3-5-2020

# Black box analysis of android malware detectors

Guruswamy Nellaivadivelu
*San Jose State University*

Fabio Di Troia
*San Jose State University*, fabio.ditroia@sjsu.edu

Mark Stamp
*San Jose State University*, mark.stamp@sjsu.edu

Array 6 (2020) 100022

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/journals/array/2590-0056/open-access-journal

# Black box analysis of android malware detectors

Guruswamy Nellaivadivelu, Fabio Di Troia, Mark Stamp [*]

*Department of Computer Science, San Jose State University, USA*

ARTICLE INFO

*Keywords:*
Malware
Android
VirusTotal
Code obfuscation
Black box analysis

ABSTRACT

If a malware detector relies heavily on a feature that is obfuscated in a given malware sample, then the detector will likely fail to correctly classify the malware. In this research, we obfuscate selected features of known Android malware samples and determine whether these obfuscated samples can still be reliably detected. Using this approach, we discover which features are most significant for various sets of Android malware detectors, in effect, performing a black box analysis of these detectors. We find that there is a surprisingly high degree of variability among the key features used by popular malware detectors.

## 1. Introduction

From a market share of 2.8% in 2009 [26], Android captured about 75% of the market by 2012 [26]. Not surprisingly, the rapid rise of Android has resulted in large quantities of Android malware—in the second quarter of 2016, some 3.5 million examples of Android malware were detected [14]. This vast amount of Android malware has placed a focus on Android security and made it imperative to develop more effective defensive tools for combating such malware. One of the challenges faced in this area is the use of code obfuscation techniques. Obfuscation can alter code to hide its actual purpose, without significantly affecting its function or performance. There are many ways to obfuscate code in an Android environment and several software applications are available that can serve as off-the-shelf code obfuscators [2].

To strengthen malware detectors, there are at least two fundamental issues that need to be addressed [11]. First, we would like to gauge the resilience of malware detectors when faced with obfuscated code. Such analysis will help us to understand the robustness of detectors when dealing with minor variants of known malware samples. The second issue concerns the possibility of uncovering important aspects of a malware detection algorithm. Thus, black box analysis of malware detectors can point towards ways to improve on existing malware detectors.

On the other hand, by studying the behavior of malware detectors, and how they respond to different obfuscation techniques, a malware writer can discover ways to defeat a particular antivirus program. This is extremely useful information for a virus writer, as malware may be made substantially more difficult to detect, yet other necessary aspects of the code can be retained, such as compactness, efficiency, and so on.

In this paper, we analyze the effect of selected code obfuscations on various Android malware detectors. In this way, we can, in effect, reverse engineer these proprietary algorithms. That is, we can determine which features a detector relies on for its capabilities. Through this process, we also gain some insight into the robustness of various detectors when faced with various obfuscations. While such information is obviously useful to a malware writer, as mentioned above, this information is also important to anyone who strives to build a better malware detector.

We emphasize that this paper is based on black box analysis. That is, we assume that we do not know the inner workings of the malware detectors under consideration. In fact, this is the most realistic scenario, as the vast majority of antivirus products are proprietary—analyzing such products would require extensive and costly software reverse engineering, and would be of questionable legality. These issues make it impractical to directly analyze a large number of antivirus products. In contrast, our black box approach is practical, and we can uncover important aspects of many such products.

The remainder of this paper is organized as follows. In Section 2, we discuss selected example of related previous work. Then in Section 3, we briefly consider relevant background topics. Section 4 contains our experimental results, where we analyze the effectiveness of individual obfuscations and combinations of obfuscators with respect to a set of Android malware detectors. Finally, Section 5 contains our conclusions and provides a brief discussion of future work.

## 2. Selected previous work

A considerable amount of research has been done on problems related to code obfuscation as it relates to malware detection. Here, we provide a brief overview of representative examples of such work.

* Corresponding author.
*E-mail address:* mark.stamp@sjsu.edu (M. Stamp).

https://doi.org/10.1016/j.array.2020.100022
Received 3 November 2019; Received in revised form 13 February 2020; Accepted 24 February 2020
Available online 5 March 2020

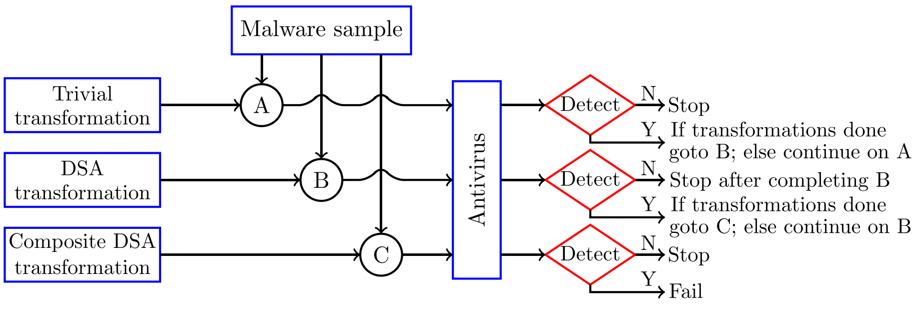

Malware sample
Trivial transformation
DSA transformation
Composite DSA transformation
A
B
C
Antivirus
Detect
N
Stop
Y
If transformations done goto B; else continue on A
Detect
N
Stop after completing B
Y
If transformations done goto C; else continue on B
Detect
N
Stop
Y
Fail

**Fig. 1.** Evaluating malware in [22].

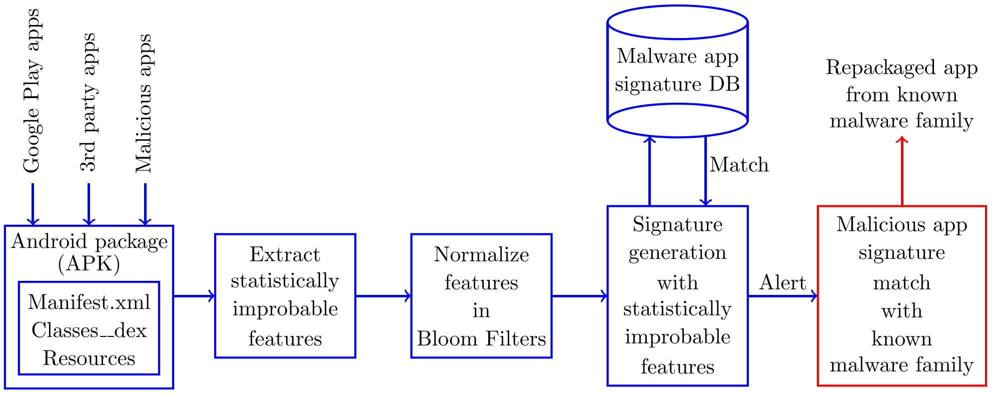

Google Play apps
3rd party apps
Malicious apps
Android package (APK)
Manifest.xml
Classes_dex
Resources
Extract statistically improbable features
Normalize features in Bloom Filters
Signature generation with statistically improbable features
Malware app signature DB
Match
Alert
Malicious app signature match with known malware family
Repackaged app from known malware family

