# A Survey of Evaluation Techniques for Android Anti-Malware using Transformation Attacks

Omkar Yeshvekar, Snehal Zende, Deepti Walvekar, Namrata Wabale, and Akash Korde
Student, D. Y. Patil College of Engineering, Akurdi, Pune

Mrs. M. Saravanapriya, Mrs. Nilam S. Patil
Assistant Professor, D. Y. Patil College of Engineering, Akurdi, Pune

***Abstract*****—**Android an open-source operating system mainly used for mobile phones have become increasingly popular. Studies suggest that mobile malware threats have recently become a real concern and the impact of malware is getting worse. 2014 saw an astounding 75 percent increase in the Android mobile malware. It is therefore imperative to evaluate the resistance and robustness of anti-malware products for android against various malware. To evaluate existing anti-malware, a systematic framework called DroidChameleon is developed with several common transformation techniques. This survey examines the effectiveness and robustness of popular antimalware tools and compare them against one another aiding in the decision making process involved with developing a secure system.

***Index Terms*** *- Android, Malware, Anti-malware, Transforma-tion*

*****

## I. Introduction

Malware had a tremendous impact on the world as we know the rising number of computer security suggests that malware is an epidemic. Devices such as smartphones and tablets are becoming increasingly popular but this popularity attracts malware authers too. Now a days, operating system such as android is “clearly today’s target”. With the growth of malware we have also seen an evolution of anti-malware tools.

Malware can be described as a program whose objective is malevolent. In this paper, we have described the efficacy of anti-malware tools. To evaluate existing anti-malware soft-ware, a systematic framework called DroidChameleon with several common transformation techniques that may be used to transform Android applications automatically is described. Some of these transformations are highly specific to the Android platform only[1]. Based on the framework, we pass known malware samples(from different families) through these transformations to generate new variants of malware, which are verified to possess the originals malicious functionality. We use these variants to evaluate the effectiveness and robustness of popular anti-malware tools.

## II. Methods of evaluation of anti-malware

### A. ADAM-

ADAM, an automated system for evaluating the detection of Android malware. ADAM is an extensible platform which is automatic, generic and able to evaluate the Android malware detection systems. ADAM applies different transformation techniques to generate different variants of each Android malware sample, and evaluates the effectiveness of different smart-phone malware detection systems in identifying such malware variants[3]. ADAM is able to automatically transform an original malware sample to different variants using repackaging and obfuscation techniques in order to evaluate the strength of different anti-malware systems against malware attacks. ADAM is built by connecting different building blocks such as transformation, scanning and analysis of malwares. These blocks help to test different anti-malwares against malware samples. But ADAM is not always able to avoid anti-malware tool. So, it will not always provide the better detection mechanism. ADAM then evaluates the detection of these variants against different smart-phone malware detection systems. Such malware transformations and detection evaluations are generic enough to support heterogeneous malware samples and malware detection systems, respectively. Lastly, ADAM can be extensible to support new implementations of malware transformations and detection evaluations.

ADAM aims for the following design goals:
1. Security analysis - ADAM checks whether an Android-based malware sample in .apk format can be detected by an existing anti-virus system. For this analysis, we do not need the source code of the malware sample.

2. Automated transformation - ADAM can automatically transforms a malware sample into different malware variants, while preserving the original malicious behavior. No manual modification of a malware sample is required.

3. Generic application - ADAM can be applied for general classes of Android-based malware samples and malware detection systems.

4. Extensibility - ADAM provides an interface that can easily integrate new implementations of transformation techniques and detection methodologies.

ADAM is composed of different building blocks. Figure 1 illustrates how different building blocks are involved in testing malware samples against anti-virus systems. Let us now describe how each building block works, and argue how the building blocks can be extended for different variants of implementation.
Transformation : implement two classes of transformation techniques: repackaging and code obfuscation.

