A Large-Scale Empirical Study on the Effects of Code Obfuscations on Android Apps and Anti-Malware Products

Mahmoud Hammad, Joshua Garcia, and Sam Malek
Department of Informatics
University of California, Irvine
Irvine, California, USA
{hammadm, joshug4, malek}@uci.edu

ABSTRACT

The Android platform has been the dominant mobile platform in recent years resulting in millions of apps and security threats against those apps. Anti-malware products aim to protect smartphone users from these threats, especially from malicious apps. However, malware authors use code obfuscation on their apps to evade detection by anti-malware products. To assess the effects of code obfuscation on Android apps and anti-malware products, we have conducted a large-scale empirical study that evaluates the effectiveness of the top anti-malware products against various obfuscation tools and strategies. To that end, we have obfuscated 3,000 benign apps and 3,000 malicious apps and generated 73,362 obfuscated apps using 29 obfuscation strategies from 7 open-source, academic, and commercial obfuscation tools. The findings of our study indicate that (1) code obfuscation significantly impacts Android anti-malware products; (2) the majority of anti-malware products are severely impacted by even trivial obfuscations; (3) in general, combined obfuscation strategies do not successfully evade anti-malware products more than individual strategies; (4) the detection of anti-malware products depend not only on the applied obfuscation strategy but also on the leveraged obfuscation tool; (5) anti-malware products are slow to adopt signatures of malicious apps; and (6) code obfuscation often results in changes to an app's semantic behaviors.

ACM Reference Format:
Mahmoud Hammad, Joshua Garcia, and Sam Malek. 2018. A Large-Scale Empirical Study on the Effects of Code Obfuscations on Android Apps and Anti-Malware Products. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3180155.3180228

1 INTRODUCTION

Android is the dominant mobile platform holding 85% of the smartphone OS market share [11]. At the same time, the number and sophistication of malicious Android apps are increasing. For instance, McAfee Labs crawled several app stores over six months in 2016 and detected more than 9 million malicious apps [10]. As another example, Kaspersky discovered more than 4 million new malware in 2016 [9].

Many reasons contribute to this meteoric rise of malware apps including: (1) the relative ease of creating a piggybacked app [35–38, 48], i.e., a mutated version of a legitimate app injected with either malicious code or embedded advertisements; and (2) the prevalence of alternative Android app stores (i.e., app stores other than the official Android app store, Google Play [19]), on which malicious apps may be distributed to users.

To protect mobile devices, users often rely on anti-malware products, which scan apps to determine if they are benign or malicious. However, many malware apps have previously evaded detection by these products. Examples of such malicious apps include Brain Test [7], VikingHorde [12], FalseGuide [18], and DressCode [8]. These apps have infected millions of users before they were detected. To evade detection, malware authors often rely on *code obfuscation.*

Code obfuscation transforms code into a form that is more difficult for humans, and possibly machines, to read, understand, and reverse engineer. These transformations change the syntax of code but not their semantics [30]. These changes could be small (e.g., inserting unused code) or sophisticated (e.g., performing bytecode encryption)[44]. Although code obfuscations are used by malware authors, they are also used by benign app developers to increase the difficulty of reverse engineering their apps.

To better protect the intellectual property of benign app developers and prevent cloning of their apps, several companies have developed obfuscation tools, or *obfuscators* for short, that implement different code transformations (e.g., identifier renaming, string encryption, reflection, etc.). Given the use of obfuscations by malware authors, the goal of this study is to assess the performance of commercial anti-malware products against various obfuscation tools and strategies.

Although some researchers have studied an individual obfuscation tool's effectiveness on a limited number of anti-malware products [32, 34, 40, 43, 44, 47], no study has performed a large-scale assessment of (1) the effect of individual and combined obfuscation strategies provided by multiple obfuscations tools on anti-malware products, (2) the effect the tools and strategies have on the accuracy of anti-malware products for benign apps and not just malicious apps, (3) the effect of time on obfuscated-app detection by those products, and (4) whether the application of obfuscation strategies result in valid, installable, and runnable apps. Due to the lack of a study regarding the effect of specific and combined obfuscation strategies on anti-malware products, it is unclear which strategies evade such products the most. None of the aforementioned studies determine the extent to which anti-malware products erroneously consider obfuscated, benign apps as malicious, which is undesirable for both anti-malware product vendors and benign app developers.

To determine if the transformations applied by obfuscation tools break an app's semantics, our study investigates the ability of obfuscation tools to generate valid, installable, and runnable apps. An obfuscated app is not useful to a benign app developer or malicious author if it cannot be executed on a device or if its benign, functional behavior changes. To ensure an app's obfuscation is

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180228

successful, our study further compares the behavior of an obfuscated app with the behavior of its corresponding original app.

Overall, this paper makes the following contributions:

- We assess the accuracy of over 60 anti-malware products on apps obfuscated using 7 obfuscation tools and 29 obfuscation strategies on 3,000 benign apps and 3,000 malicious apps, totaling over 73,000 obfuscated apps. We further consider the effect of an app's age on that accuracy.
- We evaluate the ability of 7 obfuscation tools to generate Android apps that are valid, installable, and runnable.
- Based on our results, we make suggestions for improving anti-malware products and obfuscation tools.
- To conduct this study, we have implemented a framework for obfuscating Android apps and scanning them using anti-malware products. The framework is reusable, can be extended to include more obfuscation tools and strategies, and is available online [21], along with our resulting dataset of over 73,000 obfuscated apps.

The remainder of this paper is organized as follows. Section 2 covers background information about Android apps and code obfuscation. Section 3 discusses the research questions that this study aims to answer. The research methodology of this study is presented in Section 4. The results and the findings are reported in Section 5. Section 6 discusses the results and provides recommendations to enhance anti-malware products and obfuscation tools. The threats to validity are presented in Section 7. Finally, the paper overviews the related literature (Section 8) and concludes (Section 9).

2 BACKGROUND

This section provides a brief overview of Android apps and obfuscation strategies to help the reader understand the rest of the paper.

2.1 Android Apps

Each Android app is packaged and distributed as an Android Package Kit (APK) file, which is a zipped file that is mainly written in Java. Each APK file contains a manifest file, resources (e.g., images), and the app's bytecode. An app's code is compiled into *Dalvik EXecutable* (*DEX*) format, which can be executed on a customized Java Virtual Machine (JVM). There are two JVMs that can execute the DEX format: Android Runtime (ART), introduced in Android version 5 (Lollipop); and Dalvik Virtual Machine (DVM), for older versions. *classes.dex*, the main DEX file of an Android app, is a file in the APK generated by *dx*, a utility that converts *.class* files into a DEX file.

classes.dex can be disassembled by Baksmali [23] into an Intermediate Representation (IR) format which, in turn, can be assembled by Smali [23] to generate a new variant of *classes.dex*. The new *classes.dex* can be repackaged using a tool such as *Apktool* [1], a reverse-engineering tool for Android APK files, to generate a new APK variant (e.g., an obfuscated app). Different IR formats can be generated from *classes.dex*, including Smali code using Apktool and *.class* files using DARE [42] or dex2jar [16]. Moreover, Soot [45] can generate various IRs such as Baf, Jimple, Shimple, Grimp, or even a low-level IR such as Jasmin.

2.2 Obfuscation Strategies

To study the effectiveness of anti-malware products, we applied several different obfuscation strategies on each Android app. We use the term *obfuscation strategy* to refer to a single transformation or multiple transformations applied to an Android app. We consider three types of strategies: *trivial strategies*, *non-trivial strategies*, and *combined strategies*. Table 1 presents abbreviations of the trivial and non-trivial obfuscations, which will be used throughout this paper.