**Fig. 2.** AndroSimilar [15].

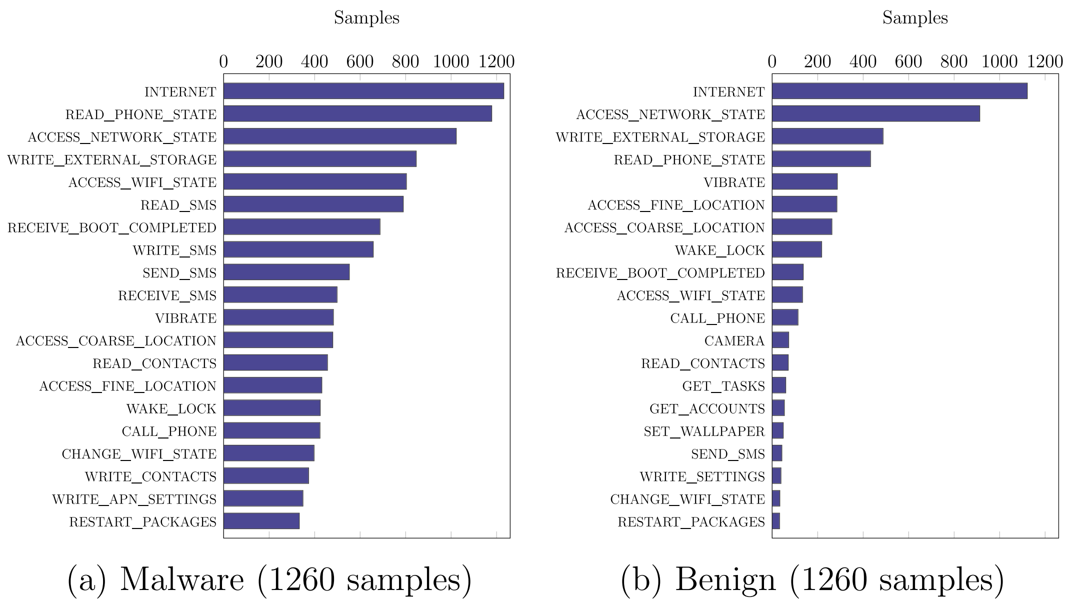


**Fig. 3.** Top 20 permissions in Android [36].

An interesting evaluation of real-world antivirus products (as of 1996) is given by Gordon and Ford in Ref. [16]. In their seminal paper from 2001, Barak et al. [7] formalize the concept of code obfuscation and show that such obfuscation cannot be as strong as cryptography. Nevertheless, malware writers have certainly found obfuscation to be useful for evading detection, which we empirically confirm for Android malware in Section 4 of this paper.

The paper by Christodorescu et al. [11] considers various ways to test and implement program obfuscation. An analysis of their proposed obfuscation methods is also provided, and the authors attempt to quantifying the effectiveness of individual obfuscators.

One powerful evasion technique employed by malware writers is metamorphism, where code is morphed at each infection. In the paper by Rastogi et al. [22], the authors propose and develop a framework that they refer to as DroidChameleon, which provides a way to transform Android applications into different forms, each having the same functionality. This is shown to be an effective means of evading signature detection. 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.

**Table 1**
Android obfuscators.

| Number | Obfuscation |
|---|---|
| 1 | resigned |
| 2 | alignment |
| 3 | rebuild |
| 4 | fields |
| 5 | debug |
| 6 | indirections |
| 7 | renaming |
| 8 | reordering |
| 9 | goto |
| 10 | arithmetic branch |
| 11 | NOP |
| 12 | lib |
| 13 | manifest |
| 14 | reflection |

**Table 2**
Software used.

| Software | Version |
|---|---|
| Java | 1.8.0_45 |
| Python | 2.7.11 |
| apktool | 2.2.1 |

Yan et al. [34] propose a technique in which a virtual machine is used to analyze an application. In most virtual machine based detection architectures, the antivirus program and the malware execute in the same environment. This makes it possible for the malware to detect the virtual machine and take evasive action. The authors of [34] claim that their virtualization environment, which they refer to as DroidScope, can detect malware without being detectable by the malware it is monitoring.

Malware analysis often relies on statistical methods. In the Android context, we can typically decompile an executable file (which has extension apk) to obtain the original source code. The Android applications in our dataset are written in Java, and hence for these applications it is generally easy to reverse engineer the apk file to recover the source code. This opens the door to many types of statistical analysis. For example, Faruki et al. [15] propose a technique that they refer to as AndroSimilar, which decompiles an apk file and extracts features based on the source code, as outlined in Fig. 2. These features are fed into an algorithm that automatically generates a signature for a given malware sample.

Unfortunately, the ability to easily decompile code also makes it easy

**Table 3**
Top 11 malware detectors in VirusTotal.

| Company | Product |
|---|---|
| ALWIL | Avast-Mobile |
| Avira | Avira |
| Cyren | Cyren |
| Eset Software | ESET-NOD32 |
| Ikarus Software | Ikarus |
| K7 Computing | K7GW |
| Kaspersky Lab | Kaspersky |
| Intel Security | McAfee |
| Symantec | Symantec Mobile Insight |
| Tencent | Tencent |
| Zoner Software | Zoner Antivirus |

**Table 4**
Tested obfuscators.

| Number | Obfuscator | Encoding |
|---|---|---|
| 1 | rebuild | 100000 |
| 2 | indirections | 010000 |
| 3 | renaming | 001000 |
| 4 | reordering | 000100 |
| 5 | goto | 000010 |
| 6 | manifest | 000001 |