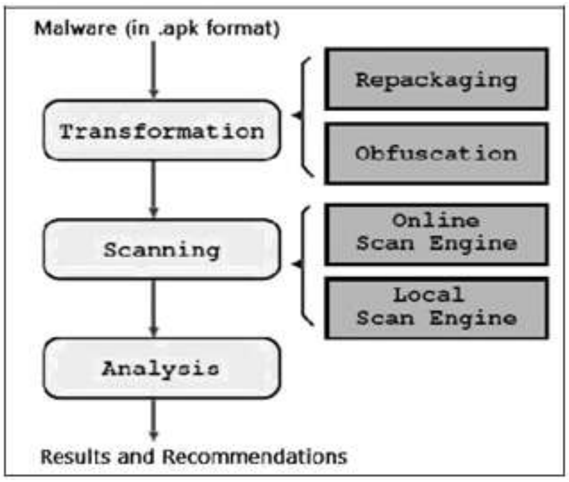

Fig. 1. Design flow of ADAM[3]

Scanning : support two types of scan engines - online and local.

Analysis : The analysis results can be summarized and presented, so as to provide recommendations for antivirus vendors to evaluate the effectiveness of the state of the art of malware detection for Android.

B. Automatic Code Obfuscation-
A tool named code obfuscator is used which converts the program into an equivalent one that is more difficult to understand and for reverse engineering[4]. It is done to protect the messages which help to preserve privacy policies between sender and receiver. As shown in Figure 2 obfuscation technique provides the protection of messages between Alice and Bob. By using source message object code is created which is then obfuscated and passed to the server. The server sends it to Bob i.e. client. The reverse operation is done by Bob to get the original source code. Although the system can easily trace the software pirates but it remains secret until the powerful de-obfuscator to be built. So, obfuscated software version release must be within short period.A variety of tools exist to perform or assist with code obfuscation. These include experimental research tools created by academics, hobbyist tools,commercial products written by professionals, and open-source. There also exist de-obfuscation tools that attempt to perform the reverse transformation. Obfuscation can make reading, writing and reverse-engineering a program difficult and time-consuming, but not necessarily impossible. Some anti-virus software, such as AVG, will also alert their users when they land on a site with code obfuscated, as one of the purposes of obfuscation can be to hide malicious code. This decreases security.

C. Effective and Efficient Malware Detection at the End Host-
Clemens Kolbitsch proposed a novel malware detection approach that is both effective and efficient, and thus, can

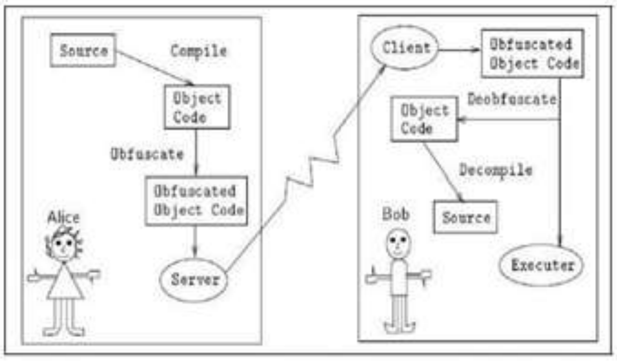

Fig. 2. Protection through obfuscation[4]

be used to replace old anti-virus tool at the end host[7]. This technique analyzes a malware to build a model that characterizes its behavior. Then, extract the program slices responsible for such information flows. For detection, execute these slices to match with these models against the runtime behavior of an unknown program. The main limitation is that it cannot generate system call signatures or find a starting point for the slicing process. The goal of our system is to effec-tively and efficiently detect malicious code at the end host. Moreover, the system should be general and not incorporate a priori knowledge about a particular malware class. Given the freedom that malware authors have when crafting malicious code, this is a challenging problem. To attack this problem, our system operates by generating detection Models based on the observation of the execution of malware programs. That is, the system execute sand monitors a malware program in a controlled analysis environment. Based on this observation, it extracts the behavior that characterizes the execution of this program. The behavior is then automatically translated into detection models that operate at the host level.

