Effectiveness of Android Obfuscation on Evading Anti-malware

Melissa Chua
Defence Science and Technology Agency
Singapore
cwanjunm@dsta.gov.sg

Vivek Balachandran
Singapore Institute of Technology
Singapore
vivek.b@singaporetech.edu.sg

ABSTRACT

Obfuscation techniques have been conventionally used for legitimate applications, including preventing application reverse engineering, tampering and protecting intellectual property. A malware author could also leverage these benign techniques to hide their malicious intents and evade anti-malware detection. As variants of known malware have been regularly found on the Google Play Store, transformed malware attacks are a real problem that security solutions today need to address. It has been proven that mainstream security tools installed on smartphones are mainly signature-based; our work focuses on evaluating the efficiency of a composite of obfuscation techniques in evading anti-malware detection. We further verified the trend of transformed malware in evading detection, with a larger and more updated database of known malware. This is also the first work to-date that presents the instability of some anti-malware tools (AMTs) against obfuscated malware. This work also proved that current mainstream AMTs do not build up resilience against obfuscation methods, but instead try to update the signature database on created variants.

ACM Reference format:
Melissa Chua & Vivek Balachandran. 2018. Effectiveness of Android Obfuscation on Evading Anti-malware. In *CODASPY '18: Eighth ACM Conference on Data and Application Security and Privacy, March 19–21, 2018, Tempe, AZ, USA.* ACM, NY, NY, USA, 3 pages. DOI: https://doi.org/10.1145/3176258.3176942

1 INTRODUCTION

The Android platform continues to dominate the mobile platform market with an approximate market share of 84.3% [1]. As mobile devices capabilities improve with better hardware, devices today are able to perform more sensitive operations like mobile banking. These functionalities come in the form of applications (APKs), which can be created by third-party developers and distributed on Google Play. Malware authors have begun tapping the openness of application markets to proliferate malicious APKs. Due to the rising threat of malicious APKs, mainstream commercial solutions have proposed similar methodologies to defend against malware. G. Hatchimonji proved that approximately 5% of all smartphones have some form of security tool installed [2]. Y. Xue added that as mainstream security tools are signature-based, they can only effectively detect malware when a comprehensive list of malware signature is provided [3]. Malware authors are then able to develop more complex obfuscation techniques to better evade signature-based detection. This phenomenon was observed in [4], where over 3000 mutated versions of a known ransomware called SLocker were cited.

As security tools on most smartphones today perform signature-based analysis, this work focuses on proposing detailed obfuscation tactics that malware authors might leverage to obstruct or evade analysis. Additionally, recent malware found are not yet updated with evasion techniques yet. This work obfuscates recent malware with potential evasion methods that are proposed in this paper. The results also confirms that recent malware are able to evade signature-based analysis with the obfuscation techniques proposed in this paper. To our knowledge, this is the only paper to-date that has proven the instability of some anti-malware tools (AMTs) detection results. We were able to conclude that current mainstream AMTs are not resilient against the obfuscation method, but merely build up their signature database to detect mutated malware.

2 PREVIOUS WORKS

Works like [5] designed frameworks with various obfuscation techniques to transform the original malware into a different form. However, the obfuscation techniques implemented were trivial transformations including renaming of identifiers, methods and files, introducing junk codes, code reordering and string encoding. Although [5] mentioned more complicated obfuscation techniques like reflection or bytecode encryption, the effectiveness of such techniques is not evaluated in the paper. The framework that is proposed in this work explains more detailed techniques to perform code reordering, for example method overloading, opaque predicate, a try-catch or switch function. Additionally, the previous works were evaluating malware created before 2013. The malware dataset in this work is larger and more updated (e.g. malware from 2013 to 2015). Malware found in 2016 were not used due to the instability of AMTs in detecting them; for a better comparison of their ability to evade detection, malware should be consistently detected by AMTs used. Cani et. al. [6] performed generic programming to mutate malware samples into new possible variants. One critical flaw in these mutated malware is that its malicious behaviour might not be retained after transformation. Due to this, we will avoid using generic programming to perform obfuscation.

3 OBFUSCATION TECHNIQUES

To evaluate the effectiveness of techniques used on malware samples in avoiding detection by AMTs, we developed an automated framework consisting of several obfuscation techniques that could potentially be used by malware authors to obfuscate their malware. Based on the framework, we were able to send large numbers of known malware through these obfuscation techniques to generate a new variants, which were verified to possess the original malicious operations. In the following paragraphs, we discuss in detail the obfuscation techniques that our automated system contains.

CODASPY'18, March 19–21, 2018, Tempe, AZ, USA.

ACM ISBN 978-1-4503-5632-9/18/03.
DOI: http://dx.doi.org/10.1145/3176258.3176942

3.1 Method Overloading

Method overloading uses polymorphism to divert the malware's flow by defining new functions to invoke functions that are originally in the malware. Dummy methods would be cloned with similar method names as the original method calls. For the malware to be executed correctly, the obfuscation method needs to ensure that function invocation is unique. The dummy method distinguishes itself from the original method by modifying the parameters and return type. This obfuscation provides a trivial means to manipulate the call graph of the application to defeat cryptographic hash matching detection.

