Uncovering and Mitigating the Impact of Code Obfuscation on Dataset Annotation with Antivirus Engines

Cuiying Gao*
Huazhong University of Science and Technology
Wuhan, China
JD.com
Beijing, China
gaocy@hust.edu.cn

Yueming Wu*
Nanyang Technological University
Singapore, Singapore
yueming.wu@ntu.edu.sg

Heng Li
Huazhong University of Science and Technology
Wuhan, China
liheng@hust.edu.cn

Wei Yuan†
Huazhong University of Science and Technology
Wuhan, China
yuanwei@mail.hust.edu.cn

Haoyu Jiang
Huazhong University of Science and Technology
Wuhan, China
jhy123456@hust.edu.cn

Qidan He
JD.com
Beijing, China
i@flanker017.me

Yang Liu
Nanyang Technological University
Singapore, Singapore
yangliu@ntu.edu.sg

Abstract

With the widespread application of machine learning-based Android malware detection methods, building a high-quality dataset has become increasingly important. Existing large-scale datasets are mostly annotated with VirusTotal by aggregating the decisions of antivirus engines, and most of them indiscriminately accept the decisions of all engines. In reality, however, these engines have different capabilities in detecting malware, especially those that have been obfuscated. Previous research has revealed that code obfuscation degrades the detection performance of these engines to varying degrees. This makes us believe that using all engines indiscriminately is unreasonable for dataset annotation. Therefore, in this paper, we first conduct a data-driven evaluation to confirm the negative effects of code obfuscation on engine-based dataset annotation. To gain a deeper understanding of the reasons behind this phenomenon, we evaluate the availability, effectiveness and robustness of every engine under various code obfuscation techniques. Then we categorize the engines and select a set of obfuscation-robust engines. Finally, we conduct comprehensive experiments to verify the effectiveness of the selected engines for dataset annotation. Our experiments show that when 50% obfuscated samples are mixed into the training set, on the classic malware detectors Drebin and Malscan, using our selected engines can effectively improve detection performance by 15.21% and 19.23%, respectively, compared to using all the engines.

CCS Concepts

• **Security and privacy → Malware and its mitigation**.

Keywords

Code obfuscation, Malware detection, Antivirus engines

1 Introduction

In recent years, machine learning [29] has been extensively utilized to design Android malware detectors [12] [27] [13] [44], which spurs an urgent demand for high-quality annotated datasets. As an online antivirus scanning service platform, VirusTotal [7] integrates over 70 antivirus engines and has become the preferred choice for dataset annotation [40] [43] [11]. To annotate a dataset, a user uploads APKs to VirusTotal to obtain their final labels by aggregating the detection decisions predicted by various engines. Label aggregation is usually based on a predetermined threshold [46][42]. If the number of engines detecting the sample to be malicious, i.e., ***Virus**Total **P**ositives* (*VTP*), is greater than or equal to the threshold, the sample is labeled as malicious. Currently, VTP has become an important consideration in determining sample labels [24] [31]. For example, when we download samples from AndroZoo [9], the largest collection of Android apps, the VTP of each sample will also be provided as a reference for annotation. Under this situation, the discriminability of VTP between benign and malicious samples is of great importance to dataset annotation.

Despite being used for several years, some skepticism remains regarding the dataset annotation method with antivirus engines [36] [37]. A major concern is that the performance of each engine varies [14], especially in terms of their sensitivity to *code obfuscation* [20] [34] [16]. Code obfuscation aims at increasing the difficulty of reverse engineering, while preserving the original functionality of Android software [15]. At present, code obfuscation has been widely used by both developers and attackers [17] [28] [32]. Developers employ code obfuscation to safeguard their intellectual property [17], whereas attackers leverage code obfuscation to shield their malicious code and evade detection [10].

At present, some researchers have proposed using only the high-reputation engines [46] for dataset annotation, hoping to achieve

*Equal contribution
†Corresponding authors

ISSTA '24, September 16–20, 2024, Vienna, Austria

ACM ISBN 979-8-4007-0612-7/24/09
https://doi.org/10.1145/3650212.3680302

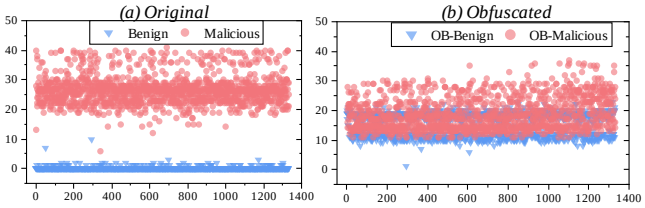

Figure 1: Visualizing the benign-malicious separability of VTP before and after obfuscation. *The horizontal axis represents sample index, and the vertical axis represents VTP.*

better detection results. Unfortunately, we observed that some high-reputation engines still reverse their labels when faced with obfuscated samples. This problem should be avoided, since code obfuscation preserves software functionality and therefore should not alter sample labels. Moreover, we realized that code obfuscation usually results in the variation of VTP, which can further reduce the discriminability of VTP between benign and malicious samples and impair the engine-based dataset annotation.

To illustrate this issue clearly, we visualize the separability of VTP in Figure 1. Figure 1 (a) demonstrates the original VTP values of benign and malicious samples. It can be seen that VTP exhibits excellent separability between benign and malicious samples. However, after the samples are obfuscated by encrypting constant strings, the separability of VTP is significantly weakened, as shown in Figure 1 (b). This indicates in the presence of code obfuscation, the engines-based dataset annotation method may introduce label noise and undermine downstream tasks.

Therefore, it is essential to carefully consider how to effectively utilize antivirus engines for dataset annotation under code obfuscation. Following the process of *'identifying problems, understanding causes, and proposing solutions'*, we investigate four issues: 1) How does code obfuscation affect VTP and further influence the entire dataset annotation process? 2) How do obfuscated and relabeled samples in the training set impair the performance of different malware detectors? 3) How does code obfuscation impact each engine and what are the differences between engines in resisting code obfuscation? 4) How to appropriately select engines for sample labeling to enhance the accuracy of dataset annotation?

Our research starts with constructing datasets for evaluating the impact of code obfuscation on VTP. We first construct an original dataset that is not only annotated by VirusTotal. We then generate obfuscated datasets by applying various obfuscations to the original dataset. Finally, we collected 131,894 valid reports. Aided by these reports, we conducted the following four studies.

(1) **Measuring the VTP variations caused by obfuscation.** We first evaluate the VTP variations of samples before and after different obfuscations. Our experiments reveal that obfuscation tends to increase VTP for benign samples and decrease VTP for malicious samples. Moreover, different types of obfuscating transformations have varying impacts on VTP.

(2) **Investigating the impacts of VTP variations.** We then construct various training sets with different proportions of obfuscated samples, which are re-labeled using VirusTotal. The training sets are used to train the Drebin and Malscan detectors. Our experiments show that as the proportion of obfuscated samples increases, the performance of each detector decreases to varying degrees.

(3) **Understanding the causes for VTP variations.** To identify the causes of VTP variations, we conduct a fine-grained experimental analysis for each engine, and clarify the differences between various engines from the following three aspects. *1) Availability*: Engines may have varying probabilities of producing available results. *2) Effectiveness*: Engines possess different capabilities in detecting malware from the original dataset. *3) Robustness*: Engines have varying degrees of sensitivity to code obfuscation techniques. Based on the above analysis, we find the causes for VTP variations before and after obfuscation. Furthermore, we observe that more than 20 engines have very low availability, with less than 1% probability of producing effective detection results, and 13 engines label all samples as benign with over 95% probability. Unfortunately, previous studies paid little attention to these two types of engines.

(4) **Improving the quality of dataset annotation.** Based on the understanding of various engines, we propose to categorize engines and appropriately select engines for dataset annotation. By using these selected engines (i.e., robust engine sets) to annotate the training set containing obfuscated samples, we can improve the performance of learning-based malware detectors by about 20%, which is better than using the high-reputation engines.

As an extension to the above studies, we also investigate how different obfuscation strategies affect the string labels output by various engines.

Finally, our contributions are summarized below.

1) **Finding**. Through extensive experiments, we analyze the impact of different code obfuscation techniques on VTP, and confirm the harm to learning-based malware detectors caused by considering all engines equally in dataset annotation.

2) **Insights**. We quantitatively analyze each engine in three aspects: availability, effectiveness and robustness, which reveals the reason for the VTP variations before and after obfuscation from different perspectives.