Table 1: Obfuscation-strategy abbreviations

Trivial Obfuscation		Non-trivial Obfuscation			
Disassembly/ Reassembly	DR	Junk code insertion	JUNK	Identifier renaming	IDR
AndroidManifest	MAN	Class renaming	CR	Control flow	CF
Alignment	ALIGN	Member reordering	MR	Reflection	REF
Repackaging	REPACK	String encryption	ENC		

Trivial obfuscation strategies are code transformations that do not change the app's bytecode. For this study, we examined the following trivial strategies:

- Repackaging (REPACK) involves unzipping the APK file and re-signing it with a different signing certificate. This simple transformation thwarts anti-malware products that rely on the app's certificate to determine if the app is malicious or not. For this transformation, we unzip an APK file using the *zipfile* Python library and resign it with our own signing certificate using *jarsigner* [20], a tool for verifying and generating digital signatures for JAR files.
- Disassembling and Reassembling (DR) involves disassembling the app using a reverse-engineering tool, such as Apktool, reassembling the app, and then signing it. By disassembling and reassembling the app, the items in *classes.dex* will be reordered. Anti-malware products that rely on matching *classes.dex* against signatures of known malicious apps would be broken.
- AndroidManifest transformation (MAN): Each Android app contains a configuration file called *AndroidManifest.xml* file, which specifies the principal components that constitute the application, including their types and capabilities, as well as required and enforced permissions. This transformation changes the manifest by adding permissions or adding components' capabilities, called *Intent Filters* in Android.
- Alignment (ALIGN) realigns all uncompressed data, such as images or raw files, in an APK file. This transformation changes the cryptographic hash of an APK file. Therefore, if an anti-malware product identifies malicious apps based on their cryptographic hashes (e.g., MD5), this transformation can thwart it.

Non-trivial obfuscation strategies are code transformations that change an app's bytecode. We study the following non-trivial obfuscation strategies:

- Junk code insertion (JUNK) adds code that does not affect the execution of an app. Junk code insertion can add null operations (nop), comments, and/or debugging information to a *classes.dex* file.
- String encryption (ENC) encrypts the strings in *classes.dex* and adds a function that decrypts the encrypted strings at runtime. Anti-malware products that rely on the string data in an app to determine if it is malicious will be evaded by this transformation.
- Control-flow manipulation (CF) changes the methods' control-flow graph by adding conditions and iterative constructs. In addition, this transformation changes the app's call graph by adding new methods and fake calls to the newly added methods.
- Members reordering (MR) changes the order of instance variables or methods in a *classes.dex* file, which evades anti-malware products that depend on the sequence of members in a class.
- Identifier Renaming (IDR) renames the instance variables and/or the method names in each Java class with randomly generated names. This transformation changes signatures generated from identifiers and changes the *method table* in Dalvik bytecode.

- Class renaming (CR) renames the classes and/or the packages in an app with randomly generated names. This transformation changes the *method table* in the Dalvik bytecode.
- Reflection (REF) transformations convert direct method invocations into reflective calls using the Java reflection API, which can evade static analyses that rely on direct method calls.

Combined strategies are combinations of the aforementioned obfuscation strategies. Previous work [40, 44] has mentioned that combining obfuscation strategies result in stronger obfuscations. Our study leverages different combined strategies to understand which combinations of transformations will result in better evasion of anti-malware products. The majority of our leveraged combined strategies have not been empirically studied in previous work.

3 RESEARCH QUESTIONS

In this paper, our primary goal is to provide a large-scale empirical study that evaluates the effectiveness of anti-malware products against various obfuscation tools and strategies. To that end, this section presents and discusses the research questions this study attempts to answer. Moreover, this section shows who will benefit from answering each research question. In the remainder of this section, we introduce and motivate each research question that we study.

RQ1. How is the accuracy of anti-malware products affected by obfuscation strategies?

The use of code obfuscation in Android apps has become popular and is leveraged by both benign and malicious app developers. Given that smartphone users rely on anti-malware products to protect their devices, it is crucial for anti-malware products to distinguish malicious apps from benign ones with high accuracy, while being resilient to obfuscation. RQ1 aims to measure the accuracy of commercial anti-malware products against a broad range of obfuscation strategies. We measure accuracy in this paper using precision and recall, since these metrics take into account false positives (i.e., benign apps marked as malicious) and false negatives (i.e., malicious apps marked as benign).

Anti-malware providers will benefit from answers to RQ1 in order to improve their products, especially against the obfuscation strategies that thwart their products the most. In addition, the answers to RQ1 can help smartphone users choose between anti-malware products. Benign app developers will benefit from answers to RQ1 by learning which obfuscation strategies prevent their apps from being flagged as malicious.

RQ2. How is the accuracy of anti-malware products affected by obfuscation tools?

Each anti-malware product's effectiveness likely varies based on the implementations of obfuscation strategies provided by an obfuscation tool. To make that determination, RQ2 measures the accuracy of anti-malware products on obfuscation tools, where accuracy is again measured in terms of precision and recall.

Anti-malware product vendors, benign app developers, and obfuscation tool developers can benefit in several ways from the answers to RQ2. Anti-malware product vendors can use this information to determine which specific implementations of obfuscation strategies may cause false positives (i.e., benign apps marked as malicious) in their products. Similarly, these vendors can benefit from learning which obfuscations result in successful evasion from detection by malicious apps. Answers to RQ2 can aid benign app developers in choosing the obfuscation tools that will prevent their apps from erroneously being marked as malicious. Furthermore, if false positives or false negatives (i.e., malicious apps marked as benign) are due to obfuscation tools, as opposed to the anti-malware products, then this information is useful for correcting obfuscation tools.

RQ3. How is the accuracy of anti-malware products affected by the year an app is created?

RQ3 aims to study the accuracy of anti-malware products on non-transformed and transformed apps over different time periods, where each time period for our study spans two years. We consider transformed apps as belonging to the same time period as their non-transformed versions. For example, if we transform apps created in time period 2012-2013, we still consider the resulting obfuscated apps as created in 2012-2013, for the purposes of RQ3.

This research question allows us to understand the effectiveness of anti-malware products when applied to different time periods and to determine if those products' detection accuracies are affected by time. Anti-malware vendors can use this information to determine the time periods that result in poor accuracy for their products, aiding them with improving results for apps created during those problematic time periods.

RQ4. To what extent do obfuscation tools result in valid, installable, and runnable apps?

Although an obfuscated app does not need to be runnable when scanned by an anti-malware product, developers of benign apps and obfuscation tools rely on those tools to produce valid, installable, and runnable apps. Similar to [34], we consider an APK to be *valid* if an obfuscation tool successfully generates a signed APK package that includes a *classes.dex* file containing correct Dalvik bytecode syntax. An app is *installable* if it can be successfully deployed into the Android runtime. For our purposes, a transformed app is *runnable* if its runtime behavior is similar to its non-transformed version. RQ4 is particularly useful for obfuscation tool developers since answers to that question provide information about transformations that result in malformed apps.

4 RESEARCH METHODOLOGY

This section describes the research methodology that we pursued in terms of our study subjects, selected obfuscation tools, our evaluation framework, and our selected anti-malware products.

4.1 Study Subjects

We used a dataset of benign apps consisting of 3,000 apps from Google Play and 3,000 malicious apps. To avoid having malicious apps in our ground-truth dataset of benign apps, we obtained benign apps from AndroZoo [26], which is a collection of more than 5.5 million apps collected from several sources, including Google Play. AndroZoo apps have been scanned by commercial anti-malware products using the VirusTotal service [4], a free online service provided by Google that scans URLs, files, and Android apps. Approximately 25,000 Google Play apps out of nearly 2 million apps in AndroZoo are marked as benign by all anti-malware products. From these 25,000 apps, we have randomly selected 3,000 apps for this study. The malicious apps belong to several malware repositories including Android Malware Genome [49], Contagio [6], AndroTotal [39], the Drebin dataset [27] and VirusShare [5]. In addition to these malware repositories, we used the VirusTotal service to include recently discovered malicious apps that belong to the following malware families: BrainTest [7], VikingHorde [12], and FalseGuide [18].