3.2 Opaque Predicate

Opaque predicate uses conditional expressions to ensure one branch always gets executed. In this obfuscation method, we combined it with junk codes to further conceal the control flow of the malware. This technique inserts an opaque predicate statement between the original source code and sets it to always execute a particular branch. The malware's original codes are added to the branch that gets executed, while junk codes are added to the remaining branches.

3.3 Try-catch

Try-catch is scripted to always execute the "catch" block, by deliberately introducing an exception error such as arithmetic exception in the "try" block. This technique first identifies the smali codes within the method call and then adds an exception statement to the try-catch structure. Lastly, the return statement of the method call is included at the end of the catch block. This technique creates an illusion that the "catch" block would occasionally get executed when the error is invoked. However, the error would always get invoked as the try-catch structure contains an additional statement to deliberately cause the error.

3.4 Switch Statement Obfuscation

Switch statement obfuscation uses the concept of switches to break up the smali codes from a method call and insert them into the different branches of the switch accordingly. The switch structure is modified such that a "goto" instruction is added at the end of each branch, enabling all branches to be executed instead of one. This allows the dynamic program flow to be similar to the original malware, but makes it harder for static analysis.

4 DATASET

This section describes the malware samples that were selected and AMTs that were used for the study. An important criterion of this investigation is that the AMT detection should also be stable enough such that the AMTs detect the malware samples consistently.

4.1 Malware Dataset

Figure 1 provides an overview of the malware families in our malware dataset contributed from the Drebin dataset [7], Contagio Mobile [8] and open source downloads. It is evident that the majority of pre-2013 malware attempted to make money from the victims through sending premium SMS without user control. After 2013, malware samples have shifted from sending premium SMS to spywares, ransomwares and banking trojan. To ensure that the work is applicable with future malwares, we focuse our evaluation on popular malware. Table 1 lists the top three malware types that this paper would be evaluating.

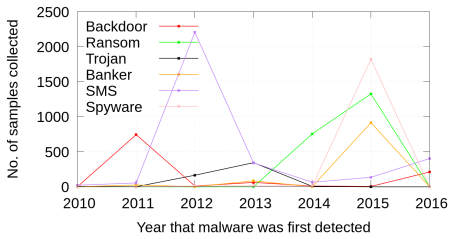

Figure 1: Evolution of malware types

Table 1: Malware dataset used to evaluate efficiency of obfuscation techniques on AMT

Malware Type	Families of Malwares	Samples	Detection Date
Spyware	SmsZombie, Vmvol, Finspy, Mecor	1851	2012–2015
Ransomware	Koler, SimpleLocker, Aples, Fusob, Roop	1560	2014–2015
Banking Trojan	Zitmo, Svpeng, Bankun, BankBot, Slembunk	1021	2011–2015

4.2 Anti-malware Tools used for Analysis

Akin to [9] and [10], our automation tool leverages VirusTotal API [11] to classify the malware samples. Using the 57 AMTs listed on VirusTotal, we evaluate the effectiveness of our proposed transformation techniques. To make the evaluation scalable for a large number of malware samples, we used the command line version of VirusTotal, which might be performing static analysis with a certain degree of signature database.

5 EVALUATION

The evaluation in this research work has two aims. First, we aim to evaluate the ability of a new variant of a known malware, transformed with a composite of obfuscation techniques, to evade detection from AMTs. The second part of the work investigates how detection results for mutated malware change over time. We wish to remark that the AMTs version would be updated across time as well. To the best of our understanding, there have been no citations that provide detection rate changes across time. This section illustrates the instability in detection results for AMTs and proved that these AMTs do not have resilience against our proposed obfuscation methods.

5.1 Effectiveness of Obfuscation

In order to quantitatively measure the ability of AMTs to detect the new malware variants, we create a measure called escape detection rate (EDR) defined by $\text{EDR}(\text{AMT}_i) = \frac{N}{T}$, where N is the number of mutated malware that manage to evade detection by AMT_i and T represents the number of original malware samples detected by AMT_i. The results of the resilience of the 57 AMTs listed in VirusTotal is tabulated in Table 2. The high proportion of AMTs with EDR greater than 0.8 proved that the obfuscation techniques highlighted

Table 2: Distribution of 57 AMTs' EDR.

EDR	0.0-0.2	0.2-0.4	0.4-0.6	0.6-0.8	0.8-1.0
No. of AMTs	7	9	2	6	33

in this work allowed the new variants of the malware to conclusively evade signature-based detection.