3) **Method**. We divide the engines into different sets based on their capabilities. Then we select the robust engine sets for dataset annotation to enhance the performance of learning-based malware detectors, surpassing the high-reputation engine sets.

4) **Dataset and code**. Since obtaining VirusTotal reports of samples is a time-consuming and labor-intensive task, we make our dataset, 131,894 reports and code available to facilitate future research.

2 Preliminary

2.1 Dataset Annotation with Antivirus Engines

In this subsection, we introduce the main steps involved in constructing datasets aided by Antivirus Engines on VirusTotal, *i.e.,* *Sample Collection*, *Report Obtaining*, and *Label Aggregation*.

- **Sample Collection:** In the first step, an adequate number of *Android packages* (APKs) are downloaded from diverse sources such as Google Play Store [5], Androzoo [9] and VirusShare [6].
- **Report Obtaining:** The collected APKs are uploaded onto VirusTotal. VirusTotal will provide an analysis report for each sample, which includes the detection result of every engine, e.g., benign, malicious, or unavailable (*i.e.,* failure).
- **Label Aggregation:** The last step involves determining the label of a sample by aggregating the outputs of engines. The commonly

used approach is threshold-based aggregation [46]. If the VTP reaches or exceeds the predetermined threshold T, the input sample will be labeled as malicious.

2.2 Code Obfuscation

Code obfuscation is a technique that uses code transformation technology to convert the original program, making it functionally equivalent but difficult to comprehend and analyze [15] [17]. Suppose the original program is denoted as P, and the transformed program is denoted as P'. We have $P' \triangleq T(P)$, where T represents the code transformation technique [15]. Code transformation has various forms, e.g., renaming methods, encrypting strings, and so on. As a standard operation, code obfuscation has been widely used in Android app development [2] [3] to offer protection against reverse engineering and enhance software security [33]. Many popular development tools, such as Android Studio, include built-in code obfuscation features that developers can easily enable. Additionally, there are many code obfuscation tools [3] [1] [2]. However, code obfuscation can also be utilized by attackers. More specifically, attackers can employ obfuscation techniques to circumvent anti-malware engines, hence making Android malware more challenging to detect and analyze.

3 Setup

In this section, we introduce the setup of our study, covering the issues of dataset construction, detection report obtaining, and research questions.

3.1 Dataset Construction

Building appropriate datasets is a fundamental prerequisite for our study. As mentioned before, we need to use two kinds of datasets in our evaluation, the original dataset and the obfuscated ones.

Original Dataset. Constructing the original dataset should meet three basic requirements: 1) the dataset should include a large number of samples; 2) the sample labels should be trustworthy; and 3) the dataset cannot solely rely on antivirus engines from VirusTotal for annotation. According to these requirements, we select the CICMalDroid dataset [26] [25] as our original dataset. As shown in Table 1, CICMalDroid contains 12,538 samples, each of which is analyzed by static analysis, dynamic analysis [38] and network traffic analysis. Based on the analysis results and logs, four categories of malware (*i.e.,* Adware, Banking, SMS, and Riskware) and one category of benign samples, are manually identified.

Table 1: The brief description of the original dataset.

Category	Adware	Banking	Riskware	SMS	Benign	Total
#Sample	1,514	2,506	2,060	2,461	4,042	12,538

Obfuscated Dataset. We then build the obfuscated datasets based on the original dataset. The reason why we decided to build our own obfuscated databases is that the reconstructed datasets can more accurately match the original dataset, helping us to observe the change in VTP for each sample.

For a sample P from the original dataset, we apply code transformation T to generate P'. Since P and P' have equivalent functionality, they should have the same label. We employ the automated obfuscation tool ObfuscAPK [10], an open-source tool supporting multiple advanced code transformation strategies, to implement T. As shown in Table 2, we utilize 11 different code obfuscation transformations to generate obfuscated samples. These code obfuscation transformations come from six popular categories, including trivial, rename, encryption, reflection, and so on. In summary, an original sample P can generate 11 different obfuscated samples P'.

In our experiments, an obfuscated sample can be benign or malicious (*i.e.,* Adware, Banking, SMS, and Riskware). It is noted that the number of generated obfuscated samples (*i.e.,* the set of P') is not constant for all original samples. This is because some code transformations (e.g., *CID*) are more complicated, and may fail to generate obfuscated samples occasionally. Fortunately, most of the samples in our dataset can successfully produce the corresponding obfuscated samples. Finally, we obtained 137,631 obfuscated samples, in addition to 12,538 samples from the original dataset. That is, the total number of all samples is 150,169.

3.2 Getting Analysis Reports

With the obfuscated samples, we obtain their analysis reports from VirusTotal following two steps. First, we upload the obtained samples onto VirusTotal using the `upload` API. Then, we calculate the sha256 of each sample and obtain the analysis report using the `scan` API. The last column of Table 2 shows the number of reports obtained from VirusTotal. Due to the size limit for each uploaded sample (i.e., the maximum allowed file size is 32MB), 5,737 samples are not successfully uploaded. In the end, we collect 131,894 analysis reports for both original and obfuscated samples.

Table 2: Obfuscation introduced and obfuscated samples.

Ob-type	T	Description	#Report
Trivial	RDM	Reordering entries in the Manifest file.	11,266
Rename	MR	Renaming method names to meaningless strings.	11,455
	FR	Renaming fields in a class to meaningless strings.	10,769
	CR	Renaming class names to meaningless strings.	10,707
Encryption	CSE	Encrypting constant strings in the code.	11,478
Junk code	AB	Inserting dead code that will not be executed using arithmetic constraints.	11,136
	NOP	Inserting Nop instructions.	11,223
Code	ROR	Changing the structure of code blocks by altering conditional branches or inserting goto instructions.	10,666
	CID	Adding intermediate calls to alter the original function call relationships without changing functionality.	9,411
Reflection	REF	Performing reflection calls on APIs.	10,954
	AREF	Performing reflection calls on dangerous APIs.	10,547
Original samples			12,282
Total number of original and obfuscated samples			131,894

3.3 Research Questions

In this work, we aim to answer the following research questions:

- **RQ1:** *How do obfuscation techniques affect the VTP of a sample?* To answer this question, we first analyze the VTP distribution of original samples, and then examine the VTP changes before and after obfuscation.
- **RQ2:** *How does the inclusion of obfuscated and relabeled samples in the training set impact the performance of learning-based detectors?* To answer this, we replace different proportions of original samples with obfuscated ones, creating a new training set. Then we train two standard detectors on this set to evaluate the impact on their performance.
- **RQ3:** *Which engines are the culprits for VTP variations?* By exploring the availability, effectiveness, and robustness of each engine, we attempt to identify the causes of the changes in VTP before and after obfuscation.
- **RQ4:** *How to effectively utilize engines for dataset annotation?* To address this, we categorize engines into different sets based on their detection effectiveness and obfuscation resistance. We then label the training set using these sets, train several malware detectors, and evaluate them on a unified test set.
- **RQ5:** *How do obfuscation techniques affect the string label output by engines?* To answer this question, we explore the effects of code obfuscation on the consistency of engines' string labels, and then summarize our key findings.

4 RQ1: Impact of Obfuscation on VTP

4.1 Goal and Setup

In this section, we aim to investigate the impact of code obfuscation on the VTP of samples coming from different categories. Here we consider five categories of the original dataset, i.e., Benign, Adware, Banking, SMS, and Riskware. We first analyze the distribution of VTP of samples in the original dataset. We then analyze the VTP variations of the samples in the corresponding obfuscated datasets. For a pair of original and obfuscated samples, we use VTP_{var} to represent the VTP variation, *i.e.,* $VTP_{var} = VTP_{ori} - VTP_{obf}$, where VTP_{ori} is the VTP of the original sample and VTP_{obf} is that of the obfuscated sample.

To further understand how different types of code obfuscation techniques affect the VTP, we analyze the VTP_{var} of samples from five categories under various transformations.

4.2 Result and Analysis

Figure 2 (a) shows the VTP_{var} of obfuscated samples from the categories of *Benign*, and Figure 2 (b) shows it of category *Banking*[1]. We first consider Figure 2 (a). The VTP of benign samples exhibits an increasing trend after the samples are obfuscated. Over 95% of the obfuscated samples have a VTP_{var} greater than 0. If the threshold of label aggregation is set to 1, more than 95% of obfuscated benign samples will be falsely labeled as malicious, as shown in Figure 2 (b). That is, a significant number of mislabelled samples will be included in datasets. In contrast, for the malicious samples, as shown in Figure 2 (b), a decreasing trend in VTP is observed, with most samples having negative VTP_{var}. These results imply that benign samples modified by obfuscation are more likely to be misclassified as malicious by many engines, and malicious samples modified by obfuscation are more likely to be misclassified as benign by many engines. That is, the separability of VTP is weakened by code obfuscation.