D. Crowdroid-
This is a behavioral based malware detection system for Android. They used detector which is embedded in an overall framework for a collection of traces collected from unlimited real users based on crowd sourcing. The system analyzed collected data in central server using two types of data sets: artificially created malwares and real malwares. It is an effective method of isolating the malware as well as alerting the users about the downloaded malwares. When it is actually going to apply on mobile, it might result an extra overhead in the processor, causes a faster battery drain. VirusMeter proposes to detect malware based on abnormal power consumption caused by malware. Crowdroid collects system calls of running apps on mobile devices and applies clustering algorithms to differentiate between benign and malicious apps[8]. Dixon proposes a system to detect malicious code by correlating power consumption pattern with the users location. Survey of the current situation of mobile malware on three popular smartphone platforms

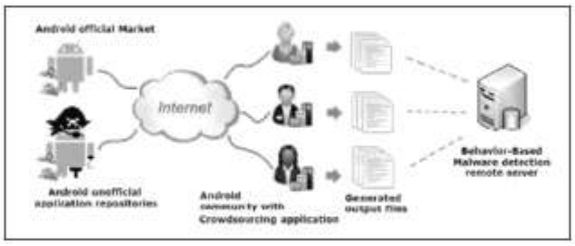
Android official Market
Internet
Android unofficial application repositories
Android community with Crowdsourcing application
Generated output files
Behavior-Based Malware detection remote server

Fig. 3. Behavior-Based Malware Detection Framework[8]

(iOS,Android and Symbian). DroidMOSS detects repackaged apps in third-party Android markets. DroidRanger is different from these systems in not detecting mobile malware on mobile devices (under resource constraints such as limited battery or CPU). Instead, it performs offline analysis to detect malware in current Android Markets. Accordingly,it needs to address different challenges by accommodating a large number of apps.

Behavior-based malware detection System framework :
Our framework is composed of several components which provide enough resources and mechanisms to detect mal-ware on the Android platform. First, we have developed a lightweight client called Crowdroid, which can be down loaded and installed from Google's Market. 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. These applications could have been downloaded both from the official Market and also from unofficial repositories.

### E. Scalable and Accurate Zero-day Android Malware Detection-

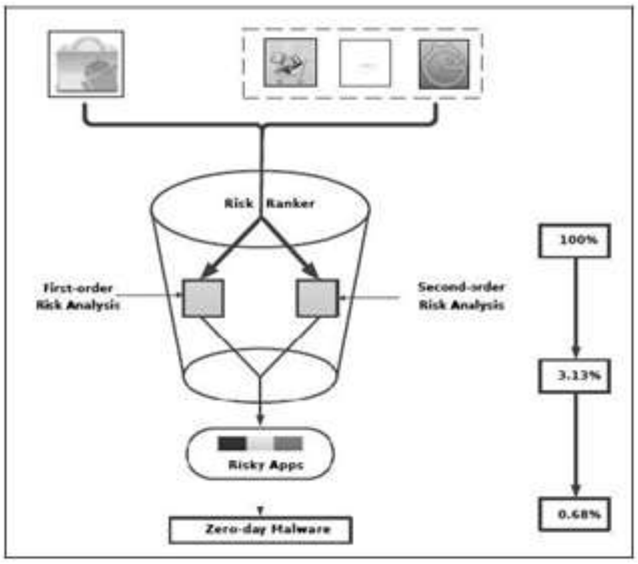

Fig. 4. The RiskRanker architecture[9]

Michael Grace et al. proposed proactive scheme to spot zero-day Android malware, It does not depend on malware samples and their signatures. It is an automated system called RiskRanker to scalable analyze whether a particular app exhibits dangerous behavior. Figure

3. Shows the overall architecture of RiskRanker. It checks and translates potential security risks into corresponding detection modules of two orders of complexity[9]. The first-order modules handle non-obfuscated apps by evaluating the risks in a straightforward manner; the second-order modules capture certain behaviors (e.g. encryption and dynamic code loading) to detect malware.

