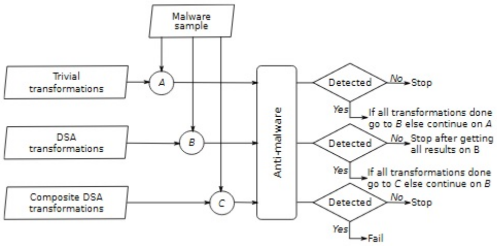
Malware sample
Trivial transformations
DSA transformations
Composite DSA transformations
A
B
C
Anti-malware
Detected
No
Stop
Yes
If all transformations done go to B else continue on A
Detected
No
Stop after getting all results on B
Yes
If all transformations done go to C else continue on B
Detected
No
Stop
Yes
Fail

Figure 1: Evaluating anti-malware

As shown in Fig. 1, the authors apply various transformations on a malware sample dataset. The output of all these transformations are processed by a malware detector (referred here as Anti-malware). The input to the anti-malware is processed sequentially. After each transformation, the anti-malwareâĂŹs output is evaluated and if the malware detection fails, the next level of transformation is applied. This helps rank the various malware detectors against each other for accurate analysis.

2.2 OBFUSCATION IN ANDROID MALWARE

A report by Google stated that a majority of malware detectors work as a binary classifier [7]. They classify an application as a malware or a benign file. In order to effectively eliminate malicious applications, it is important that malware detectors do more than just identify malware. They should be able to isolate the core parts of the application that perform the malicious acts and work at fixing the loopholes that let the program act in a malicious way. More recent malware applications employ a variety of tricks, in addition to traditional code obfuscation mechanisms. For instance, a variant of Android malware, known as Android/BadAccent, is a known banking Trojan, that steals credentials used in banking applications [8]. A variant of this malware used a mechanism known as âĂIJTapjackingâĂİ to extract the credentials from the users. In this form of attack, a screen is displayed to the user, while a second screen is hidden behind the actual visible display [9]. When a user clicks a button on the screen, assuming it to be the one that is displayed, the underlying screen gathers the input and processes the command. This is a common method of gathering details from unsuspecting users.

2.2.1 STATISTICAL ANALYSIS TECHNIQUES AND ANDROID MALWARE

One widely used approach for analyzing malware samples is the usage of statistical methods. In such methods, the Android executable file (with the extension apk), is decompiled to get the original source code. Due to the Android operating system being written in Java, it is easy to reverse engineer an apk file to retrieve the source code. This opens up many opportunities for performing statistical analysis on the obtained raw data. This also lets a researcher perform various operations on the source code, and then repackage it back into an apk. In the approach known as AndroSimilar, Faruki et al. [10] propose a new algorithm that takes into consideration various features that are known to be present in malware alone. The AndroSimiar approach, as shown in Fig. 2 decompiles an apk file and repackages it after feature extraction. To extract the features, the algorithm incorporates apps from the Google Playstore and other third party applications. These features are normalized and fed into a signature generation engine, that provides a unique signature for each malware. This is used as reference for detecting future malware applications.

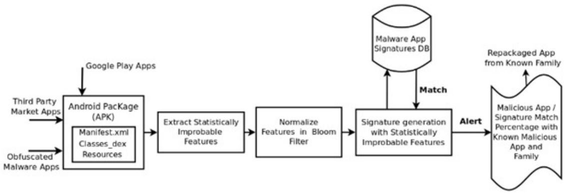
Google Play Apps
Third Party Market Apps
Obfuscated Malware Apps
Android PacKage (APK)
Manifest.xml
Classes_dex
Resources
Extract Statistically Improbable Features
Normalize Features in Bloom Filter
Signature generation with Statistically Improbable Features
Malware App Signatures DB
Match
Alert
Repackaged App from Known Family
Malicious App / Signature Match Percentage with Known Malicious App and Family

Figure 2: AndroSimilar

2.3 Conclusion

Malware in mobile devices is no longer a problem confined to labs and research areas. The rapid increase in access to computers has helped malware writers create specific, targeted programs that perform with high efficiency and exploit vulnerabilities in different operating systems. The amount of research being done in malware analysis and, more specifically, in Android malware, is in the right direction. In the fight against sophisticated metamorphic malware, it is imperative that the malware detector is better than the malware creator. In this paper, we have explored various work, that dealt with the different aspects of malware obfuscation and ways to overcome the shortcomings in today's version of malware detectors. The future of malware looks very bright and it is hoped that the malware detectors of the future will be up to the task at hand.