**Table 5**
Tested benign applications.

| | | |
|---|---|---|
| Google Gmail | Google Maps | Facebook Messenger |
| Facebook | Google | Google Text-to-Speech |
| Instagram | YouTube | Samsung Push Service |
| Google Chrome | Google Play Games | Google TalkBack |
| Google Play Music | Google Street View | Google Play Movies |
| Google Drive | Hangouts | WhatsApp Messenger |
| Android WebView | Google Photos | Google Play Newsstand |
| Skype | Slack | – |

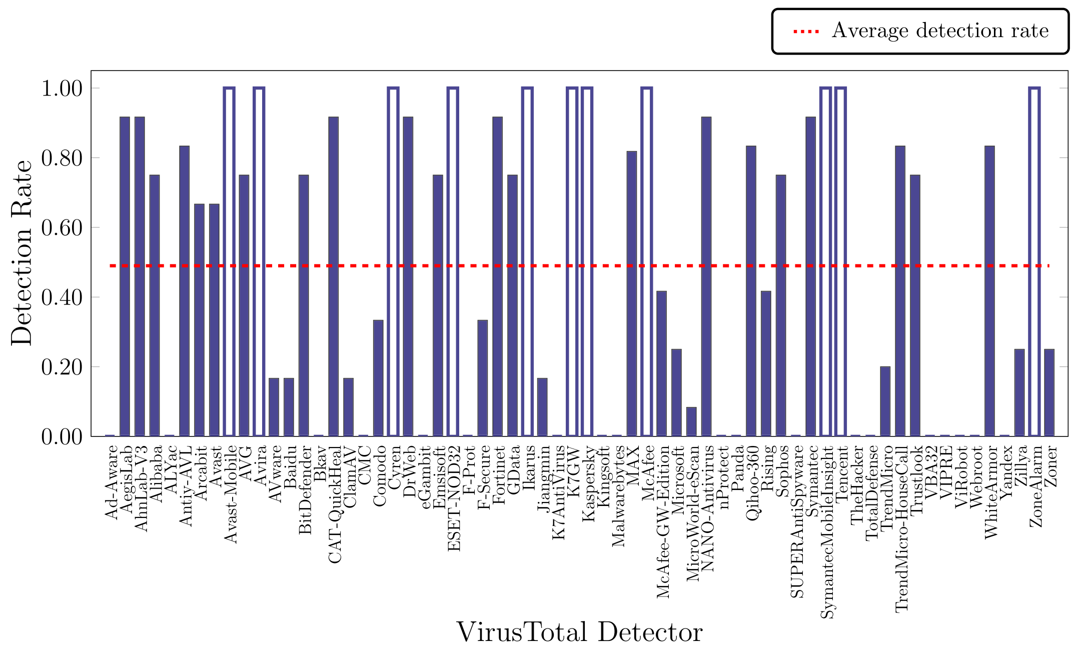


**Fig. 4.** VirusTotal detector results for unobfuscated malware samples.

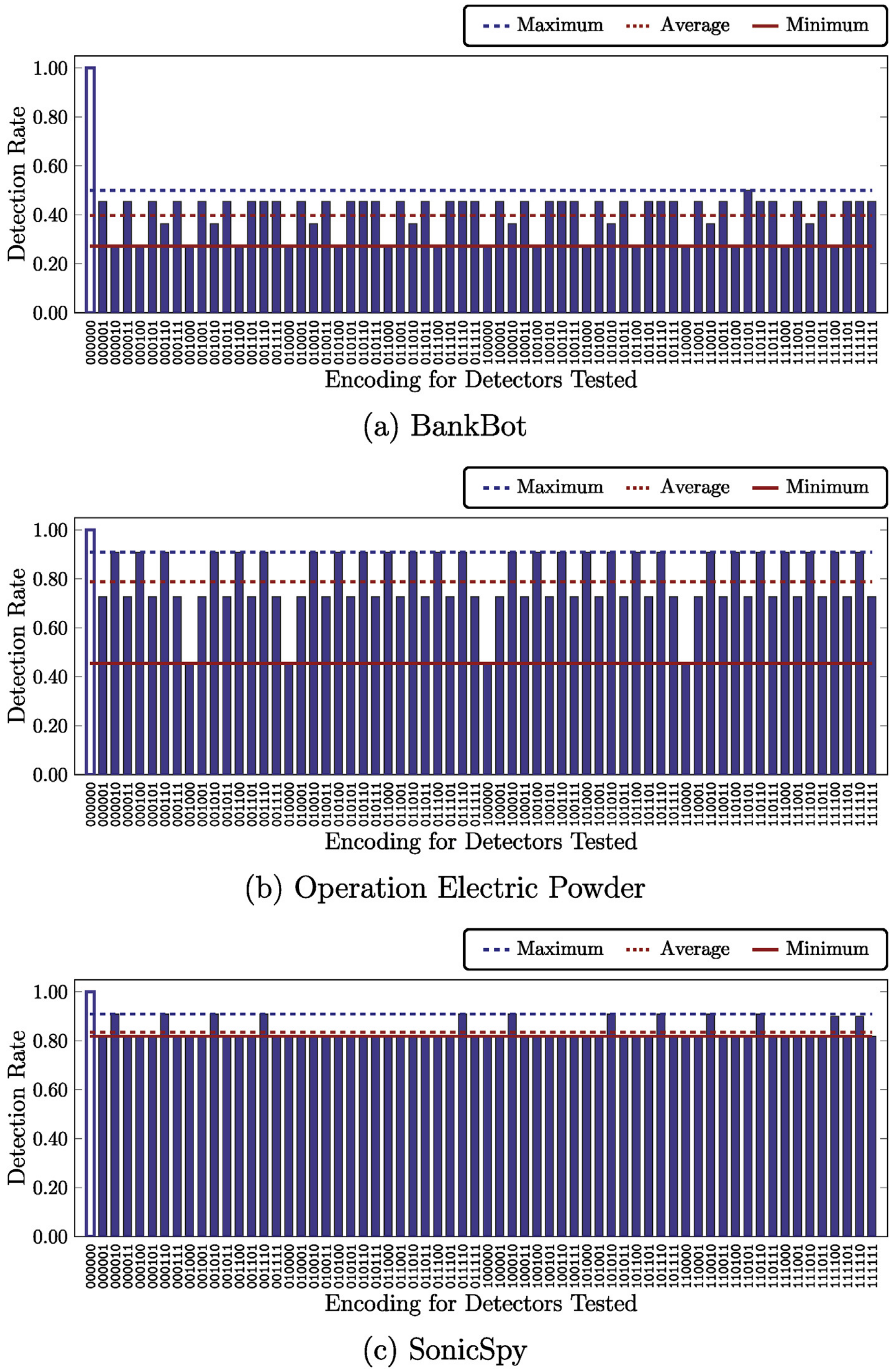


**Fig. 5.** Obfuscation results for three malware families (based on top 11 detectors).

for Android malware writers to repackage existing applications, while inserting unwanted malicious behavior. Such Trojans appear to comprise the vast majority of Android malware today.