4.2 Obfuscation Tools

We have included the following obfuscation tools for our study, whose supported obfuscation strategies are depicted in Table 2:

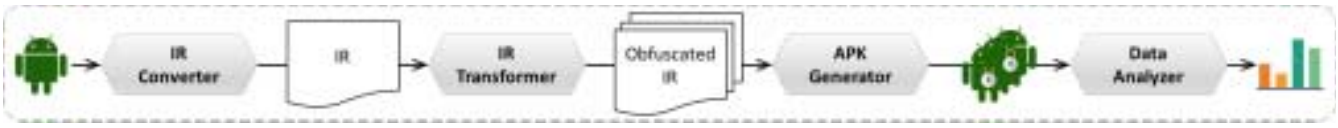

Figure 1: Obfuscation study methodology

Table 2: Obfuscation strategies of each obfuscation tool

	Trivial				Non-trivial						
Obfuscator/Strategy	ALIGN	DR	MAN	REPACK	CF	CR	ENC	IDR	JUNK	MR	REF
Apktool/Jarsigner		✓		✓							
Allatori					✓		✓	✓		✓	
DashO					✓		✓	✓			
DroidChameleon			✓		✓	✓	✓		✓		✓
ADAM	✓				✓		✓	✓	✓		
ProGuard								✓			

- Allatori [3] is a commercial Java and Android obfuscation tool that supports a wide range of obfuscation strategies. Many companies such as Amazon, Fujitsu, and Motorola rely on Allatori to protect their software systems from being reverse engineered. The providers of this tool, Smardec Inc., provided us with a full version for educational purposes.
- ProGuard [22] is a widely used open-source shrinker, optimizer, obfuscator, and preverifier for Java bytecode. A preverifier performs certain checks on Java bytecode prior to runtime. ProGuard supports identifier renaming and is the default tool in many development environments, including Android Studio [14], the official IDE for Android apps.
- ADAM [47] is a research tool for obfuscating Android apps. It transforms the Smali code of a reversed-engineered app.
- DroidChameleon [44] is a state-of-the-art research tool for obfuscating Android apps which supports a wide range of obfuscation strategies. Compared to ADAM, DroidChameleon supports more complex transformations. Like ADAM, DroidChameleon transforms the Smali code of a reversed-engineered app.
- DashO [15] is a commercial tool for obfuscating Android and Java applications. DashO provides static analysis protection and runtime security control against tampering, unauthorized debugging, and some runtime attack patterns. This tool supports control-flow, string-encryption, and identifier-renaming transformations. The providers of DashO, PreEmptive Solutions, supplied us with a full free version valid for 30 days.
- Apktool and Jarsigner were used to perform the DR and REPACK obfuscation strategies, respectively. These two transformations often work in tandem because a reassembled APK must be resigned.

We also considered another tool, DexGuard [17], which is an advanced and commercial version of ProGuard. We contacted the providers of DexGuard to obtain an educational or commercial version of their tool to run on our dataset. Unfortunately, they only allow their tool to run on a restricted number of Android apps; and they do not sell licenses for research purposes. Hence, we did not include it in this study.

4.3 Evaluation Framework

To conduct our study, we have developed the framework depicted in Figure 1, which consists of the following four modules: *IR Converter*, *IR Transformer*, *APK Generator*, and *Data Analyzer*. *IR Converter* takes an Android APK as input and converts its code to Intermediate Representation (IR) formats. *IR Transformer* utilizes all obfuscation tools to transform the IR format using a variety of obfuscation strategies. *APK Generator* repackages each obfuscated IR file and generates an obfuscated APK from that file. *Data Analyzer* scans obfuscated apps using anti-malware products, stores the scanning results in a MySQL database, analyzes the scanning results, and creates various statistical reports.

Our framework is reusable and extendable. A user can add new obfuscation tools and support different obfuscation strategies. Therefore, we make the framework available for researchers and practitioners [21]. The framework is a Python program that consists of more than 5,500 lines of code, not counting the obfuscation tools.

IR Converter. Obfuscation tools do not require source code and they work directly on the IR format. Therefore, this module converts an APK file to two IR formats: *smali* using Baksmali and Java bytecode using *dex2jar*. In our framework, we generate these two IR formats since ADAM and DroidChameleon work on *smali* code while all other obfuscation tools work on Java bytecode.

IR Transformer. This module generates several obfuscated IR files of the original IR file. The framework is configured to leverage twenty nine different obfuscation strategies using seven obfuscation tools (recall Section 4.2).

APK Generator. For each obfuscated IR file, this module generates an obfuscated Android app. First, this module leverages the *dx* tool from the Android SDK to convert an obfuscated IR to a *classes.dex* file. Next, it generates an APK file with the new *classes.dex* using Apktool. Finally, the APK file is signed using *jarsigner* with our own certificate, since the original certificate of the app cannot be obtained.

Data Analyzer. This module uses the VirusTotal service to scan apps using anti-malware products. This module uploads the apps to VirusTotal, which scans them using more than 60 up-to-date commercial anti-malware products. For each uploaded app, VirusTotal returns a unique scanning ID, which Data Analyzer uses later to download the scanning reports and stores them in a MySQL database. Data Analyzer queries and processes the database to generate various statistical reports.

4.4 Anti-malware Products

We have evaluated the accuracy and the resiliency of 61 commercial anti-malware products against obfuscations. Due to space limitations and to ensure readability, we focus on the results of the 21 most popular Android anti-malware products in this paper; however, we make the results for all 61 anti-malware products available online [21].

Table 3 shows the anti-malware products evaluated in this study and includes the following information for each product: its number of *Downloads*; its overall user satisfaction score as represented using a star-rating (*Stars*); and the number of users who reviewed the product (*Reviewers*). The numbers in Table 3 are obtained from Google Play.

5 DATA ANALYSIS AND RESULTS

For conducting our experiments, we have leveraged a high performance computing cluster (HPC), managed by our organization, that has more than 200 compute nodes with a total of more than 8,000 cores. Each compute node has 264GB-512GB RAM. We utilized HPC

Table 3: Anti-malware products (K: Thousand. M: Million)

Product	Downloads	Stars	Reviews	Product	Downloads	Stars	Reviews
Ikarus	100K - 500K	4.2	2,862	Trustlook	10M - 50M	4.4	476,671
Emsisoft	100K - 500K	4.2	1,425	McAfee	10M - 50M	4.4	506,491
Fortinet	100K - 500K	4.2	2,086	Avira	10M - 50M	4.5	441,016
AegisLab	100K - 500K	4.2	2,905	Norton Symantec	10M - 50M	4.5	946,230
F-Secure	500K - 1M	4.1	12,183				
Comodo	500K - 1M	4.6	33,395	ESET-NOD32	10M - 50M	4.7	490,840
GData	1M - 5M	4.0	8,850	Kaspersky	10M - 50M	4.7	2,061,983
Sophos	1M - 5M	4.3	11,816	DrWeb	50M - 100M	4.5	1,044,410
TrendMicro	1M - 5M	4.6	49,977	Antiy-AVL	100M - 500M	4.1	2,166
BitDefender	5M - 10M	4.5	88,809	Avast	100M - 500M	4.5	4,724,478
CAT-QuickHeal	5M - 10M	4.4	204,709	AVG	100M - 500M	4.5	5,785,171

to run thousands of jobs simultaneously. On each app, we applied 29 different obfuscation strategies: 4 trivial transformations, 7 non-trivial transformations, and 18 combined transformations. Table 4 shows the number of obfuscated apps resulting from applying the 29 obfuscation strategies leveraged by the obfuscation tools. An empty cell indicates an obfuscation strategy that is not support by a particular obfuscation tool. In total, we have generated 73,362 obfuscated apps from 3,000 benign apps and 3,000 malicious apps.

In the remainder of this section, we present the results of our experiments. We measured the effectiveness of anti-malware products at identifying malicious apps in terms of their *precision*, which measures the extent to which benign apps are labeled as malicious, and *recall*, which measures the extent to which malicious apps are labeled as benign. We use the *F-score*, i.e., the harmonic mean of precision and recall, to measure the overall detection rate of anti-malware products.

5.1 RQ1. Obfuscation Strategies

We studied the accuracy of anti-malware products with respect to a wide variety obfuscation strategies in two scenarios. In the first scenario (Section 5.1.1), we compare the detection rates of each anti-malware product on the original dataset and the obfuscated dataset. In the second scenario (Section 5.1.2), we measure the detection rate of anti-malware products against each obfuscation strategy.

5.1.1 Detection rate on original and obfuscated apps. Figure 2 shows the detection rate of 21 anti-malware products on the original dataset of 6,000 apps, depicted as black bars, and the obfuscated dataset of 73,362 apps, depicted as gray bars. Figure 2 demonstrates that the detection rate of anti-malware products on the original dataset is above 85% for 16 products, and between 75% and 85% for 4 anti-malware products. *TrendMicro* exhibits the lowest detection rate, 56%. The average detection rate is 87% on the original dataset. Consequently, prior to application of obfuscation strategies from obfuscation tools, these top anti-malware products are quite effective at protecting Android users; albeit there is room for improvement.