CHAPTER 3

Code Obfuscation

Code Obfuscation is a technique by which programmers have deliberately sought to make the functionality of their code less obvious. This technique has been used by programmers to achieve various additional objectives.

Some of the advantages of code obfuscation, that make it very popular amongst software developers are:

- Prevention of reverse engineering.
- Protection of intellectual property.
- Reducing the size of an executable.
- Protection of licensing mechanisms.
- Restricting unauthorized access.

We will look at the history of code obfuscation to appreciate the relevance of code obfuscation in today's software development perspective.

3.1 Growth of Obfuscation in Software Development

Code obfuscation has been historically associated with malware development, than with benign software development. Some of the earliest examples of attempts at obfuscation in malware can be found in the B̈rain Virus" [?] . In this variant of the malware, the malicious program would display unaffected disk partitions to users attempting to access partitions that the virus had corrupted. Although the code in itself was not encrypted, the behavior of the virus shows attempts at hiding its true usage.

In the same year, the Cascade virus was released to the world. This was an early variant of malware to use encryption to hide its true purpose. The earliest strains of obfuscated malware used a simple encryption-decryption routine to perform the decryption tasks. As the malware detectors of the time were not sophisticated enough to detect the encrypted part of the code, this simple obfuscation technique enabled a lot of malware programs to slip away undetected. This is a serious disadvantage in the design and implementation of malware detectors. We would be exploring more such flaws with the implementation of malware detectors in this project.

With the advent of advanced malware detectors and improvement in statical analysis techniques, the level of obfuscation in malware increased. Polymorphic malware uses a very high level of encryption technique to obfuscate its contents. A polymorphic malware changes the encryption in itself and provides very few traces of a signature. If a malware is truly polymorphic, then there will be no consistency between any two iterations of the same program and it would be virtually impossible to detect them using traditional signature matching techniques.

3.2 Malware Detectors

Malware detectors came into existence with the advent of different malicious programs. Before the rapid growth of the internet, malware detectors were only capable of performing scans based on signatures of known virus programs. This static analysis technique meant that new virus would be out in the wild for some time before the malware definitions of the individual anti virus programs could be updated.

With the introduction of the world wide web, the antivirus industry expanded into dynamic analysis and cloud based malware detectors. Firewalls, online scanning, and virtual machines started being increasingly used to identify malware. One major shortfall of anti virus programs is their inability to detect polymorphic virus. The various methods employed by antivirus programs include:

1. Signature Based Detection

2. Heuristics Based Detection

3. Rootkit Detection

4. On-Access Scanning

The functioning of theses methods, their limitations, and their relevance to this experiment are discussed in the next few pages.

3.2.1 Signature Based Detection

This is one of the most basic methods of malware detection that is still in use today. When a new strain of malware is detected in the "wild", antivirus firms analyze it and extract a "signature" from it. This signature extraction can either be done manually or by using automated signature detection techniques [11]. Once a signature is detected, it is updated into various malware definitions of antivirus software.

Although this method is effective against generic malware, it is highly ineffective against oligomorphic, polymorphic and metamorphic malware. These are variants of malware that encrypt itself with each iteration. In this project, we attempt to identify the various factors that contribute to malware detection and their importance in overcoming the signature detection method.

3.2.2 Heuristics Based Detection

In Heuristics based detection techniques, a single signature or pattern is used to detect multiple malware belonging to the same family. Such techniques rely on the fact that multiple malware are created from a single malware. Thus, successfully creating a signature for a base family will result in the detection of all malware related to that particular family.