As far as the authors are aware, the most similar work to that presented here is [18]. In this previous work, black box analysis techniques are applied to the Drebin Android malware detector [3]. The research in Ref. [18] highlights relevant local features, and utilizes support vector machines (SVM). This differs substantially from the present paper, as [18] is focused on attacks that target the machine learning models, rather than specific detectors. In addition, the Drebin detector is described in detail in Ref. [3], so Drebin is not a black box in the same sense as the VirusTotal [32] detectors that we consider in this paper.

## 3. Background

In this section, we turn our attention to background topics that are relevant for the experimental results that we present in Section 4. Specifically, we focus on Android malware and the various obfuscations that we consider in our experiments.

### 3.1. Android and malware

From Fig. 3 we see that there are significant differences between the set of permissions typically requested by benign and malicious Android applications [36]. In the research literature, this simple observation often

**Table 6**
Malware detection results.

| Malware | Minimum | Maximum | Average |
|---|---|---|---|
| BankBot | 0.2727 | 0.5000 | 0.3975 |
| CopyCat | 0.8000 | 0.9000 | 0.8192 |
| Godless | 0.4545 | 0.5455 | 0.4993 |
| Judy | 0.3636 | 0.5455 | 0.4450 |
| Mazar | 0.4286 | 0.5455 | 0.5429 |
| Operation Electric Powder | 0.4545 | 0.9091 | 0.7879 |
| PluginPhantom | 0.8182 | 0.8182 | 0.8182 |
| SecureUpdate | 0.4545 | 0.6000 | 0.4828 |
| SonicSpy | 0.8182 | 0.9091 | 0.8352 |
| TubeMate | 0.3000 | 0.4545 | 0.3919 |
| WireX | 0.5455 | 0.5556 | 0.5456 |
| Ztorg | 0.4545 | 0.5455 | 0.4949 |

forms the basis for effective Android malware detection results [17].

It is worth noting that there are practical limitations when performing malware detection on an Android device. In a mobile environment, processing power, memory, and, especially, battery usage are significant constraints [4]. However, offline malware detection is also possible—in particular for Android applications that are available online—in which case these issues are much less of a concern. In this research, we are assuming this latter scenario, and hence Android device limitations are not considered to be a constraint on the analysis that we perform.

From a high level perspective, malware analysis can be based on static or dynamic features. Static features can include bytecodes, or any other features that can be extracted without code execution or emulation. Dynamic features often include behaviors that are exhibited when the code executes. Generally, static analysis is more efficient, but dynamic analysis can be more revealing, since many obfuscation techniques are rendered moot once the code executes. From our experimental results, it will become clear that the Android malware detectors that we analyze are based primarily—if not exclusively—on static features.

### 3.2. *Android obfuscators*

In this section, we discuss the various obfuscators that we consider in the experiments presented in Section 4. By systematically obfuscating different aspects of the code, we can gain insight into which features contribute most to a given Android malware detector. This information can, in turn, be used to determine the most effective ways to make malware detectors more effective and more robust.

For this project, we developed a modified version of Another Android Malware Obfuscator (AAMO) [21], that we refer to as Modified AAMO, or simply MAAMO. Our MAAMO tool is available to any researchers who would like to conduct experiment similar to those discussed in this paper.

MAAMO implements a wide variety of obfuscators for Android code. The obfuscators available in MAAMO can be used independently or in any combination. To apply the selected obfuscators, we first decompile an Android application, then we perform the desired obfuscations, and finally, we recompile the code—the resulting application will have the same functionality as the original.

In MAAMO, there are 14 obfuscators available. These obfuscators are listed in Table 1, and each is discussed briefly below.

#### 3.2.1. *Resigned*

The resigned obfuscator resigns the apk file prior to recompilation. Although this obfuscator has essentially no effect on the code itself, it could serve to defeat a malware detector that relies on a specific signature that was applied to a known malware sample. In addition, if an application is expected to be signed, this obfuscation will serve to make it less obvious that the code has been modified.

#### 3.2.2. *Alignment*

The alignment obfuscator makes use of the zipalign utility in Android. This utility applies an optimization technique to apk files whereby all uncompressed data starts at a particular alignment relative to the beginning of the file. The alignment obfuscator changes this alignment before recompiling the apk file, which could affect any detectors that rely on a specific alignment.

#### 3.2.3. *Rebuild*

The rebuild obfuscator rebuilds the application file without performing any other changes. The unpacking and repackaging of the apk file affects the timestamp and related metadata that might help to identify a specific application.

#### 3.2.4. *Fields*

The fields obfuscator renames fields that are used in the application. The application is analyzed to locate the fields that appear in the source code and these fields are renamed.

#### 3.2.5. *Debug*

The debug obfuscator removes all debugging related information from the application, with this operation is performed throughout the source code. Without the debug information, the apk file will typically becomes slightly smaller in size, and it may differ in other minor aspects.

#### 3.2.6. *Indirections*

Call indirections is an advanced obfuscation technique in which various function calls are directed through different values. This obfuscator has a variety of effects, including changing the register count, changing method calls, and also redirecting all calls to methods. For typical code, this obfuscation will heavily alter control flow information. This would likely have a significant effect on many types of dynamic analysis, and hence should be a powerful obfuscation for detectors that rely on dynamic information.

#### 3.2.7. *Renaming*

The renaming obfuscation renames all variables in the source code. Note that this is a far more extensive renaming than occurs with the fields obuscation, as all variable names can be affected. Renaming could be expected to alter signatures and adversely affect other pattern matching techniques that rely on names of variables and functions.