5.2 Detection Results Over Time

The next investigation analyses the change of AMT detection results over time. We took 4432 malware samples tabulated in Table 1. The detection of 30 out of 57 AMTs remained unchanged across a time period of nine days. There were 21 AMTs that managed to detect all obfuscated malwares after one version update, when they were initially unable to do so. The remaining six AMTs had unstable detection rate for obfuscated malware. The instability is illustrated in the top four graphs in Figure 2, whereas the bottom four graphs illustrate the results of AMTs that produced stable results. This instability in the detection results of AMTs against mutated malware is beneficial for malware authors.

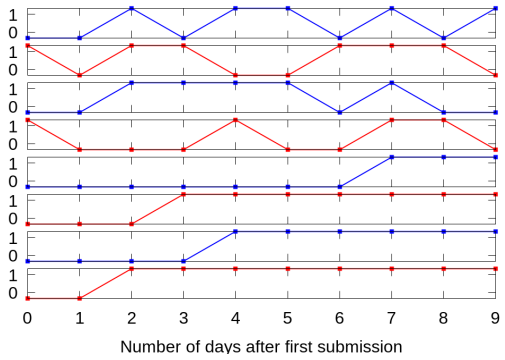

Figure 2: Detection pattern of AMTs across 9 days.

5.3 Resilience of AMTs against Obfuscation Methods

The last part of the investigtion evaluates the resilience of AMTs against the proposed obfuscation techniques. We tested all malware from Table 1 and ensured that the samples are detected as malware by VirusTotal. After obfuscation, the number of malware that are detected has reduced from 4432 to 2274, a 48.69% detection rate decrease. The obfuscated malware were continuously submitted to VirusTotal for nine days. After this period, we can see from Table 3 that the number of malware detected has reduced from 4432 to 3878. This is a 12.5% reduction in detection rate as compared to the original malware samples. It is evident that obfuscated malware were able to initially evade detection. As the malware samples are continuously resubmitted for analysis, the number of obfuscated malware samples that are able to evade AMT detection is reduced.

When the original malware samples were re-obfuscated with the same techniques to produce a different hash value, it resulted in a decrease in detection rate of 48.44% from the original samples. This implies that the AMTs on VirusTotal do not build resilience against the obfuscation technique, but instead use signature-based analysis to detect obfuscated malware. Such a detection scheme would imply that AMTs are constantly playing catch up with the malware authors, who could easily evade signature-based detection by using trivial obfuscation techniques to quickly change the cryptographic hash of a malware and mutate it.

Table 3: Malware dataset used to evaluate efficiency of obfuscation techniques on AMT

Description	SHA(input)	AM detected	Decrease
Submitted after obfuscation	x_1	2274	48.69%
Submitted same variant 9 days later	x_1	3878	12.50%
Submitted after re-obfuscation	x_2	2285	48.44%

6 CONCLUSIONS

This study proved that malware authors can increase a malware's evasion rate by performing obfuscation techniques that were highlighted in this study. In our analysis, we managed to confirm the trend of mutated malware evading detection with a larger and more updated pool of known malware. The novelty in this work identified the instability in detection results for some AMTs. We also highlighted that the AMTs do not build resilience against the technique used to obfuscate the malware, but only update their signature database to be resilient to the specific variant of the malware. The trends highlighted here emphasised the ease of eventually evading current mainstream security tools for a malware author.

REFERENCES

[1] Inc. IDC Research. Smartphone os market share, August 2016.
[2] G. Hatchimonji. Is mobile anti-virus even necessary?, September 2013.
[3] Y. Xue et al. Auditing anti-malware tools by evolving android malware and dynamic loading technique. *IEEE Transactions on Information Forensics and Security*, 12(7):1529–1544, July 2017.
[4] TrendMicro. Ransomware recap: Slocker copycats wannacry, July 2017.
[5] V. Rastogi et al. Droidchameleon: Evaluating android anti-malware against transformation attacks. *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 329–334, 2013.
[6] A. Cani et al. Towards automated malware creation: Code generation and code integration. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 329–334, 2014.
[7] A. Daniel et al. Drebin: Effective and explainable detection of android malware in your pocket. *Symposium on Network and Distributed System Security*, Feb 2014.
[8] M. Parkour. Contagio mobile: Mobile malware minidump, July 2017.
[9] M. Spreitzenbarth. The evil inside a droid - android malware: Past, present and future. *In Proceedings of the BALTIC CONFERENCE Network Security and Forensics*, 2012.
[10] M. Spreitzenbarth et al. Mobile-sandbox: having a deeper look into android applications. *In Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1808–1815, Mar 2013.
[11] VirusTotal. Virustotal public api v2.0, September 2012.
[12] V. Balachandran et al. Function level control flow obfuscation for software security. *Proceedings of the IEEE 8th International Conference on Intelligent and Software Intensive Systems*, pages 133–140, Oct 2014.
[13] V. Balachandran et al. Control flow obfuscation for android applications. *IEEE International Conference on Systems, Man and Cybernetics*, pages 463–469, Dec 2014.
[14] V. Balachandran et al. Control flow obfuscation for android applications. *Computer and Security*, 61 Issue C:72–93, Aug 2016.