In addition, it is observed that the VTP changing trend is similar for different types of malicious samples under the same obfuscation technique. However, the VTP changing trend differs for the samples generated by different obfuscating transformations. In comparison, the obfuscation technique CSE has a more pronounced impact on VTP. This phenomenon is particularly evident on the benign samples and malicious samples from the Banking category. Specifically, the VTP_{var} of samples generated by CSE is larger than that of samples generated by other obfuscating transformations, followed by CID. As shown in Figure 2 (a), over 50% of samples obfuscated by CSE have a VTP_{var} greater than 15.

***Answer to RQ1**: Code obfuscation can impact the dataset annotation with engines. Specifically, obfuscated benign samples are more likely to be labeled as malicious, while obfuscated malicious samples are more likely to be labeled as benign. Furthermore, applying CSE and CID obfuscations to original samples will result in a more noticeable mislabeling of engines.*

5 RQ2: Impact of Obfuscation on Android Malware Detectors

In this section, we will explore the impact of code obfuscation on the performance of learning-based malware detectors by adding obfuscated samples into the training set in varying proportions. To clarify the concepts, we use the term "learning-based detectors" (or "detectors" for short) to specifically refer to malware detectors constructed by academic approaches. These detectors' design and evaluation process is usually public, allowing for replication and verification. We refer to the Antivirus Engines as "engines." These engines are often the products of commercial companies, and their internal mechanisms are kept confidential. They are provided to users as a black box.

5.1 Learning-based Detectors

In recent years, various types of supervised learning-based Android malware detectors have emerged. The basic steps in building these detectors include dataset construction, preprocessing, feature extraction, model training, and model testing. The quality of the training set is crucial for the final detection performance. In the previous sections, we described the process of constructing a dataset using engines and quantitatively analyzed the impact of code obfuscation on dataset annotation. Next, we will inject obfuscated samples in different proportions into the training set, and relabel them based on VTP. Accordingly, we will explore the effect of doing this on the performance of Android malware detectors.

5.2 Experimental Setup

In the following, we describe our experimental setup, including the evaluated detectors, datasets, and evaluation metrics.

- **Detectors:**

  1) *Drebin*: Drebin is a string-style feature based detector [18]. Drebin first extracts information, such as request permissions and hardware components, from the APK and encodes them into vectors. It then uses SVM to classify the feature vectors.

[1]Due to space limitations, the VTP_{var} of other categories are given in Figure 1 of supplementary material. The distributions of VTP_{var} of other categories are similar to that of the Banking category.

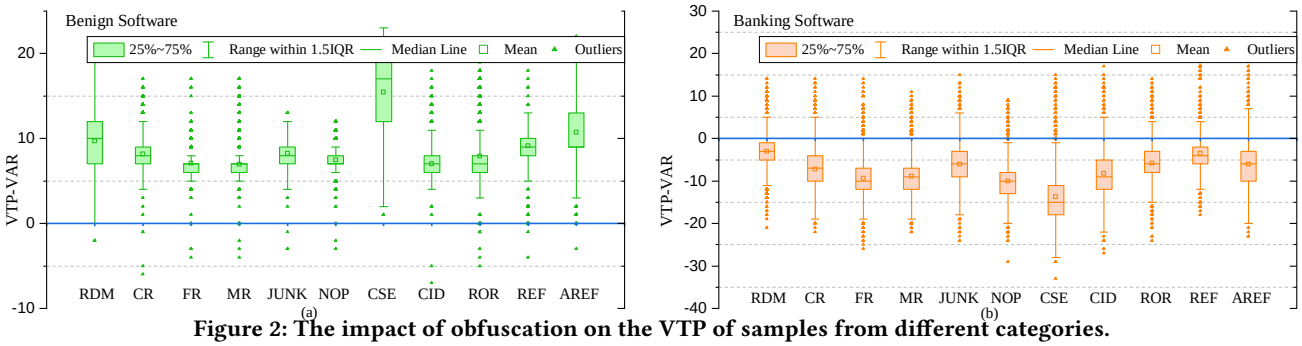

Figure 2: The impact of obfuscation on the VTP of samples from different categories.

Table 3: Performance of different malware detectors under training sets mixed with different proportions of obfuscated samples.

OBRat	CID								CSE							
	Drebin				Malscan				Drebin				Malscan			
	P	R	A	F1	P	R	A	F1	P	R	A	F1	P	R	A	F1
Original	0.982	0.982	0.982	0.982	0.966	0.965	0.965	0.965	0.982	0.982	0.982	0.982	0.966	0.965	0.965	0.965
10%	0.969	0.968	0.968	0.968	0.970	0.970	0.970	0.970	0.960	0.959	0.959	0.959	0.962	0.961	0.961	0.961
20%	0.954	0.952	0.952	0.952	0.966	0.966	0.966	0.966	0.937	0.931	0.931	0.931	0.952	0.951	0.951	0.951
30%	0.939	0.934	0.934	0.934	0.957	0.956	0.956	0.956	0.905	**0.889**	**0.889**	**0.888**	0.919	0.911	0.911	0.91
40%	**0.890**	**0.871**	**0.871**	**0.869**	0.925	0.920	0.920	0.920	**0.849**	**0.800**	**0.800**	**0.793**	**0.880**	**0.851**	**0.851**	**0.848**
50%	**0.872**	**0.843**	**0.843**	**0.840**	0.925	0.918	0.918	0.918	**0.823**	**0.741**	**0.741**	**0.724**	**0.854**	**0.800**	**0.800**	**0.792**

2) *Malscan*: Malscan is a graph-based detector. Malscan extracts function call graphs from APKs, selects 21,986 sensitive APIs, calculates their centrality to build the APK's features, and leverages a machine learning-based model for classification.

- **Dataset:** We select 8,084 samples from the CICMalDroid dataset to construct a balanced dataset for our experiments, consisting of an equal number of benign and malicious software samples. We divide the dataset into a training set and a test set at a ratio of 4:1. Then, we randomly choose 10%, 20%, 30%, 40%, and 50% of the training samples and replace them with obfuscated samples to create new training sets. For convenience, we will denote this ratio as ***OBRat***. These new training sets are then re-labeled using VirusTotal, to simulate real-world scenarios, where users collect obfuscated samples, label them using VTP, and add them into the training set. In our experiments, we use two obfuscation techniques, CID obfuscation and CSE obfuscation (detailed descriptions of these techniques are provided in Section 3). In RQ1, we find that these two obfuscation methods significantly affect the antivirus engines. We use the VTP as the standard to differentiate between benign and malicious software, with a VTP threshold of 1, to align with the existing work's settings [46]. This means that if at least one engine identifies a sample as malicious, then the sample is determined to be malicious. Otherwise, the sample is declared as benign.
- **Metric:** In our research, we employ classic evaluation metrics from machine learning to assess the performance of detection models, including Accuracy (A), Precision (P), Recall (R), and F1-Score (F1).

5.3 Result and Analysis

As shown in Table 3, we analyze the impact on two different malware detectors when different proportions and types of obfuscated samples are introduced into the training set. Here, *OBRat* represents the proportion of obfuscated samples. *Original* denotes the original performance, i.e., training the model using the original labels of the dataset. *CID* and *CSE* represent the types of obfuscated samples mixed into the training set.

The results demonstrate that when using the original training set, the performance of the two detectors is exceptionally high. However, as the proportion of obfuscated samples increases, all performance metrics exhibit a declining trend. For instance, with 50%-mixed *CID* obfuscated samples, Drebin and Malscan's F1 decrease to 0.840 and 0.918, representing the reductions of 16.99% and 5.14% from their original performance, respectively. Similarly, with 50%-mixed *CSE* obfuscated samples, Drebin and Malscan's F1 decrease to 0.724 and 0.792, representing reductions of 35.62% and 21.87% from their original performance, respectively.

Additionally, it can be seen that for the Malscan, when a small number of *CID* obfuscated samples are introduced into the training set, their performance slightly surpasses the original performance. This is because *CID* obfuscation changes the features of Malscan more, which may create some "robust" samples, similar to the effects of data augmentation [23]. However, as the proportion of obfuscated samples increases, the generation of more samples with label errors adversely affects the performance of the detectors.