#### 3.2.8. *Reordering*

The use of reordering changes the order of the code in the application. This obfuscator changes the location of certain parts of the code and adjusts the control flow accordingly, so that the code executes in the proper order. Such reordering can make it possible to evade signature detection, since signature detection typically depends on the order of instructions.

#### 3.2.9. *Goto*

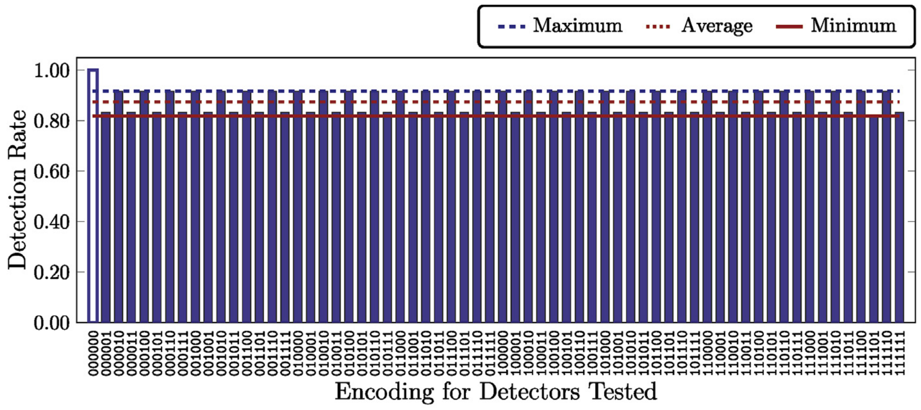
The goto obfuscation changes the control flow by inserting forward and backward jumps into the code. Such jumps can drastically alter program flow and should have a negative impact on any malware detection technique that relies heavily on control flow information. Such information is often used in dynamic analysis.

#### 3.2.10. *Arithmetic branch*

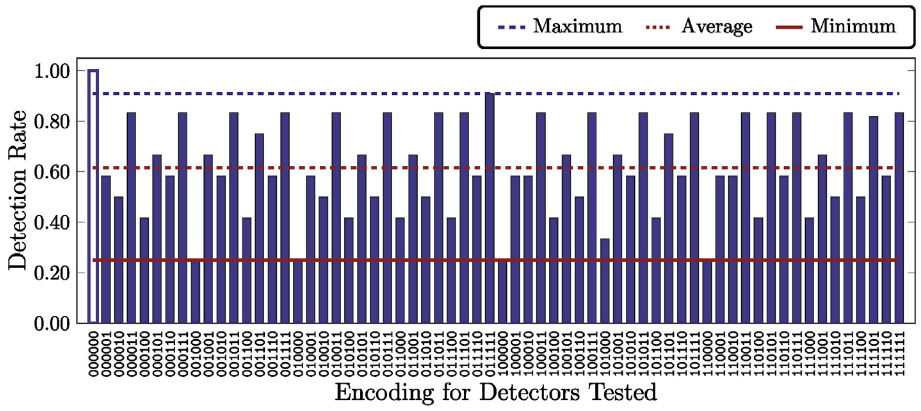
The arithmetic branch obfuscator inserts a branch condition, where only one branch can actually execute. This can greatly complicate analysis that relies on control flow, and it has the effect of inserting dead code, which can negatively impact static analysis. As with all other obfuscations considered here, this operation leaves the function of the original code unchanged.

#### 3.2.11. *NOP*

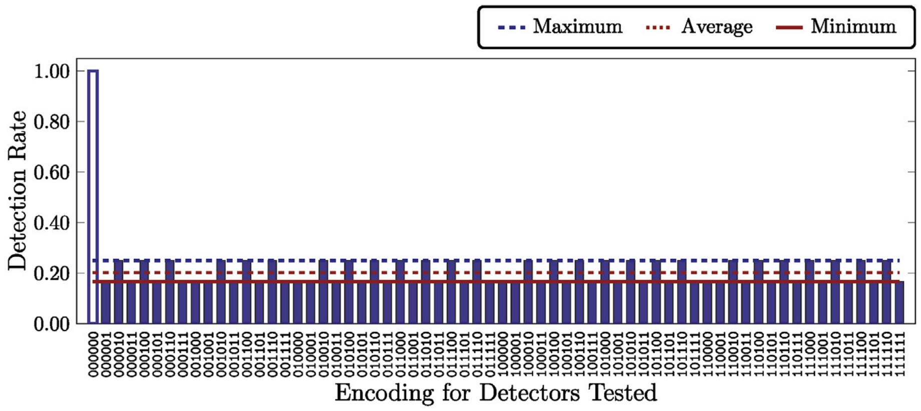
Inserting a do-nothing or no-operation (NOP) instruction is the simplest means available for breaking signatures. In our MAAMO



(a) Kaspersky



(b) McAfee



(c) Tencent

**Fig. 6.** Detector-based results for three popular antivirus (based on 12 malware samples).

implementation, such instructions are inserted at random throughout the code.

### 3.2.12. *Lib*

In the lib obfuscation, MD5 hashing is used to rename the file and associated paths. A proxy method is created, which serves to deal with the hashed values.

### 3.2.13. *Manifest*

The AndroidManifest.xml file is modified by the manifest obfuscator. The manifest file contains a variety of important information—from a malware detection point of view, most significantly, it deals with permissions. Among other operations, this obfuscator encrypts the values of resources and also replaces characters in user-defined identifiers.

### 3.2.14. *Reflection*

The reflection obfuscator takes advantage of the Android dynamic code loading API. Specifically, all static method calls are converted into reflection calls and the reflect method is invoked on a string that contains the target method. It is not clear that this would have a large impact on most malware detection techniques.

**Table 7**
Summary of results for top 11 detectors.

| Detector | Minimum | Maximum | Average |
|---|---|---|---|
| Avast-Mobile | 0.5833 | 0.5833 | 0.5833 |
| Avira | 0.2500 | 0.3636 | 0.3280 |
| Cyren | 0.2500 | 0.3333 | 0.2861 |
| ESET-NOD32 | 1.0000 | 1.0000 | 1.0000 |
| Ikarus | 0.9167 | 1.0000 | 0.9947 |
| K7GW | 0.1667 | 0.3333 | 0.2169 |
| Kaspersky | 0.8182 | 0.9167 | 0.8741 |
| McAfee | 0.2500 | 0.9091 | 0.6160 |
| Symantec Mobile Insight | 0.5000 | 0.5455 | 0.5007 |
| Tencent | 0.1667 | 0.2500 | 0.2024 |
| ZoneAlarm | 0.8182 | 0.9167 | 0.8741 |

## 4. Experiments and results

In this section, we briefly discuss our experimental setup, and we provide details on the data and malware detectors used in the experiments. Then we present our experimental results in some detail.