## III. Framework Design

In this paper, we focus on the evaluation of anti-malware products for Android. Specifically, we attempt to deduce the kind of signatures that these products use to detect malware and how resistant these signatures are against changes in the malware binaries.

We classify our transformations as trivial (which do not

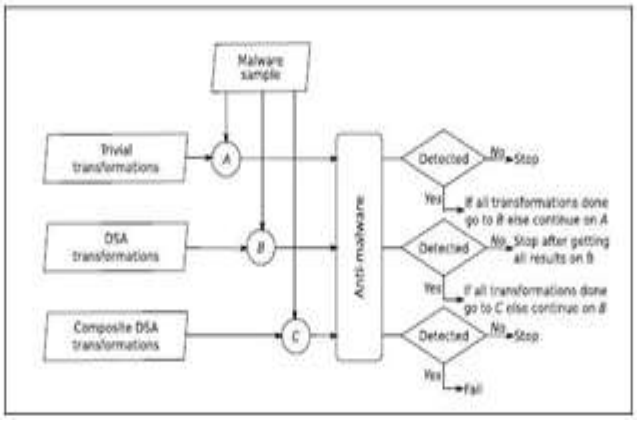
Malware sample
Trivial transformations
DSA transformations
Composite DSA transformations
A
B
C
Anti-malware
Detected
No
Stop
Yes
If all transformations done go to B else continue on A
Detected
No
Stop after getting all results on B
Yes
If all transformations done go to C else continue on B
Detected
No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

Fig. 5. Evaluating anti-malware[1]

require code level changes), those which result in variants that can still be detected by static analysis (DSA), and those which can render malware undetectable by static analysis (NSA). In the rest of this section, we describe the different kinds of transformations that we have in the DroidChameleon framework.

### A. Trivial Transformations-

Trivial transformations do not require code-level changes. We have transformations such as Repacking, Diassembling and reassembling, changing package name in this category.

### B. Transformation Attacks Detectable by Static Analysis (DSA)-

The application of DSA transformations does not break all types of static analysis. Various transformations in DSA are Identifier Renaming, Data Encoding, Call Indirections, code Reordering, Junk code Insertion, Encrypting payloads and native exploits, Other simple and composite transformations.

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

These transformations can break all kinds of static analysis. it focuses on Reflection and Bytecode Encryption transforma-tion.

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 |

## IV. CONCLUSION

In this paper we have described different methods of evaluation of anti-malware such as ADAM, Automatic Code Obfuscation, Effective and Efficient Malware Detection at the End Host, Crowdroid, Scalable and Accurate Zero-day Android Malware Detection against transformation attack. We have also described DroidChameleon framework and various transformation techniques.

## ACKNOWLEDGMENT

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, Mrs Shanthi Guru for her comments, introspections and support which helped us throughout our work.

## REFERENCES

[1] Vaibhav Rastogi, 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: Eval-uating Android anti-malware against transformation 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.

[5] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, “Synthesizing near-optimal malware specifications from suspicious behaviors”, in Proc. IEEE Symp. SP, May 2010, pp. 4560.

[6] 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.

[7] Iker Burguera and Urko Zurutuza, Simin Nadjm-Tehrani, “Crowdroid: Behavior-Based Malware Detection System for Android”, in ACM, October 17, 2011, pp. 1-11.

[8] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, “RiskRanker: Scalable and accurate zero-day android malware detection, in Proc. 10th Int. Conf. Mobile Syst., Appl., Ser-vices, 2012, pp. 281294.

[9] Seemadevi M. Shelake, Prof. Prakash. B. Dhainje, Dr. Deshmukh Pradeep K., “Evaluating efficiency of Anti-Malware using Transformation Attacks”, in International Jour-nal of Advanced Research in Computer Science and Software Engineering,Volume 5, Issue 3, March 2015, pp. 773-777.