> ***Answer to RQ2****: The performance of the malware detectors is impaired by relabeling obfuscated samples incorporated into its training set. When 50% CID-obfuscated samples are mixed in, Drebin's F1 drops by 16.99%. Similarly, after incorporating 50% CSE-obfuscated samples, Drebin's and Malscan's F1 decrease by 35.62% and 21.87%, respectively.*

6 RQ3: Evaluating Individual Engines

6.1 Goal and Setup

In RQ1 and RQ2, we found that code obfuscation may lead to the change of VTP for samples, and adversely affect model training. In this section, we will further investigate the underlying reasons behind this problem, by investigating which engines experience

label flipping when facing code obfuscation. To achieve this goal, we evaluate every engine from the perspectives of availability, effectiveness and robustness. Since our goal is not to compare various engines, we will use *Engine Index* and *Engine ID* to refer to the engines' original names.

We first analyze the availability of each engine. Each engine returns three possible results for an uploaded sample: malicious, benign, or unavailable. For an engine, we use *AvailRate* to represent its availability rate, i.e., the probability of the engine returning available results (*i.e.,* benign or malicious). If an engine has a low *AvailRate*, it should be excluded in the subsequent analysis to avoid collecting insufficient reports. We then focus on engines with high *AvailRate* and analyze their effectiveness and robustness to code obfuscation. Finally, we will classify the engines based on the experimental results.

6.2 Result and Analysis

6.2.1 Availability. In the previous experiments, we obtained 131,894 valid analysis reports. We found **93** engines from these reports. We then calculate the *AvailRate* for each engine, as shown in Figure 3. This figure reveals that 60 engines have an *AvailRate* exceeding 0.9, accounting for 64.5% of all engines. For the remaining 33 engines, their *AvailRate*s are not more than 0.6. Furthermore, EN55 and EN59 are not selected by us due to their low *AvailRate* on certain categories of the original dataset. Specifically, EN59 only has a 1.85% *AvailRate* in the Adware category and EN55 has a 5.34% *AvailRate* in the Benign category.

It is noted that different engines exhibit varying degrees of changes in their *AvailRate* between the original and obfuscated samples. Take samples in the Adware category for example. The *AvailRate* of *EN59* engine on the original dataset is 0.018, while the average *AvailRate* on the obfuscated dataset is 0.976. For *EN63*, its *AvailRate* on the original dataset is 0.968, while the average *AvailRate* on the obfuscated dataset is only 0.105. For EN73 and EN69, their *AvailRate*s are relatively low on both the original and obfuscated datasets. Then we illustrate the variations in *AvailRate* of each engine before and after obfuscation. The result shows that the *AvailRate* of 4 engines has been changed by more than 30%, and that of 14 engines has been changed by more than 20%. It is worth noting that such changes will lead to the VTP variation before and after obfuscation. However, these changes in *AvailRate* do not follow a clear pattern and have a random nature. Therefore, we attribute this impact to the random factor. Fortunately, the impact of these engines on VTP is limited, as most of them exhibit a low *AvailRate*.

Taking the above two points into consideration, a total of 35 engines that generate random factors have been identified. In the next subsection, therefore, we will select **58** engines with an adequate number of reports on both original and obfuscated samples for in-depth analysis. For convenience, the selected 58 engines are marked in red in Table 1 of the supplementary material.

6.2.2 Effectiveness and Robustness. We further investigate the effectiveness and robustness of the selected 58 engines. Specifically, we first calculate the *false positive rate* (FPR) and *false negative rate* (FNR) for each engine on the original dataset and the various obfuscated datasets. FPR represents the proportion of benign samples that are predicted as malicious among all benign samples. FNR

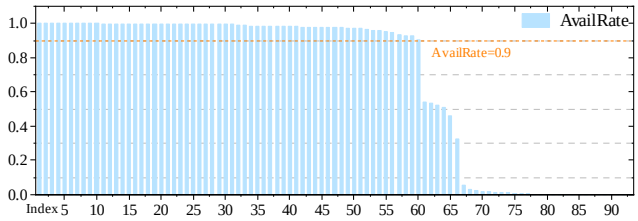

Figure 3: The *AvailRate* of 93 engines of VirusTotal.

denotes the proportion of malicious samples that are predicted as benign among all malicious samples. The FPR (or FNR) before and after obfuscation can reflect the label-flipping problem of benign (or malicious) samples. Figure 4 and Figure 5 present the FPR and FNR for each engine on the original dataset and various obfuscated datasets, respectively. Every row corresponds to an engine, and every column indicates a dataset. "ORI" represents the original dataset, and "RDM~ARE" represents the obfuscated datasets with different types of code transformations[2]. In these two figures, heatmaps are used to visualize the results, where the color intensity represents the magnitude of the corresponding value. Darker **red** color indicates values closer to 1, darker **blue** color indicates values closer to 0, and **white** color represents values of 0.5.

To understand the label flipping of each engine on benign samples, we focus on the metric of FPR. As shown in Figure 4, the FPR of all engines is low on the original dataset. This indicates that the engines can correctly identify the majority of benign samples as benign. However, it is obvious that when the original samples are obfuscated, the FPR of some engines increases significantly, indicating that a large number of benign samples experience label flipping and are falsely classified as malicious. Moreover, different engines exhibit varying degrees of label flipping in response to various obfuscating transformations. For example, engines such as EN58 and EN30 show a significant boost in FPR over all categories of obfuscated datasets. This reveals that benign samples obfuscated with any type of obfuscation technique are at higher risk of being misclassified as malicious by these engines. Additionally, some engines show higher label flipping rates on specific obfuscation techniques. For instance, the engine EN11 exhibits higher FPR on CSE, REF, and AREF obfuscated datasets compared to other obfuscated datasets. This tells us that benign samples transformed by these obfuscation techniques are more likely to be misclassified as malicious by EN11. In addition, several engines, such as EN37, EN20, and EN9, have higher FPR on CSE obfuscated datasets. This implies that applying the CSE obfuscation technique to benign samples is more likely to result in label flipping on multiple engines.

On the other hand, we observe that some engines (located at the bottom of Figure 4) consistently maintain low FPR on different types of obfuscated datasets. This phenomenon suggests that these engines may possess relatively high stability against code obfuscation techniques. As shown in Figure 5, however, when we shift our focus to the FNR of these engines, we observe that their FNR is consistently high. This implies that regardless of whether the samples are benign or malicious, these engines tend to classify

[2]Here we use code transformation types to represent the corresponding obfuscated datasets for convenience of representation.

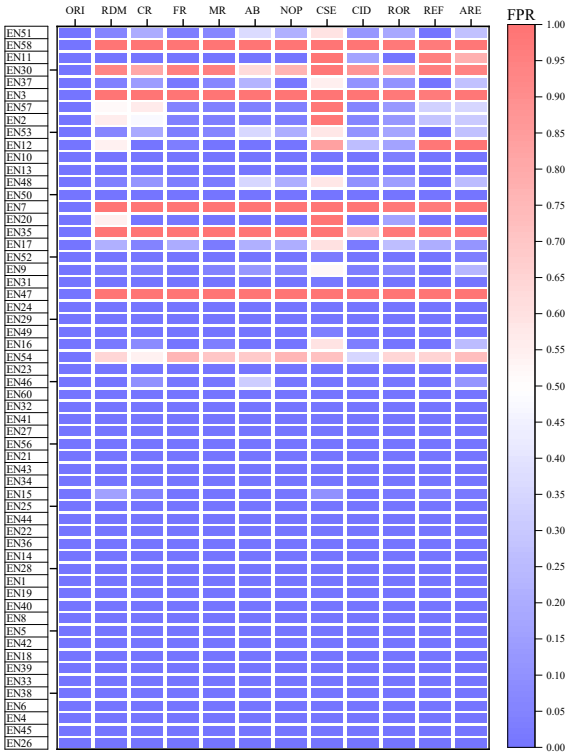

Figure 4: FPR of various engines on different datasets.