### 4.1. Environment and setup

In our experiments, each apk file is decompiled into its source code which is then obfuscated before being repackaged. To decompile the apk, we use apktool [33]. As discussed in Section 3.2, we use our MAAMO tool to perform all obfuscations. A list of the various software used to conduct the experiments reported in this research appears in Table 2.

### 4.2. Malware samples and preliminaries

To evaluate and analyze malware detectors, we performed experiments on known Android malware. As discussed above, obfuscators were applied to perform a black box analysis of malware detectors, and to obtain an understanding of the state of the art in Android malware detection in general.

The Contagio dataset was used as our source of Android malware samples [13]. This dataset has been used in many recent research studies, including, for example, [17].

We use the VirusTotal [31] website as our source for malware detection results. At the time of our experiments, VirusTotal included 64 applicable malware detectors that we tested on each sample. These detectors include products from virtually all of the leading anti-virus companies, including Kaspersky, McAfee, Microsoft, Sophos, Symantec, and Trend Micro. In a few sporadic instances, VirusTotal did not return a result for a particular detector.

The VirusTotal website uploads a malware file to its database and then performs a scan using the various malware detectors available at the website. Each uploaded file is hashed and stored in the database to reduce duplicate effort and minimize scan times. We used VirusTotal to scan each malware sample, and each obfuscated variant of a given sample.

The experimental results provided below are all based on the following Android malware families.

**BankBot** is a credential stealing malware that mimics banking sites to trick users into revealing confidential information [23]. This malware has repeatedly found its way onto the Google Play Store [20].

**CopyCat** is a highly advanced form of adware that is estimated to have infected some 14,000,000 devices and to have generated an impressive amount of fraudulent advertising revenue for its developers [9].

**Godless** is also a very sophisticated malware that is the Android equivalent of an exploit kit for a PC. This malware primarily serves ads, but it is capable of significantly more [8].

**Judy** is adware that is often claimed to be the most widespread malware that has yet appeared on Google Play [10].

**Mazar** pretends to be a generic "MMS Messaging" application, but is actually a botnet application that can erase data from an infected device, and perform other malicious activities [27].

**Operation Electric Powder** is spyware that is distributed as a the seemingly harmless application pokemon.apk. The name of the malware derives from the fact that it plays a role in an attack on the Israel Electric Company (IEC) [12].

**PluginPhantom** is an information stealing Trojan that uses multiple plugins, making static detection difficult [35].

**SecureUpdate** acts as a downloader, that serves primarily to fetch additional malware-. Some versions of SecureUpdate include a credential stealing feature [6].

**SonicSpy** is spyware that has infected a large number of applications [24].

**TubeMate** is a legitimate applications, but some versions of it have been Trojanized and act as aggressive forms of adware [30].

**WireX** is a botnet that is capable of launching a DDoS attack. This malware was recently found to have infected hundreds of apps on the Google Play Store [1].

**Ztorg** can cause infected phones to send premium rate SMS messages [28,29],

We tested each of the 64 malware detectors in VirusTotal on examples of the 12 malware types discussed above. Without MAAMO obfuscation, we obtained the results in Fig. 4, where the hollow bars represent the detectors that flagged all 12 samples as malware. All of the remaining detectors failed on at least one of the samples—with 19 of the 64 failing to flag any of the malware samples as malicious.

Again, of the 64 relevant detectors that were available in VirusTotal at the time of our testing, only the 11 corresponding to the hollow bars in Fig. 4 were able to correctly identify all 12 of the (unobfuscated) malware samples discussed above. These 11 detectors are listed in Table 3.

For the experiments discussed below, we focus our attention on the 11 detectors that appear in Table 3. With respect to our malware sample set, these are the most effective detectors available, and hence should provide the greatest challenge for the various obfuscation techniques considered.

Intuitively, it would seem that the 6 obfuscators listed in Table 4 are likely to be the most effective at evading malware detection. And, based on preliminary experiments [19], we found this to be case when considered in isolation. Hence, it would appear that the features affected by these obfuscators figure more prominently in the malware detectors considered, as compared to the features affected by the remaining obfuscators. Therefore, in the experiments discussed below, we focus on these 6 obfuscators. In Section 4.4 we analyze the effect of each of the 64 different combinations of these obfuscators on the detectability of each malware sample. The goal is to determine which of features (or combinations of features) are most influential with respect to Android malware detectors. Then in Section 4.5 we consider these results per detector, instead of per malware. These latter results give us an inside look at each of the Android malware detectors listed in Table 3.

### 4.3. False positives

Before giving our main experimental results, we briefly consider false positives. Antivirus products are generally tuned so as to generate an extremely low false positive rate [5], in spite of the fact that this will tend to result in a significantly higher false negative rate. While this may seem counterintuitive, false positives can be disastrous for any antivirus product, since customers will lose confidence in a product that flags a known clean application as malware. Another concern is that a developer whose application is incorrectly flagged as malicious might suffer a major financial loss, and thus any such developer has an incentive to publicize such failings. Such cases would serve to damage the reputation of the offending antivirus product.

Because we expect the false positive rate to be negligible, here we focus entirely on false negatives in our experiments presented in Sections 4.4 and 4.5, below. To verify that this is indeed a reasonable approach, we first tested all of the 23 well-known benign applications listed in Table 5 against all of the malware detectors in VirusTotal. We found that all of these benign Android applications were correctly classified as benign by every malware detector in VirusTotal.

### *4.4. Obfuscator-based results*

In this section, we present results for the set of 12 malware samples discussed above. In each case, we test every combination of the 6 obfuscators listed in Table 4. For selected cases, we give the results in the form of bar graphs, and for all cases we summarize the results in tabular form. In the graphs below, the obfuscations employed are encoded as a binary 6-tuple, where a 1 indicates that the corresponding obfuscation is used, and a 0 indicates that the obfuscation is not used. For example, 000001 indicates that only the manifest obfuscation was employed, while 010100 indicates that both the indirections and reordering obfuscations were used.