Once obfuscation strategies are applied, the detection rates for those anti-malware products decrease significantly, as shown in Figure 2. For example, *AegisLab* achieves the highest detection rate on the original dataset, 96%, since it mislabeled only 247 apps in the original dataset. Its detection rate has dropped to 55% on the obfuscated dataset—a 40% decrease—as it mislabeled 27,636 apps. Other anti-malware products are also severely impacted by code obfuscation. While the average detection rate of anti-malware products on the original apps is 87%, the average detection rate on the obfuscated dataset is 67%—a 20% decrease.

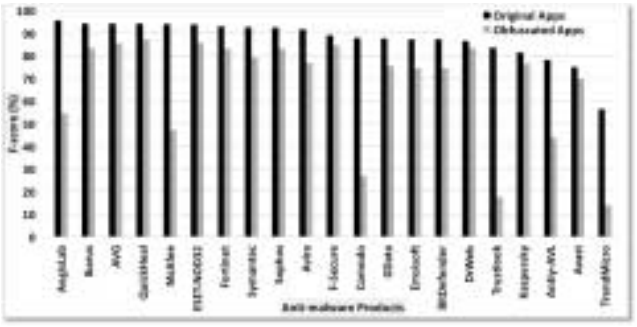

Figure 2: Detection rate of 21 anti-malware products on 6,000 original apps and 73,362 obfuscated apps.

Finding 1: Code obfuscation significantly impacts Android anti-malware products. The average detection rate for the top anti-malware products decreases from 87% to 67%—a 20% decrease.

5.1.2 Detection rate for each obfuscation strategy. To better understand the impact of every obfuscation strategy on each anti-malware product, Table 5 presents the detection rate of anti-malware products, expressed as the F-score, on the original dataset and the various obfuscation strategies. For example, the detection rate of *Symantec* on the original dataset is 93%. This detection rate has dropped to 64% on obfuscated apps using MAN, to 69% on obfuscated apps using ENC, and to 31% on obfuscated apps using ENC_IDR.

Table 5 demonstrates that the majority of anti-malware products are not affected by REF. We consider a transformation's effect on a product's detection rate to be negligible if the detection rate has either improved, decreased by less than 3%, or remains above 85%. In fact, the accuracy of *F-Secure*, *GData*, *BitDefender*, and *Emsisoft* improves on apps obfuscated using REF. This result indicates that the intensive use of *code reflection* makes an app look suspicious to our studied anti-malware products, improving their detection rates. Unfortunately, this phenomenon may result in false positives for certain anti-malware products. For instance, *AVG* erroneously marked 307 benign apps obfuscated using REF as malicious, while also correctly detecting nearly all malicious apps obfuscated using REF.

Finding 2: REF transformations make apps look suspicious, increasing the chance of an app being labeled as malicious.

Perhaps most surprising is that certain trivial obfuscation strategies are quite effective against the top anti-malware products. Notably, the anti-malware products that we studied rely heavily on analyzing an app's manifest file, which contains configuration information. Consequently, these products are often evaded by apps obfuscated using MAN, which involves the trivial addition or modification of permissions or Intent filters. For example, the detection rate of *McAfee* dropped from 94% to 21% for apps obfuscated using MAN. Overall, the average detection rate of anti-malware products fell to 60% from 87% when apps are obfuscated using MAN—a 28% decrease.

Finding 3: MAN, which is a trivial obfuscation strategy, severely impacts many anti-malware products, on average, decreasing a product's detection rate by 28%.

Another interesting, possibly counter-intuitive, conclusion that we can draw from Table 5 is that combined transformations are not always superior to individual transformations. For instance, while

Table 4: Number of obfuscated apps using the obfuscation strategy in the column leveraged by the obfuscator in the row.

	Trivial				Non-trivial							Combined Strategies																			
Obfuscator/Strategy	ALIGN	DR	MAN	REPACK	CF	CR	ENC	IDR	JUNK	MR	REF	CF_ENC	CF_IDR	CF_MAN	CF_MR	CR_MAN	ENC_IDR	ENC_MAN	JUNK_MAN	MAN_REF	CF_CR_MAN	CF_ENC_IDR	CF_ENC_MR	CF_IDR_MR	CF_REF_MAN	ENC_REF_MAN	CF_ENC_IDR_MR	CF_CR_ENC_MAN_REF	CF_CR_ENC_JUNK_MAN_REF	Total apps
Apktool/Jarsigner		3,693		5,880																										9,573
Allatori					1,612		1,613	1,609		1,612					1,609								1,607	1,607			1,606			12,875
DashO					1,094		1,089	1,097				1,082	1,084				1,077					1,083									7,606
DroidChameleon			3,597		2,593	1,952	687		351		1,487	610		1,594		1,752		679	349	1,385	1,361					993	658		570	314	20,932
ADAM	5,487				0	4,175	2,708	4,119	4,182																						20,671
ProGuard								1,705																							1,705
Total apps	5,487	3,693	3,597	5,880	5,299	6,127	6,097	8,530	4,533	1,612	1,487	1,692	1,084	1,594	1,609	1,752	1,077	679	349	1,385	1,361	1,083	1,607	1,607	993	658	1,606	570	314	73,362

Table 5: (RQ1) Detection rate of anti-malware products, measured by their F-score (%), against each obfuscation strategy.

		Trivial				Non-trivial							Combined Strategies																	
Anti-malware	Original	ALIGN	DR	MAN	REPACK	CF	CR	ENC	IDR	JUNK	MR	REF	CF_ENC	CF_IDR	CF_MAN	CF_MR	CR_MAN	ENC_IDR	ENC_MAN	JUNK_MAN	MAN_REF	CF_CR_MAN	CF_ENC_IDR	CF_ENC_MR	CF_IDR_MR	CF_REF_MAN	ENC_REF_MAN	CF_ENC_IDR_MR	CF_CR_ENC_MAN_REF	CF_CR_ENC_JUNK_MAN_REF
AegisLab	96	84	52	36	92	52	35	53	39	62	66	58	41	11	57	66	37	3	68	61	58	43	4	65	66	66	68	63	58	35
Ikarus	94	95	94	66	96	75	84	81	86	86	93	88	75	77	84	88	85	64	89	86	89	86	60	84	87	93	89	82	92	87
CAT-QuickHeal	94	95	93	92	94	89	**91**	75	88	89	93	94	73	**92**	93	93	91	55	91	90	94	91	54	76	91	94	84	**70**	89	80
AVG	94	75	96	63	96	79	83	85	91	84	97	88	77	**77**	83	97	85	58	90	85	89	85	57	92	96	92	90	**92**	91	85
McAfee	94	90	50	21	93	47	20	45	52	16	73	23	20	41	17	73	20	21	22	20	22	17	22	66	66	15	18	63	14	20
ESET-NOD32	94	94	92	91	94	93	93	80	91	46	94	66	75	92	92	94	93	68	86	46	66	91	61	80	92	68	59	72	80	57
Fortinet	93	94	89	86	93	88	83	78	84	75	91	86	67	79	91	88	83	58	87	77	87	84	50	83	83	89	75	72	72	56
Symantec	93	86	87	64	88	76	84	69	79	84	92	88	63	68	83	92	85	31	90	85	89	85	31	75	90	92	90	73	91	85
Sophos	93	93	91	90	93	89	85	70	79	93	88	92	70	84	93	87	86	52	91	95	92	87	50	66	72	93	91	51	91	85
Avira	92	92	87	84	92	85	84	65	78	78	87	60	61	78	86	86	85	38	80	80	59	83	38	69	85	58	33	63	73	61
F-Secure	89	87	87	85	90	85	82	81	84	95	90	94	73	65	91	90	82	53	93	94	92	84	53	87	88	93	93	84	80	71
Comodo	88	88	27	16	82	22	17	19	24	17	19	15	17	20	11	33	16	20	14	18	16	11	20	20	26	9	14	23	11	18
GData	88	91	84	75	88	79	61	75	77	95	85	91	62	54	79	85	50	46	83	79	81	50	45	79	77	81	82	77	57	41
BitDefender	87	90	84	73	88	78	60	74	75	95	85	91	61	46	76	85	45	44	83	78	78	46	43	79	77	77	83	77	58	41
Emsisoft	87	90	84	73	88	78	60	74	75	95	85	91	61	46	76	85	45	43	83	78	78	46	43	79	77	77	83	77	59	41
DrWeb	87	88	83	81	88	89	86	90	86	93	88	40	92	89	90	88	86	90	94	94	41	90	89	88	87	39	40	87	36	35
Trustlook	84	10	23	0	48	17	0	22	20	0	36	0	2	3	0	38	0	1	0	0	0	0	2	40	39	0	0	40	0	0
Kaspersky	81	83	75	70	82	81	76	70	75	88	81	80	73	77	81	81	77	50	86	89	81	81	50	70	77	83	84	64	91	85
Antiy-AVL	78	79	56	26	80	41	22	47	47	13	68	18	20	25	12	70	21	24	12	8	20	12	25	70	69	12	15	65	8	7
Avast	75	75	63	57	75	73	66	66	66	78	74	75	71	67	78	74	68	46	91	79	76	76	45	60	69	83	91	47	91	86
TrendMicro	56	57	11	7	48	12	7	10	14	6	16	10	9	15	6	16	7	11	5	7	10	5	10	12	14	7	4	12	3	5
AVERAGE	**87**	**83**	**72**	**60**	**85**	**68**	**61**	**63**	**67**	**66**	**76**	**64**	**55**	**57**	**66**	**77**	**59**	**42**	**68**	**64**	**63**	**60**	**41**	**68**	**73**	**63**	**61**	**64**	**59**	**51**