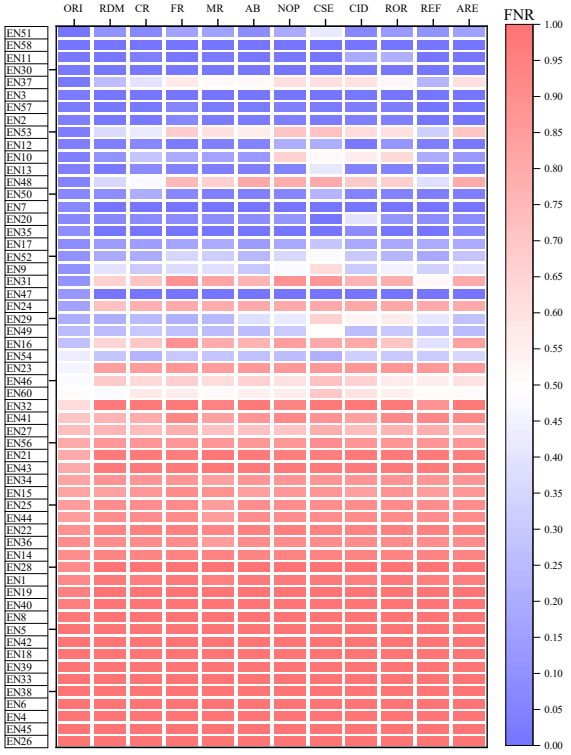

Figure 5: FNR of various engines on different datasets.

them as benign with a significant probability without discrimination. Surprisingly, this problem exists on 13 engines, accounting for 22.4% of all the engines. We believe that these engines themselves are biased.

We continue to analyze the FNR of the remaining engines. Here we consider the malicious samples in both the original and the obfuscated datasets. It is observed that the FNR of many engines increases when malicious samples are obfuscated. Moreover, these engines have varying degrees of FNR increase. Engines such as EN37, EN48, EN16, EN9, etc., exhibit an FNR increment of over 0.2 regardless of the type of obfuscated transformations applied to the malicious samples. This means that their output labels are reversed when handling more than 20% of the malicious samples. Once again, CSE exhibits the most significant impact on the labels of malicious samples. 10 engines such as EN37, EN9, and others have an FNR increment of over 0.5 on the CSE dataset. This indicates that when the original malicious samples are obfuscated with CSE transformation, these engines encounter label flipping on over 50% of the samples. Among them, the engines of EN48 and EN31 show a label flipping rate of over 80%. The engine **EN50** demonstrates relatively stable FNR on other types of obfuscated datasets, but on CR and CSE, the label flipping rate exceeds 20%.

Answer to RQ3: *The main causes for VTP changes are two-fold: the random factor and the intrinsic factor.* 35 *low-availability engines contribute to the random factor, and* 58 *obfuscation-vulnerable engines contribute to the intrinsic factor. In comparison, the impact of the random factor is relatively small, while the impact of the intrinsic factor is significant.*

7 RQ4: Mitigating the Impact of Code Obfuscation

From RQ3, we observe that the quality of various engines is inconsistent. If all engines are used for labeling, the presence of code obfuscation may lead to label flipping, and hence degrade the detection performance. Therefore, it is necessary to categorize the engines based on a deeper understanding of the engines. As a preliminary attempt to tackle obfuscation for dataset annotation, we mitigate the impact of code obfuscation by selecting and using more robust engines, based on understanding and categorization of various engines. Then we use the SVTP, which is the decision aggregation result of the various engine sets, to replace the VTP.

7.1 Taxonomy

According to the analysis of each engine in RQ3, we categorize the total 93 engines into several sets.

(1) *Low-availability Engines*: These engines return valid results with a lower probability, as we have discussed in RQ3. There are **35** engines in this category.

(2) *Biased Engines*: These engines can return results, but the results are not informative as they almost classify all samples as benign. There are **13** engines in this category.

After excluding the low-availability and biased engines, we classify the remaining ones based on detection performance and obfuscation robustness. We calculate the F1 of each engine on the original dataset to represent its original detection effectiveness, denoted as $F1_{ori}$. Then we compute the change in F1 caused by code obfuscation, *i.e.*, $F1_{Var}$. For every engine, we use the average $F1_{Var}$ on various obfuscated datasets to represent its overall obfuscation

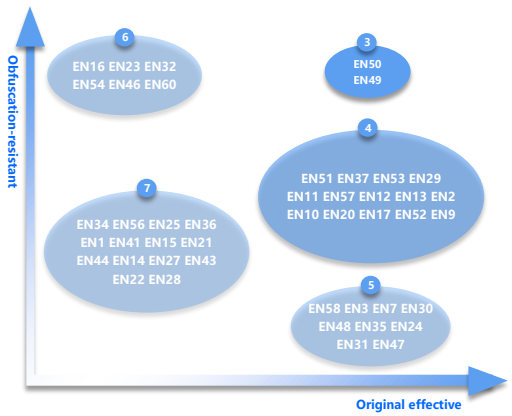

Figure 6: Cluster analysis of engines in sets ③-⑦.

sensitivity, denoted as AVG_{obf}. We use the standard deviation of $F1_{Var}$, denoted as STD_{obf}, to represent its sensitivity to different obfuscation transformations.

We then group the rest engines into five categories. Subsequently, we analyze the effectiveness and robustness of the engines in these five categories and summarize the characteristics of each category. Here we first depict the cluster analysis results of the rest engines in Figure 6, where the x-axis represents detection effectiveness and the y-axis denotes obfuscation robustness. We then summarize the 5 categories as follows.

③ *Robust Engines*: These engines have a high detection capability for original samples, and exhibit lower sensitivity to obfuscation. That is, their AVG_{obf} and STD_{obf} are relatively low. There are **2** engines in this set, as shown in Figure 6. ④ *Partially obfuscation-vulnerable engines*: These engines have a high detection capability for original samples. However, they are vulnerable to some of the obfuscation transformations, *i.e.,* their STD_{obf} is very high. There are **14** engines in this set. ⑤ *Obfuscation-vulnerable engines*: These engines have a high recognition capability for original samples. However, they experience a certain degree of F1 decrease on all obfuscation datasets, *i.e.,* they have a high AVG_{obf} and a relatively small STD_{obf}. There are **9** engines in this set. ⑥ *Ineffective and obfuscation-resistant engines*: These engines have a low recognition capability for original samples, but they are resistant to obfuscation techniques. There are **6** engines in this set. ⑦ *Ineffective and obfuscation-vulnerable engines*: These engines have a low recognition capability for original samples, and they are vulnerable to obfuscation techniques. There are **14** engines in this set.

Inspired by our categorization, we can attribute the VTP change to two kinds of factors, i.e., *random* and *intrinsic*. The engines in set ① contribute to the random factor. These engines occasionally output valid labels for input samples, slightly changing the VTP in an unpredictable manner. The engines that are vulnerable to obfuscation to different extents contribute to the intrinsic factor. There are a total of **28** such engines, which exist in three sets, ④, ⑤ and ⑥. These engines have a significant and definite impact on the VTP change between original and obfuscated samples.

7.2 Method

In the following, we propose to use different sets of engines in dataset annotation, in order to better train various malware detectors. It should be noted that based on the engine categorization in the last subsection, one can design multiple methods to tackle code obfuscation. For simplicity and as a demonstration of method design, in this subsection, we only present a simple yet effective method. Furthermore, we use extensive experiments to evaluate the proposed method. We use the same experimental setup as RQ2.

7.2.1 Method Overview. We first give a brief introduction to our proposed method. To more clearly demonstrate the impact of different engine sets on sample annotation, we annotate the dataset by aggregating the results of engines from different sets. For clarity, the original decision aggregation result on all engines is denoted as VTP, while the decision aggregation results of the various engine sets are denoted as SVTP. Figure 7 illustrates the difference between VTP and SVTP.

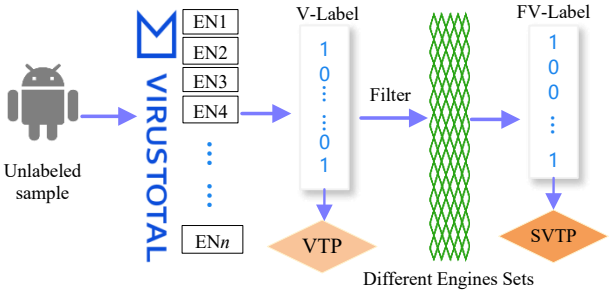

Figure 7: Computational methods of VTP and SVTP.