3.2.3 Rootkit Detection

A rootkit is a type of software that attempts to gain administrator privileges in a system without the knowledge of the user running it. In many cases, the rootkits contain software within them that becomes undetectable to antivirus programs. Rootkits usually have full administrative access and also have the ability to hide themselves from the list of running processes. Modern antivirus software scans for rootkits in specific, to detect them. It is very difficult to remove a rootkit when compared to other generic malware programs.

3.2.4 On-Access Scanning

In this method, the antivirus program looks out for any threats that might happen on a real-time basis. The antivirus monitors the system in which it is installed and looks for suspicious activity whenever the computer's memory is loaded with fresh data from the storage disks. This might happen when a USB drive is inserted, an email attachment is opened or a even when an already existing file is opened by a user or a program. This type of scanning is more effective as it does not rely solely on malware definitions to detect viruses.

3.3 Android Malware Detectors

LIST OF REFERENCES

[1] M. A. D. F. D.Emm, R.Unuchek, ‘‘It threat evolution in q2 2016,’’ 2016, accessed 2016-10-10.

[2] A.Apvrille and R.Nigam, ‘‘Obfuscation in android malware, and how to fight back,’’ *Virus Bulletin*, pp. 1--10, 2014.

[3] M.Christodorescu and S.Jha, ‘‘Testing malware detectors,’’ in *ACM SIGSOFT international symposium on Software testing and analysis*, New York, NY, USA, 2004, pp. 34--44.

[4] S. Gordon and R. Ford, ‘‘Real world anti-virus product reviews and evaluations--the current state of affairs,’’ in *Proceedings of the 1996 National Information Systems Security Conference*, 1996.

[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, ‘‘On the (im) possibility of obfuscating programs,’’ in *Annual International Cryptology Conference.* Springer, 2001, pp. 1--18.

[6] V. Rastogi, Y. Chen, and X. Jiang, ‘‘Droidchameleon: evaluating android anti-malware against transformation attacks,’’ in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security.* ACM, 2013, pp. 329--334.

[7] Google, ‘‘Android security 2014 year in review,’’ https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2014_Report_Final.pdf, accessed 2016-12-01.

[8] S. Rasthofer, I. Asrar, S. Huber, and E. Bodden, ‘‘An investigation of the android/badaccents malware which exploits a new android tapjacking attack,’’ Technical report, TU Darmstadt, Fraunhofer SIT and McAfee Mobile Research, Tech. Rep., 2015.

[9] Q. A. Chen, Z. Qian, and Z. M. Mao, ‘‘Peeking into your app without actually seeing it: Ui state inference and novel android attacks,’’ in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 1037--1052.

[10] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, ‘‘Androsimilar: robust statistical feature signature for android malware detection,’’ in *Proceedings of the 6th International Conference on Security of Information and Networks.* ACM, 2013, pp. 152--159.

[11] K. Ask, ‘‘Automatic malware signature generation,’’ Ph.D. dissertation, 2006.

APPENDIX A

Zorak Likes Beans

A.1 Oh Yes He Does

Appendices can have sections and subsections and so on.

A.2 Really

Sections, subsections, or whatever should come in pairs.

APPENDIX B

Everybody Wants to Be Space Ghost

Space Ghost: Everybody wants to be Space Ghost

Everybody near and far

Hey, ma, look at me, I'm on TV

Everybody wants to be a star

I'm Space Ghost, Mr. Space Ghost

I've got big muscles, And I can dance

When Zorak tries to bug me, I zap him with my power bands

Zorak: Uhhhh… I don't think that will be necessary

Space Ghost: I think I'll zap him with my power bands

On Saturn and Jupiter and Neptune, too

I've been hearing it from coast to coast

I'd give anything, If for just one day

I could be a super hero like Space Ghost

Zorak: Ugghh… Why would anyone want to be Space Ghost?

Space Ghost: Everybody wants to be just like me!