the detection rate of *AVG* against CF is 79%, its detection rate against combined transformations that include CF is between 57% and 97%.

Finding 4: In general, combined transformations do not affect detection rates more than single transformations: The average detection rate of anti-malware products is 61% for single non-trivial obfuscations, and 61% for combined obfuscations.

Figure 3 contains box-and-whisker plots illustrating the impact of each obfuscation strategy on all anti-malware products. These results suggest that some obfuscation strategies have negligible effects on the majority of anti-malware products. For example, REPACK did not affect 19 anti-malware products. Similarly, the use of the MR transformations did not affect 14 anti-malware products. Lastly, the REF transformation did not thwart the majority of anti-malware products.

Figure 3 demonstrates that ENC_IDR and CF_ENC_IDR are very effective in thwarting anti-malware products. In fact, these two transformations evaded all anti-malware products except *DrWeb*.

Finding 5: ENC_IDR and CF_ENC_IDR are the most successful transformations for evading anti-malware products.

5.2 RQ2. Obfuscation Tools

For RQ2, we studied the detection rate of anti-malware products on apps transformed using various obfuscation tools. To that end, we analyze the results of each anti-malware product's detection rate on each obfuscation tool. We further assess the overall effect of each obfuscation tool across all studied anti-malware products.

Table 6 depicts the detection rate of each anti-malware product on apps transformed using each obfuscation tool. From Table 6, we

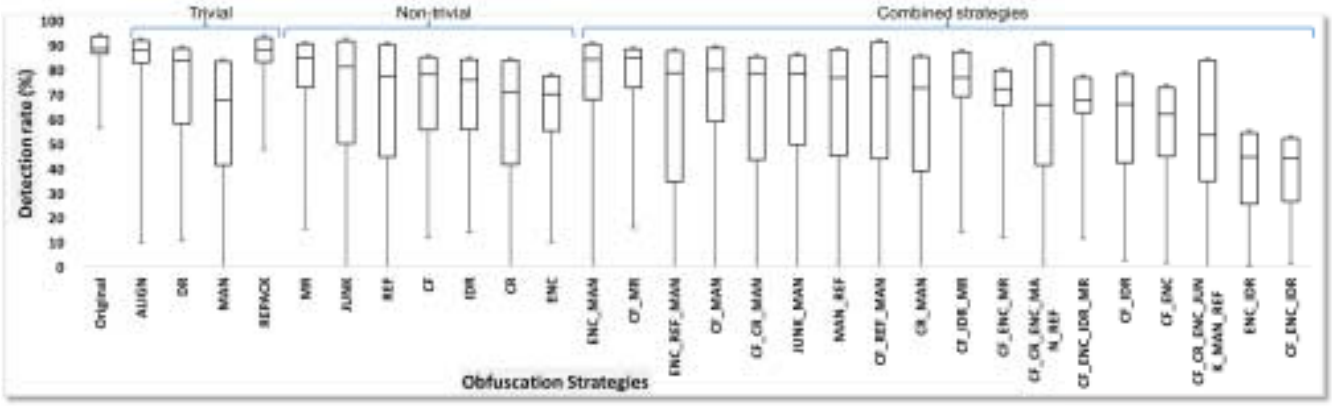

Figure 3: (RQ1) The average detection rate of all anti-malware products regarding each obfuscation strategy.

Table 6: Detection rate of anti-malware products (F-score (%)) against each obfuscation tool

Anti-malware	Original	ADAM	Apktool/ Jarsigner	Allatori	Droid Chameleon	ProGuard	DashO
AegisLab	96	83	79	66	53	9	13
Ikarus	94	95	95	88	82	80	71
CAT-QuickHeal	94	95	94	86	90	80	75
AVG	94	75	96	95	81	90	71
McAfee	94	90	79	68	19	37	31
ESET-NOD32	94	94	93	88	83	90	80
Fortinet	93	94	91	85	85	74	69
SymantecMobileInsight	93	86	88	86	82	72	51
Sophos	93	93	92	75	90	78	69
Avira	92	92	90	80	77	62	60
F-Secure	89	87	89	88	89	92	61
Comodo	88	88	65	25	14	23	20
GData	88	90	86	81	76	90	50
BitDefender	87	90	87	81	74	90	45
Emsisoft	87	90	86	81	74	90	45
DrWeb	87	88	86	88	77	81	89
Trustlook	84	10	37	39	0	0	2
Kaspersky	81	83	79	76	81	70	66
Antiy-AVL	78	79	72	69	16	23	25
Avast	75	75	71	67	77	60	58
TrendMicro	56	57	35	14	7	10	12
AVERAGE	87	83	80	73	63	62	51

observe that some anti-malware products are severly impacted by all obfuscation tools. For example, the detection rate of *Trustlook* dropped to less than 40% on apps obfuscated using any of our studied tools. Furthermore, *Trustlook* marked all apps obfuscated by the following tools: DroidChameleon, ProGuard, and DashO as benign apps. Likewise, *TrendMicro* and *Comodo* are evaded by all obfuscation tools, except ADAM.

The box-and-whisker plot shown in Figure 4 depicts the effect of each obfuscation tool on all anti-malware products. The figure shows that the top anti-malware products are resilient against Apktool/Jarsigner, ADAM, and Allatori. At the same time, DashO evades the top anti-malware products more often than the other products.

We further assessed the variability of each obfuscation tool on our studied anti-malware products, which indicates how consistently each tool affects the accuracy of those products. To that end, we considered the interquartile range (IQR) of the box plots in Figure 4.

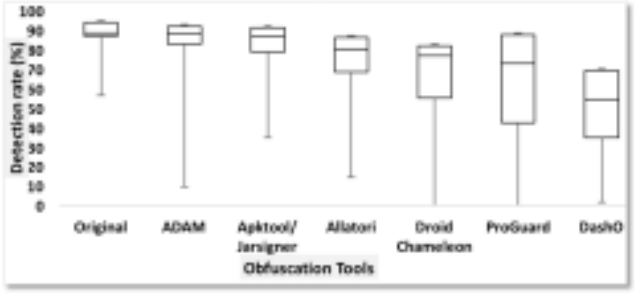

Figure 4: (RQ2) The average detection rate of all anti-malware products regarding each obfuscation tool.

IQR is the difference between the lower bound and the upper bound of a box, which conveys the central tendency of top anti-malware products against an obfuscation tool. A small IQR indicates that the behavior of an anti-malware product is highly consistent. Figure 4 shows that ProGuard has the largest IQR.