For a sample x, we first upload it onto VirusTotal to obtain the output *V-Label*. The VTP is the sum of *V-Label*. Unlike VTP, the calculation of SVTP requires passing through a filter. This filter will screen out the results output by engines that do not belong to the considered engine sets, producing an output *FV-Label*. The SVTP is the sum of *FV-Label*. With the SVTP, one can determine whether the sample is benign or malicious based on a threshold. To optimize the annotation performance of each engine set, we utilized ROC curves to help determine the optimal decision threshold. We found that when the SVTP threshold is set to 1, the performance of each engine set was optimal. This setting is consistent with the setting of VTP as well.

7.2.2 Comparison of Different Engine Sets. Here we compare the detection performance brought by different engine sets. Figure 8 presents the test results (*F1*) of various detectors trained on training sets annotated by different engine sets. The training sets annotated by different sets are referred to as *SetN*. For example, *Set2* refers to the training set annotated by engines belonging to the engine set ②. Here, we do not use engines from set ① to label samples because their availability is too low to obtain a sufficient number of annotated samples. Specifically, *Set58* denotes the result annotated by 58 engines after filtering out the engines from the engine set ①.

The results indicate that Set2 generally exhibits lower performance, with the F1 consistently ranging between 0.3 and 0.4. This is attributed to the engines in Set2 predominantly classifying samples as benign, resulting in a high degree of randomness in the detection outcomes. Set3 demonstrates higher performance and is comparatively the most stable. For instance, on the Drebin detector, the performance remains virtually unchanged when 50% of CID

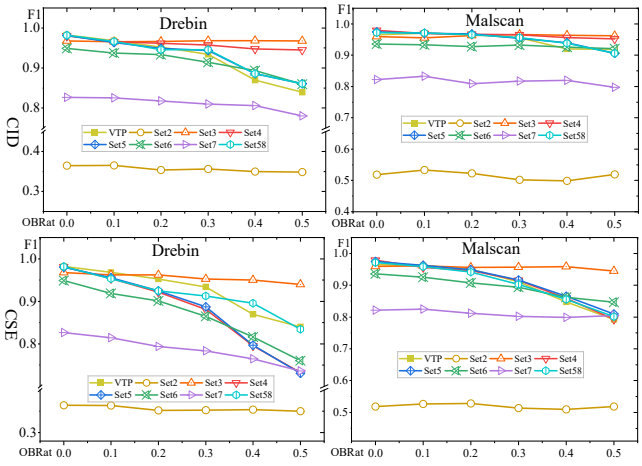

Figure 8: The F1 of different detectors on the training dataset labeled with different engine sets.

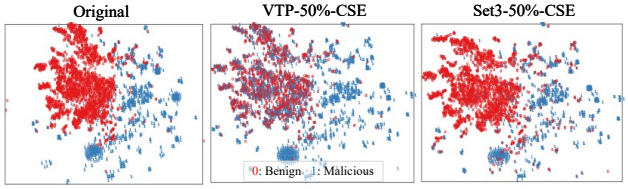

Figure 9: The distributions of the original, the VTP-annotated containing 50% CSE obfuscated, and the Set3-annotated containing 50% CSE obfuscated samples.

obfuscated samples are introduced into the training set. Across all cases, the maximum performance drop for Set3 is only 2.95%. Particularly in scenarios with CID obfuscation, the performance decline for various detectors does not exceed 1% across all proportions of obfuscated samples. Set4 exhibits high initial performance[3] and good stability in some cases. For example, when different proportions of CID and CSE obfuscated samples are introduced, Drebin's performance remains largely unaffected. Set5 has a high initial performance but shows less stability. Set6 is also less stable, with performance significantly degraded compared to the initial. Although Set7 demonstrates high stability, its performance is lower. Additionally, the results from Set58 reveal that filtering out engines with lower effectiveness, as compared to the VTP, is beneficial for performance enhancement. In summary, it is apparent that with the appropriate selection of engines for dataset annotation, when training sets are mixed with various proportions of obfuscated samples, the performance of Drebin and Malscan can be maintained at 97.28% and 99.32% of their initial capabilities, respectively. At their highest performance levels, these two detectors surpass the VTP by 15.21% and 19.23%, respectively.

To clearly understand why the annotation with Set3 can lead to better performance, we take Malscan as an example, and depict the distribution difference between three training sets: 1) the original training set, 2) the VTP-annotated training set that contains 50% CSE-obfuscated samples, and 3) the Set3-annotated training set that contains 50% CSE-obfuscated samples in Figure 9. Here we

[3]Here we define the *initial performance* of each set as the performance without the injection of obfuscated samples in the training set, i.e., the performance when *OBRat=0.0*.

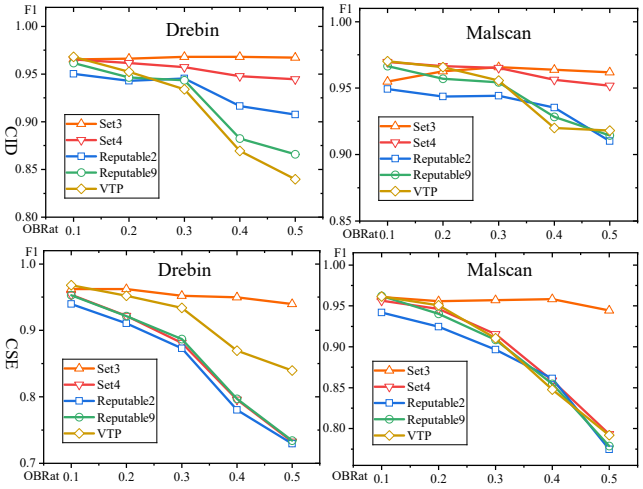

Figure 10: The F1 of different detectors on the dataset labeled with different engine sets and high-reputation engine sets.

used t-SNE [39] for visualization. It is evident that the distribution of the training data annotated with Set3 closely resembles that of the original training data, exhibiting high separability between benign and malicious samples. Conversely, the VTP-based annotation results in a significant overlap between the distributions of benign and malicious samples. Under this situation, the models trained on the VTP-annotated datasets have inferior performance, while those trained on Set3-annotated datasets demonstrate better performance.

7.2.3 Comparing Our Select Engine Sets with High-reputation Engine Sets. Some papers have mentioned several engines with high reputations [12] [46], which fall into two high-reputation engine sets shown in Table 4. The engine set Reputable2 includes 2 high-reputation engines, while the engine set Reputable9 contains 9 high-reputation engines. Among them, the *EN61* engine was excluded from this set due to its low availability, so we primarily used the remaining 8 engines in the engine set. In this subsection, we compare these two high-reputation engine sets with our selected engine sets through experiments.

Table 4: The introduction of high-reputation engines.

Set	Engines
Reputable2	EN35, EN51
Reputable9	EN35, EN51, EN61, EN15, EN58, EN37, EN52, EN11, EN50

The results are depicted in Figure 10. Observed that as the proportion of CID obfuscated samples injected into the training set increases to 50%, Drebin's F1 decreases by 6.12% and 13.15% on Reputable2 and Reputable9, respectively, while it only decreases by 0.04% and 3.80% on Set3 and Set4, respectively. Similarly, a similar trend is observed for Malscan. When CSE obfuscation is injected into the training set, both detectors show stable F1 only on Set3 annotations. Taking Drebin as an example, its F1 decreased by 32.04% and 33.50% on Reputable2 and Reputable9, respectively, while the F1 score on Set3 only decreased by 2.95%. In summary, compared to using high-reputation engines, using Set3 for dataset annotation

averagely improves Drebin's performance by 14.83% and Malscan's performance by 15.90%. This indicates that high-reputation engines selected in previous literature are not able to robustly annotate samples in the presence of code obfuscation.

Answer to RQ4: *We identify two engine sets with better performance. When the training datasets are injected with high proportions of obfuscated samples: 1) Using Set3 for dataset annotation can improve the performance of Drebin and Malscan by 15.21% and 19.23%, respectively, as compared to using VTP. 2) Using Set3 for dataset annotation can improve the performance of Drebin and Malscan by 14.83% and 15.90%, respectively, as compared to using the high-reputation engine sets.*

8 RQ5: The Impact of Code Obfuscation on the String Labels of Engines

In this section, we investigate how different obfuscations affect the string labels output by various engines. We observed that when an engine labels a sample as malicious, it will generate a string label, e.g., *ADWARE/ANDR.Mobisec.FRRR.Gen*. The string label can reveal some information about the sample [21][36], such as the family label. To study the consistency of string labels before and after obfuscation, we select 15 engines for evaluation, which can maintain a malicious detection ratio of over 1/3 after obfuscation. We first introduce an evaluation metric known as the **string labels Consistency Ratio (SLConsistRatio)**, calculated as the proportion of samples for which an engine produces identical string labels before and after obfuscation.

