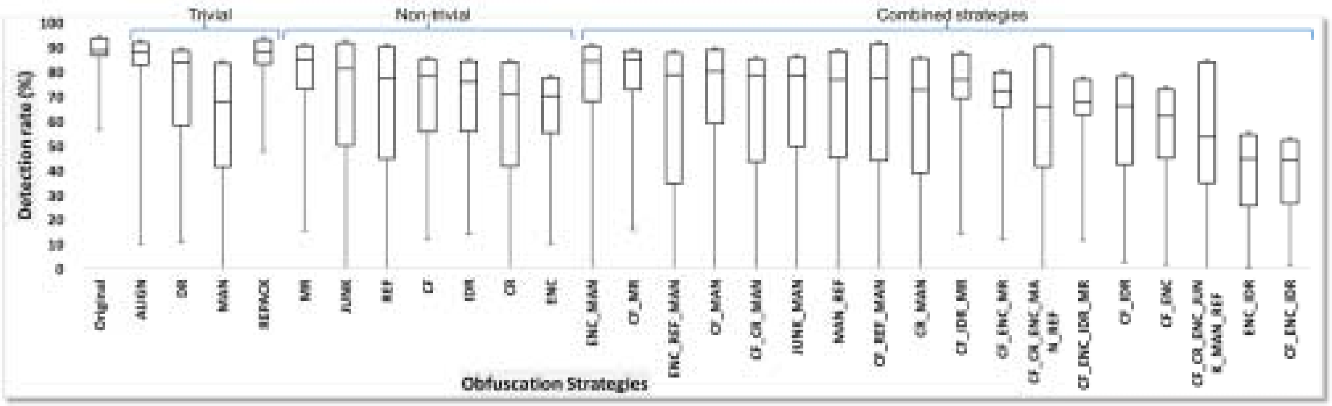

Figure 3: (RQ1) The average detection rate of all anti-malware products regarding each obfuscation strategy.

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.

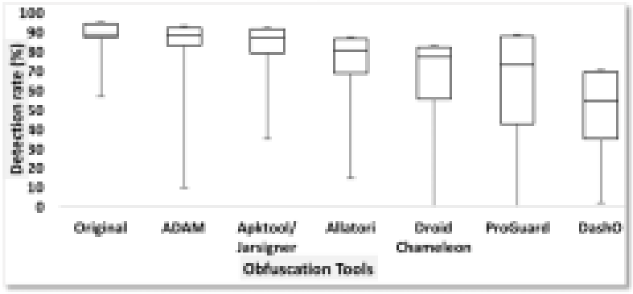

Figure 4: (RQ2) The average detection rate of all anti-malware products regarding each obfuscation tool.

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.

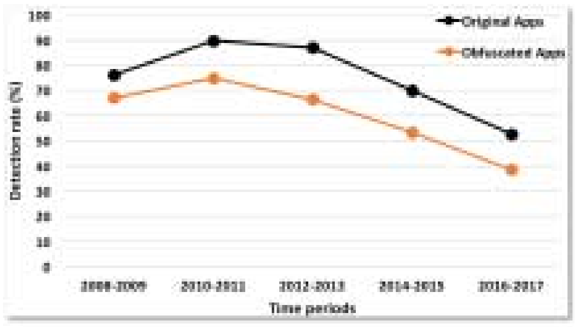

Figure 5: (RQ3) Time-aware analysis.

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.IiIIIiiii: 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

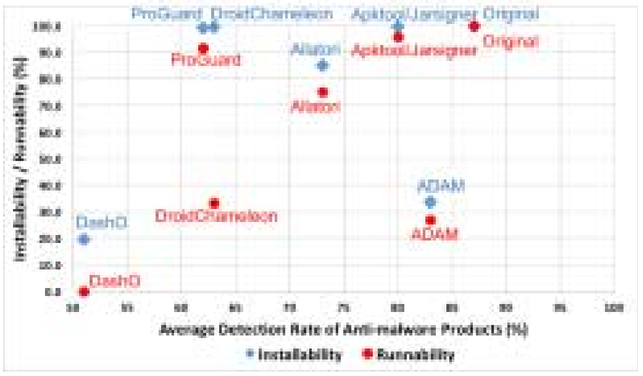

Figure 6: Anti-malware detection, installability, and runnability with respect to obfuscators

(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