ADAM and Apktool/Jarsigner, as shown in Figure 4, have a relatively high median, 88% and 86%, respectively, with the lowest IQR, i.e., 9% and 13%, respectively. This indicates that anti-malware products are resilient to these tools. Consequently, apps obfuscated by ADAM and Apktool/Jarsigner work well for benign app developers, who would want to obfuscate apps without having them be falsely reported as malicious, and would be least useful for malware authors.

Finding 6: ADAM and Apktool/Jarsigner produce obfuscations that reduce anti-malware product accuracy the least.

Figure 4 suggests that DashO is most successful at evading anti-malware products, which aids malware authors. The average detection rate of anti-malware products on obfuscated apps using DashO is 51% with a median of 58%—a 37% decrease in the average detection rate, and a 32% median decrease.

Finding 7: DashO reduces the accuracy of anti-malware products more than other obfuscation tools in our study.

5.3 RQ3. Time-Aware Analysis

A significant factor that may interact with the effect of obfuscations on anti-malware product accuracy is time. For RQ3, we conducted a time-aware analysis that studies the accuracy of anti-malware products on original and obfuscated apps that belong to the same time period for the past 10 years. Figure 5 depicts the results of this

analysis. We grouped apps into two-year time periods, due to the fact that some years only have a few apps, mainly 2009 with 29 apps, and 2017 with 130 apps. Similar to [25], we consider the year of the last modified date of *classes.dex* in an app as the year from which it originates. We consider any transformed app as belonging to the same year as its original version, in order to determine the actual effect of obfuscation on product accuracy for each time period.

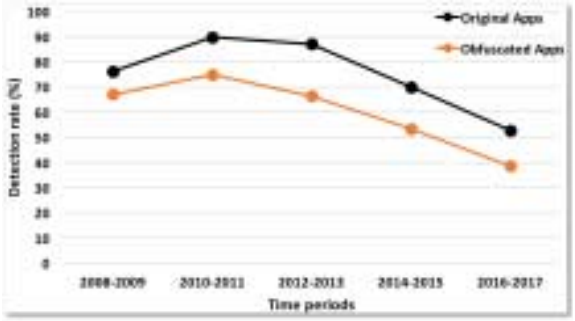

Figure 5: (RQ3) Time-aware analysis.

For the original dataset, Figure 5 shows that the top anti-malware products maintain similar detection rates for any two consecutive time periods prior to and including 2012-2013. Unfortunately, there are significant decreases in the detection rate starting from the time period between 2012-2013 and 2014-2015; the average detection rate is 70% for 2014-2015, falling from 87% in 2012-2013. By 2016-2017, the average detection rate falls to 53% on the original apps.

For the obfuscated dataset, Figure 5 illustrates that the detection rates of anti-malware products decrease in a largely linear fashion. The average detection rate starts at 67% on obfuscated apps from 2008-2009 and decreases to 39% on obfuscated apps from 2016-2017. These results suggest that anti-malware products are slow to adopt signatures of malicious apps.

The average detection rate on original apps from 2010 to 2013 are higher than the average detection rate on older apps. This likely occurs because many malicious apps from 2010-2013 are well-known and highly disseminated among security analysts. For example, Android Malware Genome is a dataset of malicious Android apps that are widely used, are described in a highly cited paper [49], and were released in the 2010-2013 time period.

Finding 8: The average detection rates of anti-malware products tend to decrease over time, indicating that such products are slow to adopt signatures of malicious apps.

5.4 RQ4. Valid, installable, and runnable apps

An obfuscated app is not useful for app authors, unless the app can run on a device. For that reason, RQ4 measures the ability of obfuscation tools to generate valid, installable, and runnable apps.

5.4.1 Valid apps. Recall from Section 3, a *valid* obfuscated app corresponds to a signed APK package that includes a *classes.dex* file containing the correct Dalvik bytecode syntax. Table 7 depicts the ability of obfuscation tools to generate valid obfuscated apps, which we refer to as the *obfuscation rate*. The obfuscation rate is measured using the ratio of the number of valid apps generated by an obfuscation tool to the number of apps successfully retargeted to Java bytecode. For example, ProGuard obfuscated 684 benign apps out of 1,688 successfully retargeted benign apps, resulting in an obfuscation rate of 41% for benign apps. Similarly, ProGuard obfuscated 1,021 malicious apps out of 2,005 retargeted malicious apps; hence, ProGuard's obfuscation rate on malicious apps is 51%.

Table 7 shows that DashO has the lowest obfuscation rate (30%) whereas DroidChameleon achieves the highest obfuscation rate (60%). There are many reasons behind these low obfuscation rates, including exceptions raised by obfuscation tools while transforming an app and their inability to produce a valid obfuscated *classes.dex* file. For instance, Allatori raised this exception "*com.allatori.IiIIIIiiii: Only final fields may have an initial value!*" on many apps. We contacted the provider of Allatori about this exception, who informed us that this problem has been reported by other users, but could not be reproduced. Consequently, we helped them reproduce it to improve their product. They reported to us that this exception is mainly caused by the use of *dex2jar*, although a fix for the exception is still in progress.

Table 7: The ability of obfuscators to generate valid APKs.

	ProGuard	Allatori	DroidChameleon	DashO	ADAM
Benign	40.52%	25.77%	81.57%	13.39%	79.57%
Malicious	50.92%	58.70%	56.10%	43.24%	59.87%
Total	46.17%	43.65%	59.95%	29.60%	69.72%

5.4.2 Installable and runnable apps. To measure an obfuscation tool's ability to generate installable apps, we identified all original apps that have at least one app transformed by each obfuscation tool. For each original app, if there is more than one app transformed using the same obfuscation tool, we randomly select one of them. Using that process, we randomly selected 250 original apps along with their obfuscated versions, resulting in the selection of 1,750 obfuscated apps. We ran this experiment on a MacBook Pro with a 2.2 GHz Intel Core i7 and 16GB RAM, and installed the apps on an Android device. After we confirmed that all 250 original apps were successfully installed on the Android device, we installed the obfuscated apps.

In addition to measuring app *installability after obfuscation*, i.e., the extent to which an obfuscator can generate installable apps, we further measured app *runnability after obfuscation*, i.e., the extent to which an obfuscator can generate runnable apps. For our study, a runnable app can be order-agnostic or order-aware. A runnable app is *order-agnostic* if its obfuscated version exhibits the same *set* of running components and exceptions as its original version; a runnable app is *order-aware* if its obfuscated version exhibits the same *sequence* of running components and exceptions as its original version. To determine app runnability after obfuscation, we recorded the sequence of (1) components that execute and (2) exceptions that occur during execution of an app using *Monkey* [24], a program that generates pseudo-random streams of user events (e.g., clicks, touches, or gestures) and system-level events. We then checked for equality of the sequences of running components and exceptions of an original app and its obfuscated version, using 1,000 events for each app as input. We further ran Monkey using the same random seed for each original app and its obfuscated version, in order to test both app versions using the same sequence of inputs.

To conduct this experiment, we used the 250 original apps and the 1,341 successfully installed apps from the previous experiment, i.e., the total installed apps mentioned in Table 8. To measure whether an obfuscated app runs successfully, we have modified and instrumented the Android framework [13] to include probes for monitoring the running components. We installed our modified Android framework on a Nexus 5X device.

Table 8 shows the ability of obfuscation tools to produce installable and runnable apps, including the following information: the

total number of obfuscated apps that we *Examined* per obfuscation tool; the number of successfully *Installed* apps; the number of runnable apps that are *Order-Agnostic*; and the number of runnable apps that are *Order-Aware*.

Table 8: Installable and runnable apps of each obfuscator.

Obfuscator	Examined	Installed	Order-Agnostic	Order-Aware
Jarsigner	250	249	248	150
Apktool	250	249	246	154
DroidChameleon	250	249	83	31
ProGuard	250	248	237	131
Allatori	250	213	188	122
ADAM	250	84	67	46
DashO	250	49	0	0
Total Apps	**1,750**	**1,341**	**1,069**	**634**

Many obfuscation tools produce installable apps. Our results demonstrate that almost all apps transformed by Apktool/Jarsigner, DroidChameleon, and ProGuard have successfully installed. In addition, only 37 apps obfuscated by Allatori have not installed successfully. Moreover, Table 8 shows that most apps obfuscated using ADAM or DashO are not installable. Successfully installed apps obfuscated using ADAM all utilize the ALIGN transformation. All obfuscated apps using the non-trivial obfuscations of ADAM are not installable.