As shown in Table 5, we present the *SLConsistRatio* of these 15 engines under various obfuscation techniques. These obfuscation techniques have varied impacts on the consistency of the string labels output by different engines. Overall, compared to other types of obfuscation, CSE and CID generally have a more pronounced effect on most engines. Moreover, it can be seen that the engines EN57 and EN37 show a notably low *SLConsistRatio* across all obfuscation techniques. Under various obfuscation techniques, EN57 changes the string label of the sample only from the original *'Malicious (score: 85)'* to *'Malicious (score: 99)'*. For EN37, the string label output by it contains a unique identifier, leading to a high inconsistency in its output string labels. In comparison, the engines EN50 and EN51 exhibit high robustness, particularly when facing CSE and CID obfuscations.

Table 5: The *SLConsistRatio* of engines under different obfuscation techniques.

Obfuscation	AREF	AB	CID	CR	CSE	FR	MR	NOP	RMF	REF	ROR
EN50	0.94	0.944	0.822	0.685	0.773	0.814	0.938	0.937	0.924	0.94	0.936
EN49	0.79	0.787	0.773	0.583	0.547	0.734	0.744	0.836	0.78	0.826	0.835
EN51	0.91	0.91	0.901	0.788	0.628	0.888	0.888	0.928	0.885	0.889	0.895
EN11	0.596	0.816	0.314	0.737	0.336	0.813	0.707	0.815	0.814	0.623	0.418
EN37	0.001	0	0	0.001	0.002	0.002	0.001	0.002	0	0	0.001
EN57	0	0	0	0	0	0	0	0	0	0	0
EN2	0.412	0.491	0.389	0.269	0.145	0.577	0.619	0.506	0.615	0.392	0.497
EN12	0.76	0.646	0.255	0.589	0.316	0.666	0.827	0.519	0.825	0.575	0.48
EN10	0.883	0.877	0.364	0.549	0.382	0.79	0.793	0.323	0.862	0.685	0.353
EN13	0.157	0.166	0.197	0.126	0.083	0.12	0.136	0.136	0.327	0.115	0.11
EN20	0.655	0.592	0.268	0.651	0.454	0.675	0.676	0.611	0.668	0.602	0.622
EN17	0.677	0.702	0.603	0.638	0.509	0.658	0.689	0.666	0.66	0.667	0.697
EN52	0.559	0.468	0.456	0.376	0.245	0.347	0.487	0.538	0.436	0.483	0.508
EN9	0.886	0.741	0.696	0.559	0.544	0.81	0.83	0.889	0.692	0.774	0.872
EN29	0.851	0.601	0.553	0.822	0.511	0.923	0.89	0.589	0.924	0.736	0.485

The color scale is set by the value of the column. Darker ***red*** color indicates values closer to 1, *darker* ***blue*** color indicates values closer to 0.

In the following, we investigate the consistency changes between the string labels output by engines before and after obfuscation, with a focus on family labels. Here we introduce the AVClass[36], which can aggregate the string labels produced by various engines and input the constant family labels. Note that if less than two engines can generate a consistent family label within a sample's AV report, the sample will be marked as '*SINGLETON*'. We define **FMConsistRatio** as the ratio of samples whose family labels remain consistent before and after obfuscation to the total number of samples, similar to *SLConsistRatio*. The experimental results[4] are shown in Figure 11. It can be seen that the FMConsistRatio decreases under all types of obfuscation. In comparison to other obfuscation techniques, CSE significantly lowers FMConsistRatio, with a ratio of only 0.528, while the latter remains above 0.7 for other types of obfuscation.

We further calculated the proportion of samples that are transitioned from an original family label to *SINGLETON* under each obfuscation technology, which is called **SingleRatio**. The experimental results, as shown in Figure 12, indicate that the SingleRatio for most obfuscation categories exceeds 20%. This demonstrates that obfuscation affects the consistency of string labels among engines.

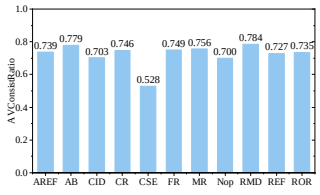

Figure 11: The FMConsistRatio under different obfuscation technologies.

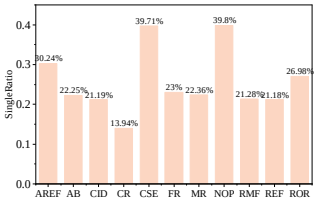

Figure 12: The SingleRatio under different obfuscations.

Answer to RQ5*: Code obfuscation not only impacts the consistency of string labels within a single engine, but also impacts the consistency of string labels across different engines.*

9 Discussion

Engine Selection. VirusTotal integrates numerous commercial engines to provide free malware scanning services. However, it is crucial to have a deeper understanding of each engine and not blindly trust high-reputation engines. Therefore, we meticulously demonstrate the usability, effectiveness, and robustness of each engine before and after obfuscation, offering users an additional perspective for engine selection. We find that if a sample is obfuscated, only the engines in Set3 can provide relatively reliable detection results. To verify the generalizability of this finding on other malware datasets, we conduct experiments on a different dataset Malradar [41]. The details can be found in supplementary material due to the space limitations.

Engines Detection Mechanism. Since engines' internal mechanisms are kept confidential, and provided to users as a black box, it is challenging to analyze the impact of obfuscation on them from

[4]Notably, we exclude samples labeled as *SINGLETON* in the original dataset, since the transition from *SINGLETON* to *SINGLETON* does not indicate consistent engine outputs. Moreover, these samples only account for 2.97% of all samples.

the root. In theory, code obfuscation has a significant impact on signature-based engines. For instance, it can help malware evade detection by such engines. However, our experiments reveal that different obfuscations have varying effects on different engines. For example, certain code obfuscation techniques cause benign software to be identified as malware by the engines. Therefore, we have reason to believe that these engines are not exclusively based on signatures to identify malware. On the other hand, signature-based detection methods cannot identify unseen malware, but this ability is particularly important for engines. It is crucial for each engine to enhance its capability to detect unknown malware.

Label Noise. The findings of our work also alert us to be vigilant about the issue of label noise caused by obfuscation in Android malware datasets. Label noise caused by obfuscation can lead detectors to learn incorrect decision boundaries, hence degrading detection performance. Furthermore, this uncertainty can also be exploited by attackers to more easily generate adversarial examples. In our work, we conduct a more refined analysis of the performance of various engines, seeking robust engines to improve the application of antivirus engines in dataset annotation. We hope that our work will draw the attention of researchers and motivate them to prompt more solutions for label noise.

10 Limitations

Here we discuss the limitations of this work. ***1) Obfuscation Tools.*** We employed ObfuscAPK [10] to implement transformations, as it is open-source and can provide advanced strategies. Considering that different tools may obfuscate the programs in a slightly different way, we use a commercial obfuscation tool, Allatori [3], for further validation. We chose to conduct experiments on CSE obfuscation, as it has the largest impact on VTP. Our experiments show that there is no significant difference in the changing trend of VTP when using either ObfuscAPK or Allatori. Specifically, there is an increase in VTP on benign samples and a decrease in malicious samples. Therefore, we believe our method can also work when facing other obfuscation tools. Moreover, some new interesting obfuscation technologies have emerged, such as VM-based obfuscation. At present, we are primarily concerned with bytecode-based obfuscation due to its prevalence. ***2) Dataset.*** The original dataset used in our experiments may not comprehensively represent the data distribution in the real world. Furthermore, there may be some errors in the labels, which are inevitable in practice even if the samples are dynamically analyzed and manually scanned. Moreover, there may exist some obfuscated samples in our original dataset. To further validate the dataset, we manually collect a dataset with 963 clean benign samples from F-droid [4], whose code is available to us. We obfuscate these samples and upload them to VirusTotal for analysis. We find that the analysis reports exhibit the same pattern as our previous results.

11 Related Work

In this section, we review previous studies and highlight the differences between ours and theirs.