Fig. 5 (a) gives results for all possible subsets of the obfuscators listed in Table 4 for the BankBot malware. In each case, the obfuscated BankBot samples were tested against each of the 11 detectors listed in Table 3. In Fig. 5(b) and (c), we give the analogous results for Operation Electric Powder and SonicSpy, respectively. Note that in each of these bar graphs, the hollow bar on the left-hand side (the bar labeled "000000") gives the detection rate in the case where no obfuscation is applied. Since we are only using detectors that had a 100% detection rate on our malware samples, this bar is at 1.0 in each case. The other bars represent the detection rates when the specified obfuscations are applied, with the maximum, minimum, and average cases indicated. Each of the maximum, minimum, and average is computed over the 63 obfuscations "000001" through "111111." Note that the unobuscated case of "000000" is not included in the average case calculation.

From the BankBot results in Fig. 5(a), we see that the when any obfuscation is applied, BankBot is only detected by half or fewer of the detectors. For example, if we apply only the rebuild obfuscation (denoted as "100000"), then only about 27% of the detectors correctly classify this obfuscated form of BankBot as malware. Over all possible obfuscations, we see that on average, only about 40% of the detectors were able to recognize the obfuscated version of BankBot.

In Fig. 5(c), we see that obfuscations have a minimal impact on detection results for SonicSpy. This provides a sharp contrast to the BankBot results in Fig. 5(a).

The results in Fig. 5(b) illustrate yet another distinct case. Specifically, for the malware known as Operation Electric Powder, most obfuscations have little effect, but selected obfuscations (e.g., "001000") are able to significantly reduce the detection rate.

The detection rates for each of the 12 malware samples—over all 64 combinations of the obfuscations under consideration—are given in Table 6. Again, these results were obtained by restricting our attention to the top 11 detectors listed in Table 3.

### *4.5. Detector-based results*

Fig. 6(a) gives results for all possible subsets of the 6 obfuscators under consideration, with respect to Kaspersky antivirus. The specified obfuscations were applied the same set of malware as in the previous section, and these obfuscated samples were then tested against the Kaspersky antivirus. Fig. 6(b) and (c) give the analogous results for McAfee and Tencent antivirus, respectively.

From the results in Fig. 6, we see that Kaspersky is the most robust of these three detectors, in the sense that obfuscations have relatively little effect on its accuracy. In contrast, Tencent in Fig. 6(c) is the most fragile, as any obfuscation decreases the detection rate to a very small percentage of the unobfuscated case. Finally, from Fig. 6(b), we see that McAfee is somewhat in between these two extremes—some obfuscations are very effective, while other obfuscations cause only a minor reduction in accuracy.

In Table 7, we summarize the results for all 11 of the detectors considered in this section. From these results, we see that the effect of the obfuscations under consideration varies widely over this set of malware detectors.

### **4.6. Discussion**

Not surprisingly, the results in this section clearly show that obfuscation can be highly effective. We find that to a large degree, the optimal obfuscation depends on the specific malware detector, as well as the actual malware under consideration. This indicates that the malware detectors considered here are fairly diverse, in the sense that they apparently rely on different features or combinations of features for detection and, furthermore, the precise set of features appears to vary somewhat for different malware samples.

Remarkably, one of the tested virus detectors (ESET-NOD32) was able to classify all obfuscated samples as malware, while two additional detectors (Ikarus and Kaspersky) also yielded impressively high detection results. At the other extreme, 19 of the 64 detectors available in VirusTotal failed to detect any of the original (unobfuscated) malware samples in our tests. These results indicate that there is an extremely wide range of capability among the detectors in VirusTotal.

Perhaps more surprising than the range in detector capabilities is the range in the malware samples themselves. Some samples (e.g., BankBot and TubeMate) seem to be extremely easy to obfuscate, in the sense that virtually any obfuscation has a large impact. On the other hand, we found that some samples (e.g., CopyCat and SonicSpy) were not effectively obfuscated with any combination of the obfuscators under consideration. And, for at least one application (Operation Electric Powder), selected obfuscations were effective, while most obfuscations had only a limited effect.

## **5. Conclusion and future work**

Our results clearly show that fairly straightforward obfuscation techniques can be highly effective against a collection of strong malware detectors available on VirusTotal. Our results also indicate that there is a high degree of diversity among these malware detectors, in the sense that no single feature—or even a combination of features—seems to dominate the overall detection results. In addition, there appears to be some diversity even within a single malware detector, in the sense that different malware samples often yield different obfuscation profiles with respect to a given antivirus.

There are several possible avenues for related future work. First, it would be useful to conduct large-scale experiments. The problem here lies primarily with the VirusTotal API, which is slow and difficult to use, making it challenging to accumulate large numbers of useful results [32], even with access to a higher data rate than is typically publicly available.

We performed exhaustive experiments on 6 of the 14 obfuscators listed in Table 1. The 6 we chose appear to be the most effective individually, but it is possible that in combination, some of the other 8 obfuscators might yield good results.

Another interesting area of related research would consist of carefully analyzing the strengths and weaknesses of various machine learning based malware detectors when facing obfuscations of the type considered in this paper. Malware detectors based on hidden Markov models (HMM), support vector machines (SVM), deep learning, and a wide variety of other machine learning techniques have recently shown great promise [25]. It would be useful to quantify the robustness of such techniques, while comparing machine learning based results to existing antivirus products. Our MAAMO tool, along with the results presented in this paper, could form the basis for such research.

### **Declaration of competing interest**

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## CRediT authorship contribution statement

**Guruswamy Nellaivadivelu:** Investigation. **Fabio Di Troia:** Investigation. **Mark Stamp:** Investigation.

## References

[1] InfoSec Akamai. The WireX botnet: an example of cross-organizational cooperation. 2019. August 2017. Accessed 28-6 https://blogs.akamai.com/2017/08/the-wirex-botnet-an-example-of-cross-organizational-cooperation.html.

[2] Apvrille A, Nigam R. Obfuscation in Android malware, and how to fight back. Virus Bull 2014:1–10.

[3] Arp D, Spreitzenbarth M, Gascon H, Drebin K Rieck. Effective and explainable detection of Android malware in your pocket. 2014. https://www.sec.cs.tu-bs.de/pubs/2014-ndss.pdf.

[4] Arshad S, Shah MA, Khan A, Ahmed M. Android malware detection & protection: a survey. Int J Adv Comput Sci Appl 2016;7(2):463–75.