The runnability of apps obfuscated using our studied obfuscation tools varies greatly depending on the tool. Table 8 shows that almost all obfuscated apps using *Jarsigner* and *Apktool* are runnable in an order-agnostic fashion. 249 apps obfuscated using DroidChameleon are installable; only 83 of those installable apps are order-agnostic and runnable; and only 31 of those installable apps are order-aware and runnable. All apps transformed by DroidChameleon using the ENC transformation are missing a function that decrypts encrypted strings, causing these apps to crash at runtime and raise the error `java.lang.NoClassDefFoundError`. All apps that become unrunnable after transformation by DashO raise the same error, i.e., `java.lang.ExceptionInInitializerError`. Given that DashO is not an open-source tool, we could not investigate this problem further. Table 8 shows that 95% of the installable apps generated by ProGuard are runnable. Likewise, 88% of the installable apps generated by Allatori are runnable.

Finding 9: The percentage of obfuscated apps that are both installable and runnable in an order-aware fashion with respect to component behaviors varies from 0%-62%. These results suggest a significant need for improving obfuscation tools so that applying their transformations retain an app's original behavior.

6 DISCUSSION

For anti-malware product vendors, our study suggests several areas for which anti-malware products in general can be significantly improved. Recall that overall on our obfuscated dataset, anti-malware products experienced a 20% decrease in their detection rate of malicious apps compared to the original dataset (Finding 1). In particular, transformations that mainly involved identifier-name manipulation (i.e., MAN, ENC_IDR, and CF_ENC_IDR) substantially affected obfuscation tools (Findings 3 and 5). Manifest-file transformations (i.e., MAN) that simply involve addition of permissions that are not necessarily used or fake component capabilities resulted in a 28% decrease, on average, for the top-performing anti-malware products

Figure 6: Anti-malware detection, installability, and runnability with respect to obfuscators

(Finding 3). Overall, these results indicate that anti-malware product vendors would significantly benefit from performing a deeper analysis into the code of an app, potentially focusing on the security-sensitive code used by apps through static or dynamic analysis. The importance of performing deeper analysis is further highlighted by (1) the fact that transformations need not necessarily be combined to evade anti-malware products (Finding 4) and (2) this evasion worsens for newer apps (Finding 8).

For benign-app developers, our study provides some guidance as to how particular obfuscations may be used. Specifically, we have found that reflection transformations tend to increase significantly the possibility of a benign app being labeled as malicious. Therefore, benign app developers may wish to avoid such transformations to avoid this false labeling, and to reduce overhead exhibited by reflection. In the general case, benign-app developers need not be overly concerned about their apps being falsely labeled as malicious when combining obfuscations (Finding 4), except in the case of reflection transformations (Finding 2).

The major finding for obfuscation-tool developers is that our study indicates that many of their transformations result in invalid, non-installable, or unrunnable apps (Finding 9). Although some of that burden lies on benign-app developers to ensure that obfuscations they apply do not adversely affect their apps, obfuscation-tool developers would benefit from aiding benign-app developers in the task of ensuring their apps remain runnable after obfuscation.

We further examine the implications of the interaction between (1) obfuscations tools' ability to produce installable and runnable apps and (2) the anti-malware product detection rate on malicious apps obfuscated using those tools. Figure 6 visually depicts the interaction between these two phenomena. Obfuscation tools that lie on the upper-right corner of the figure are preferred tools for benign app developers, obfuscation-tool providers, and anti-malware vendors. These obfuscation tools reliably generate installable and runnable apps while maintaining a high detection rate of malicious apps for anti-malware products. In our study, no tool is above 80% for both its anti-malware detection rate and installability and runnability after obfuscation. Consequently, significant improvements can be made to these tools along those dimensions.

Obfuscation tools that lie on the upper-left corner of Figure 6 are tools that exhibit properties particularly useful for malware authors. These tools reliably generate installable and runnable apps while evading the detection of anti-malware products. ProGuard is an example of such a tool. Although few tools appear close to the upper-left corner of the chart, malware authors are likely to expend

the extra effort needed to ensure that their obfuscations result in installable and runnable apps.

7 THREATS TO VALIDITY

This section presents our study's threats to external and construct validity, and the actions we have taken to mitigate them.

External validity measures the extent to which the results of our study can be generalized. One threat to external validity for our study is whether our study's findings can be generalized to other apps outside of our study. To mitigate this threat, we obtained benign and malicious apps from diverse sources that vary across application domains, in terms of app size, and originate from various time periods.

To ensure our findings are likely to generalize to other obfuscation strategies and tools, we employed 29 obfuscation strategies from 7 obfuscation tools—the largest number of strategies and tools utilized to date for a study about app obfuscation. We further obtained obfuscation tools that are academic, open-source, and commercial—aiding in generalizability to these three different sources.

Another threat to external validity is our selection of anti-malware products. To mitigate this threat, we have selected over 60 anti-malware products from VirusTotal, and focused on the most popular and well-rated 21 products for our study, due to space limitations. However, we make the results of our study, along with the complete list of anti-malware products and apps, available online [21]. The findings for the anti-malware products not discussed in this paper are consistent with the findings in this paper.

Construct validity is concerned with whether our study's measurements or measurement procedures validly quantify the constructs or concepts we intend to quantify. A threat to construct validity is the metrics and measurement procedures we used to quantify the ability of obfuscation tools to produce runnable apps whose behavior before obfuscation is similar to behavior after obfuscation. To measure these constructs, we compared the set or sequence of running components and thrown exceptions of apps before and after obfuscation. These measurement procedures and associated metrics are sensible given that components are functional units of behavior—making them a sensible means of identifying high-level behavior—and exceptions are the main means of identifying errors in apps whose test cases and oracles are unavailable, which is the case for many apps on Google Play.

Another threat to construct validity is the labeling of our apps as benign or malicious. To mitigate this threat, our dataset of benign apps are marked as benign by over 50 anti-malware products. Similarly, our malicious apps are obtained from repositories containing apps manually labeled as malicious by security experts.

8 RELATED WORK

We divide previous work related to our study into four categories: (1) studies about similarity of a repackaged app with its original version, (2) obfuscation strategies for PC and desktop software, (3) obfuscation tools specifically designed for Android, and (4) studies about the effects of obfuscation on anti-malware products. In the remainder of this section, we discuss each of these areas of related work and conclude the section with the key differences between our study and the most similar related work.

Researchers have studied the similarity between original apps and repackaged apps. Huang et al. [34] used Androguard [2], an Android reverse-engineering framework for malware analysis, to study the obfuscation resilience of repackaging detection algorithms. Faruki et al. [32] compared the performance of anti-malware products and Androguard's code similarity with AndroSimilar [33], their tool for detecting obfuscated apps. Wang and Rountev describe an approach for determining which obfuscation tool was applied to an obfuscated app [46].

A few studies have considered the application of obfuscation strategies in the context of PC and desktop software. Collberg et al. produced a taxonomy of transformations for obfuscation with a focus on Java [31]. Collberg et al. [30] also implemented a tool called SandMark for evaluating the effectiveness of code obfuscation to protect Java-based software systems from piracy, tampering, and reverse engineering. Christodorescu and Jha [28] evaluated the resilience of anti-malware products against code obfuscation applied to Visual Basic programs and proposed a semantics-aware malware-detection algorithm in [29].

Previous work has produced a few obfuscation tools specifically designed for Android apps. Zheng et al. [47] proposed ADAM, a framework for obfuscating Android apps and testing them on anti-malware products. They evaluated ADAM's effectiveness for evading 10 anti-malware products on 222 transformed, malicious apps. Rastogi et al. [44] presented DroidChameleon, a tool for obfuscating Android apps, and assessed obfuscations on six apps from Android Malware Genome [49].

Another set of studies focused on the effects of obfuscations on anti-malware products, without proposing new obfuscation tools. Maiorca et al. [40] studied the effects of code obfuscated by a single tool on 13 anti-malware products. Pomilia [43] studied the performance of 9 anti-malware products on a dataset obfuscated using Allatori. Morales et al. [41] studied the resilience of 4 anti-malware products after transforming 2 viruses on Windows Mobile OS.