[5] Comparatives AV. Details of false alarms. SeptemberAccessed 2019-06-28, https://www.av-comparatives.org/wp-content/uploads/2017/10/avc_fps_201709_en.pdf; 2017.

[6] Bar T, Lancaster T. Targeted attacks in the Middle East using KASPERAGENT and MICROPSIA. April 2017. Accessed 2019-06-28, https://researchcenter.paloaltonetworks.com/2017/04/unit42-targeted-attacks-middle-east-using-kasperagent-micropsia/.

[7] Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K. On the (im)possibility of obfuscating programs. In: Annual international cryptology conference. Springer; 2001. p. 1–18.

[8] Barth B. Blasphemy! Godless malware preys on nearly 90 percent of Android devices. June. SC Media; 2016. Accessed 2019-06-28, https://www.scmagazine.com/blasphemy-godless-malware-preys-on-nearly-90-percent-of-android-devices/article/529594/.

[9] Check Point Mobile Research Team. How the CopyCat malware infected Android devices around the world. July 2017. Accessed 2019-06-28, https://blog.checkpoint.com/2017/07/06/how-the-copycat-malware-infected-android-devices-around-the-world/.

[10] Check Point Mobile Research Team. The Judy malware: possibly the largest malware campaign found on Google Play. May 2017. Accessed 2019-06-28, https://blog.checkpoint.com/2017/05/25/judy-malware-possibly-largest-malware-campaign-found-google-play/.

[11] Christodorescu M, Jha S. Testing malware detectors. In: ACM SIGSOFT international symposium on Software testing and analysis; 2004. p. 34–44. New York, NY, USA.

[12] ClearSky Research Team. Operation electric powder — who is targeting Israel electric Company?. Accessed 2019-06-28, http://www.clearskysec.com/iec/; March 2017.

[13] Contagio mobile malware mini dump. Accessed 2016-12-15, http://contagiominidump.blogspot.com/.

[14] Emm D, Unuchek R, Garnaeva M, Ivanov A, Makrushin D, Sinitsyn F. IT threat evolution in Q2 2016. Securelist. Accessed 2016-10-10, https://securelist.com/analysis/quarterly-malware-reports/75640/it-threat-evolution-in-q2-2016-statistics/; 2016.

[15] Faruki P, Ganmoor V, Laxmi V, Gaur MS, AndroSimilar A Bharmal. Robust statistical feature signature for Android malware detection. In: Proceedings of the 6th international conference on security of information and networks. ACM; 2013. p. 152–9.

[16] Gordon S, Ford R. Real world anti-virus product reviews and evaluations — the current state of affairs. In: Proceedings of the 1996 national information systems security conference; 1996.

[17] Kapratwar A, Troia FD, Stamp M. Static and dynamic analysis of Android malware. In: Proceedings of the 3rd international conference on information systems security and privacy. ScitePress; 2017. p. 653–62.

[18] Melis M, Maiorca D, Biggio B, Giacinto G, Roli F. Explaining black-box Android malware detection. https://arxiv.org/pdf/1803.03544.pdf.

[19] Nellaivadivelu G. Black box analysis of Android malware detectors. San Jose State University; 2017. Master's Projects 545, https://scholarworks.sjsu.edu/etd_projects/545.

[20] Palmer D. BankBot android malware sneaks into the Google play Store — for the third time. ZDNet. Accessed 2019-06-28, http://www.zdnet.com/article/bankbot-android-malware-sneaks-into-the-google-play-store-for-the-third-time/; November 2017.

[21] Preda MD, Maggi F. Testing Android malware detectors against code obfuscation: a systematization of knowledge and unified methodology. J Comput Virol Hacking Tech 2016:1–24.

[22] Rastogi V, Chen Y, Jiang X. DroidChameleon: evaluating Android anti-malware against transformation attacks. In: Proceedings of the 8th ACM SIGSAC symposium on information, computer and communications security. ACM; 2013. p. 329–34.

[23] Seals T. BankBot android trojan re-emerges globally. *InfoSecurity magazine*. 2017. Accessed 2019-06-28, https://www.infosecurity-magazine.com/news/bankbot-android-trojan-reemerges/.

[24] Spring T. Apps infected with SonicSpy spyware removed from Google play. Threatpost. Accessed 2019-06-28, https://threatpost.com/apps-infected-with-sonicspy-spyware-removed-from-google-play/127406/; August 2017.

[25] Stamp M. Introduction to machine learning with applications in information security. Boca Raton: Chapman and Hall/CRC; 2017.

[26] Statista. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016. Accessed 2016-12-01, https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[27] Mazar Symantec. BOT malware invades and erases Android devices. Accessed 2019-06-28, https://us.norton.com/internetsecurity-emerging-threats-mazar-bot-malware-invades-and-erases-android-devices.html; 2017.

[28] Ztorg R Unuchek. From rooting to SMS. Securelist. Accessed 2019-06-28, https://securelist.com/ztorg-from-rooting-to-sms/78775/; June 2017.

[29] Ztorg R Unuchek. Money for infecting your smartphone. Securelist. May 2017. Accessed 2019-06-28, https://securelist.com/ztorg-money-for-infecting-your-smartphone/78325/.

[30] Virus & malware removal. TubeMate "virus" android removal. Accessed 2019-06-28, https://howtoremove.guide/tubemate-virus-android-remove/; 2014.

[31] Virus total. Accessed 2017-04-15, https://www.virustotal.com.

[32] VirusTotal. Virustotal public api v2.0. Accessed 2019-06-28, https://www.virustotal.com/en/documentation/public-api/.

[33] Wisniewski R, Tumbleson C, Apktool. Accessed 2016-10-26, https://ibotpeaches.github.io/Apktool/.

[34] Yan L-K, DroidScope H Yin. Seamlessly reconstructing the os and dalvik semantic views for dynamic Android malware analysis. In: USENIX security symposium; 2012. p. 569–84.

[35] Zheng C, Luo T. PluginPhantom: new android trojan abuses "DroidPlugin" framework. Accessed 2019-06-28, https://researchcenter.paloaltonetworks.com/2016/11/unit42-pluginphantom-new-android-trojan-abuses-droidplugin-framework/; November 2016.

[36] Zhou Y, Jiang X. Dissecting android malware: characterization and evolution. In: 2012 IEEE symposium on security and privacy; May 2012. p. 95–109.