All the aforementioned previous work that have studied either obfuscation tools or the effects of obfuscation strategies on anti-malware products focus on a single obfuscation tool and a small number of anti-malware products and apps. None of these studies have performed a large-scale empirical study considering the effects that occur due to the concurrent utilization of anti-malware products, various obfuscation tools, and their supported obfuscation strategies. None of these studies assessed these effects on benign apps; the effects of combining obfuscation strategies; and the ability of obfuscation tools to produce valid, installable, and runnable apps.

9 CONCLUSION

In this paper, we have evaluated the effectiveness of the top-rated Android anti-malware products against code obfuscation. We used a large dataset consisting of 3,000 benign apps, 3,000 malicious apps, and 73,362 obfuscated apps. To obfuscate Android apps, we applied 29 different obfuscation strategies using 7 commercial, open-source, and academic obfuscation tools. Our study includes the following key findings: (1) all anti-malware products succumb to code obfuscation; (2) most products are susceptible to trivial transformations, such as simple changes to the Android manifest file; (3) the detection rates of anti-malware products depend not only on the obfuscation strategy applied but also on the leveraged obfuscation tool; (4) anti-malware products are slow to adopt signatures of malicious apps; and (5) code obfuscation to a significant extent results in apps that do not exhibit the same behavior as their original versions. Based on our overall findings, we have provided guidance and recommendations for improving anti-malware products and obfuscation tools. To that end, Our study results, including the framework that we developed to conduct this study, are available publicly online [21].

10 ACKNOWLEDGMENTS

This work was supported in part by awards CCF-1252644, CNS-1629771, and CCF-1618132 from the National Science Foundation, HSHQDC-14-C-B0040 from the Department of Homeland Security, and FA95501610030 from the Air Force Office of Scientific Research.

REFERENCES

[1] Apktool. https://ibotpeaches.github.io/Apktool/. (2010).
[2] Androguard: Reverse engineering and malware analysis of Android apps by BlackHat. https://github.com/androguard. (2011).
[3] Allatori Obfuscator. http://www.allatori.com/. (January 2012).
[4] VirusTotal-Free virus, malware and URL scanner. https://www.virustotal.com/en. (2012).
[5] VirusShare. http://virusshare.com/. (August 2013).
[6] Contagio Malware Repository. http://contagiodump.blogspot.it. (2015).
[7] Brain Test Lookout Report. https://blog.lookout.com/blog/2016/01/06/brain-test-re-emerges/. (2016).
[8] DressCode Android malware. http://blog.checkpoint.com/2016/08/31/dresscode-android-malware-discovered-on-google-play/. (2016).
[9] Kaspersky Security Bulletin. https://kasperskycontenthub.com/securelist/files/2016/12/Kaspersky_Security_Bulletin_2016_Review_ENG.pdf. (2016).
[10] McAfee mobile threats report. https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf. (2016).
[11] Smartphone OS Market Share, 2017 Q1. International Data Corporation. http://www.idc.com/promo/smartphone-market-share/os. (2016).
[12] VikingHorde Android malware. http://blog.checkpoint.com/2016/05/09/viking-horde-a-new-type-of-android-malware-on-google-play/. (2016).
[13] Android Open Source Project. https://source.android.com/. (July 2017).
[14] Android Studio. https://developer.android.com/studio/build/shrink-code.html. (2017).
[15] DashO. https://www.preemptive.com/. (2017).
[16] Dex2jar: Tools to work with android. dex and java. class files. https://github.com/pxb1988/dex2jar. (2017).
[17] DexGuard. https://www.guardsquare.com/en. (2017).
[18] FalseGuide Android malware. http://blog.checkpoint.com/2017/04/24/falseguide-misleads-users-googleplay/. (2017).
[19] Google Play App Store. https://play.google.com/store?hl=en. (2017).
[20] jarsigner - JAR Signing and Verification Tool. https://docs.oracle.com/javase/6/docs/technotes/tools/windows/jarsigner.html. (2017).
[21] Obfuscation Study Framework. http://www.ics.uci.edu/~seal/projects/obfuscation/index.html. (August 2017).
[22] ProGuard. https://www.guardsquare.com/en/proguard. (2017).
[23] Smali/Backsmali. https://github.com/JesusFreke/smali. (2017).
[24] UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html. (August 2017).
[25] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Are your training datasets yet relevant. In *International Symposium on Engineering Secure Software and Systems*. Springer, Milan, Italy, 51–67.
[26] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, Austin, Texas, 468–471.
[27] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceeding of Network and Distributed System Security Symposium*. San Diego, California.
[28] Mihai Christodorescu and Somesh Jha. 2004. Testing malware detectors. *International Symposium on Software Testing and Analysis (ISSTA'04)* (July 2004).
[29] Mihai Christodorescu, Somesh Jha, Sanjit A Seshia, Dawn Song, and Randal E Bryant. 2005. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*. IEEE, Oakland, CA, 32–46.
[30] Christian Collberg, GR Myles, and Andrew Huntwork. 2003. Sandmark-a tool for software protection research. *IEEE security & privacy* 99, 4 (2003), 40–49.
[31] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. *A taxonomy of obfuscating transformations.* Technical Report TR148. Department of Computer Science, The University of Auckland, New Zealand.
[32] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2014. Evaluation of android anti-malware techniques against dalvik bytecode obfuscation. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, Beijing, China, 414–421.
[33] Parvez Faruki, Vijay Laxmi, Ammar Bharmal, Manoj Singh Gaur, and Vijay Ganmoor. 2015. AndroSimilar: Robust signature for detecting variants of Android malware. *Journal of Information Security and Applications* 22 (June 2015), 66–80.
[34] Heqing Huang, Sencun Zhu, Peng Liu, and Dinghao Wu. 2013. A framework for evaluating mobile app repackaging detection algorithms. In *International Conference on Trust and Trustworthy Computing*. Springer, London, UK, 169–186.
[35] Li Li, Tegawendé François D Assise Bissyande, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Yves Le Traon. 2016. *Static analysis of android apps: A systematic literature review.* Technical Report. SnT.
[36] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Haipeng Cai, David Lo, and Yves Le Traon. 2017. Automatically locating malicious packages in piggybacked android apps. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17)*. IEEE Press, Buenos Aires, Argentina, 170–174.
[37] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security* 12, 6 (June 2017), 1269–1284.
[38] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding Android app piggybacking. In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, Buenos Aires, Argentina, 359–361.
[39] Federico Maggi, Andrea Valdi, and Stefano Zanero. 2013. AndroTotal: a flexible, scalable toolbox and service for testing mobile malware detectors. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*. ACM, Berlin, Germany, 49–54.
[40] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. 2015. Stealth attacks: An extended insight into the obfuscation effects on android malware. *Computers & Security* 51 (March 2015), 16–31.
[41] Jose Andre Morales, Peter J Clarke, Yi Deng, and BM Golam Kibria. 2006. Testing and evaluating virus detectors for handheld devices. *Journal in Computer Virology* 2, 2 (2006), 135–147.
[42] Damien Octeau, Somesh Jha, and Patrick McDaniel. 2012. Retargeting Android applications to Java bytecode. In *International Symposium on the Foundations of Software Engineering*. ACM, Cary, North Carolina, 6.
[43] M. Pomilia. A Study on Obfuscation Techniques for Android Malware. (2016). http://www.dis.uniroma1.it/~midlab
[44] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2014. Catch me if you can: Evaluating android anti-malware against transformation attacks. *IEEE Transactions on Information Forensics and Security* 9, 1 (2014), 99–108.
[45] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 1999. Soot - a Java Bytecode Optimization Framework. In *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '99)*. IBM Press. http://dl.acm.org/citation.cfm?id=781995.782008
[46] Yan Wang and Atanas Rountev. 2017. Who Changed You? Obfuscator Identification for Android. (May 2017).
[47] Min Zheng, Patrick PC Lee, and John CS Lui. 2012. ADAM: an automatic and extensible platform to stress test android anti-virus systems. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, Heraklion, Crete, Greece, 82–101.
[48] Wu Zhou, Yajin Zhou, Michael Grace, Xuxian Jiang, and Shihong Zou. 2013. Fast, scalable detection of piggybacked mobile applications. In *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, San Antonio, TX, 185–196.
[49] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*. IEEE, San Francisco, California, 95–109.