Currently, some studies have explored the characteristics of commercial engines. AV-Meter [30] employed a dataset to examine the performance of engines in detection rates, labeling accuracy, and consistency. Kantchelian et al. [22] focused on the issue of **label noise** introduced by threshold-based sample annotation using engines, and proposed an unsupervised labeling approach utilizing a generative Bayesian model. Hurier et al. [21] conducted an extensive investigation into the inconsistency among AV engines using a large dataset, proposing a set of metrics to quantify the inconsistency from different perspectives. Hammad et al. [20] evaluated top anti-malware products on different obfuscation datasets, and found that code obfuscation has a significant impact on most of them. Gashi et al. [19] revealed the issue of the same engine producing inconsistent labels for identical samples at different times. Furthermore, Zhu et al. [46] analyzed the label aggregation methods used in prior research and examined the dynamic labeling issue of VirusTotal. They showed that threshold-based aggregation methods offer greater stability and recommended a feasible range of thresholds. Moreover, they observed that obfuscated PE files are more likely to generate false positives. To address the problem of dynamic labeling, Salem et al. [35] proposed a machine learning-based labeling method, which requires training based on analysis reports from a certain period in the past. Based on confident learning, Wang et al. [42] developed a new labeling tool by integrating three existing malware detectors [12] [44] [8] to identify noisy labels in the dataset. Xu et al. [45] introduced a general framework to decrease the noise level of the existing dataset. They employed two identical deep-learning models for differential training and an outlier detection algorithm was then utilized to identify noisy samples.

Different from these studies, our work attached more importance to how obfuscation affects engines in dataset annotation. We identified the causes of negative impact by evaluating the engine in availability, effectiveness, and robustness. Finally, based on our evaluation results, we construct a robust engine set to improve dataset annotation in the presence of code obfuscation.

12 Conclusion

In this paper, we conduct a data-driven evaluation to analyze the impact of code obfuscation on dataset annotation with antivirus engines. We quantify the availability, effectiveness, and robustness of the engines in VirusTotal to assist in identifying the factors that impact dataset annotation with antivirus engines. We further propose an engine selection method to identify the robust engine sets for data annotation when obfuscations exist. Finally, we explore how different obfuscation techniques affect the string labels output by engines as an extended study. We believe our work can assist researchers when using antivirus engines for dataset annotation in the presence of code obfuscation.

Acknowledgments

This work is partially supported by the Fundamental Research Funds for the Central Universities (YCJJ20230466), the Postdoctoral Fellowship Program of CPSF under Grant Number GZB20240248, the China Postdoctoral Science Foundation under Grant Number 2024M751010 and supported by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

References

[1] 2022. DashO. (2022). https://www.preemptive.com/products/dasho/overview.
[2] 2022. Proguard. (2022). https://www.guardsquare.com/en/products/proguard.
[3] 2023. Allatori. (2023). https://allatori.com/.
[4] 2023. F-droid. (2023). https://f-droid.org/packages/.
[5] 2023. Google play store. (2023). https://play.google.com/store/apps.
[6] 2023. VirusShare. (2023). https://virusshare.com/.
[7] 2023. Virustotal. (2023). https://www.virustotal.com/.
[8] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. 2016. Empirical assessment of machine learning-based malware detectors for android. *Empirical Software Engineering* 21, 1 (2016), 183–211. https://doi.org/10.1007/s10664-014-9352-6
[9] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, USA, 468–471.
[10] Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, and Alessio Merlo. 2020. Obfuscapk: An open-source black-box obfuscation tool for Android apps. *SoftwareX* 11 (2020), 100403. https://doi.org/10.1016/j.softx.2020.100403
[11] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, and et al. 2022. Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity22/presentation/arp
[12] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceedings of Symposium on Network and Distributed System Security (NDSS)*, Vol. 14. Internet Society, 23–26. https://doi.org/10.14722/ndss.2014.23247
[13] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining Apps for Abnormal Usage of Sensitive Data. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 426–436. https://doi.org/10.1109/ICSE.2015.61
[14] Marcus Botacin, Fabricio Ceschin, Paulo de Geus, and André Grégio. 2020. We need to talk about antiviruses: challenges & pitfalls of AV evaluations. *Computers & Security* 95 (2020), 101859. https://doi.org/10.1016/j.cose.2020.101859
[15] Christian Collberg and Douglas Low. 1997. *A taxonomy of obfuscating transformations*. Technical Report. Department of Computer Science, The University of Auckland, New Zealand.
[16] Thaís Damásio, Michael Canesche, Vinícius Pacheco, Marcus Botacin, Anderson Faustino da Silva, and Fernando M. Quintão Pereira. 2023. A Game-Based Framework to Compare Program Classifiers and Evaders. In *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization (CGO 2023)*. ACM, New York, NY, USA, 108–121. https://doi.org/10.1145/3579990.3580012
[17] Li Menghao Dong Shuaike and et al. 2018. Understanding android obfuscation techniques: A large-scale investigation in the wild. In *Proc. SecureComm*. https://doi.org/10.1007/978-3-030-01701-9_10
[18] C. Gao, G. Huang, H. Li, B. Wu, Y. Wu, and W. Yuan. 2024. A Comprehensive Study of Learning-based Android Malware Detectors under Challenging Environments. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA, 104–116. https://doi.org/10.1145/3597503.3623320
[19] Ilir Gashi, Bertrand Sobesto, Stephen Mason, Vladimir Stankovic, and Michel Cukier. 2013. A study of the relationship between antivirus regressions and label changes. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. 441–450. https://doi.org/10.1109/ISSRE.2013.6698897
[20] 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 *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 421–431. https://doi.org/10.1145/3180155.3180228
[21] Médéric Hurier, Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. On the Lack of Consensus in Anti-Virus Decisions: Metrics and Insights on Building Ground Truths of Android Malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer International Publishing, Cham, 142–162. https://doi.org/10.1007/978-3-319-40667-1_8
[22] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. 2015. Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security (AISec '15)*. Association for Computing Machinery, New York, NY, USA, 45–56. https://doi.org/10.1145/2808769.2808780
[23] Bohan Li, Yutai Hou, and Wanxiang Che. 2022. Data augmentation approaches in natural language processing: A survey. *Ai Open* 3 (2022), 71–90. https://doi.org/10.1016/j.aiopen.2022.03.001
[24] Heng Li, Zhang Cheng, Bang Wu, Liheng Yuan, Cuiying Gao, Wei Yuan, and Xiapu Luo. 2023. Black-box Adversarial Example Attack towards FCG Based Android Malware Detection under Incomplete Feature Information. (2023). https://doi.org/10.5555/3620237.3620304
[25] Samaneh Mahdavifar, Dima Alhadidi, Ali Ghorbani, et al. 2022. Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of Network and Systems Management* 30, 1 (2022), 1–34. https://doi.org/10.1007/s10922-021-09634-4
[26] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, and et al. 2020. Dynamic android malware category classification using semi-supervised deep learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/-CyberSciTech)*. IEEE, 515–522. https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094
[27] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J. Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society. https://doi.org/10.14722/ndss.2017.23353
[28] O. Mirzaei, JMD Fuentes, and et al. 2018. ANDRODET: An adaptive Android obfuscation detector. *FUTURE GENER COMP SY* 90, 4 (2018). https://doi.org/10.1016/j.future.2018.07.066
[29] Tom Michael Mitchell et al. 2007. *Machine learning*. Vol. 1. McGraw-hill New York.
[30] Aziz Mohaisen and Omar Alrawi. 2014. AV-Meter: An Evaluation of Antivirus Scans and Labels. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer International Publishing, Cham, 112–131. https://doi.org/10.1007/978-3-319-08509-8_7
[31] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, Lorenzo Cavallaro, et al. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Security Symposium*. 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury
[32] Fabio Pierazzi, Feargus Pendlebury, and et al. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1332–1349. https://doi.org/10.1109/SP40000.2020.00073
[33] Joel Reardon, Álvaro Feal, Primal Wijesekera, and et al. 2019. 50 ways to leak your data: An exploration of apps' circumvention of the android permissions system. In *28th USENIX security symposium (USENIX security 19)*. 603–620.
[34] Xiaolei Ren, Michael Ho, Jiang Ming, Yu Lei, and Li Li. 2021. Unleashing the Hidden Power of Compiler Optimization on Binary Code Difference: An Empirical Study. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021)*. ACM, New York, NY, USA, 142–157. https://doi.org/10.1145/3453483.3454035
[35] Aleieldin Salem, Sebastian Banescu, and Alexander Pretschner. 2021. Maat: Automatically analyzing virustotal for accurate labeling and effective malware detection. *ACM Transactions on Privacy and Security (TOPS)* 24, 4 (2021), 1–35. https://doi.org/10.1145/3465361
[36] Marcos Sebastián, Richard Rivera, and et al. 2016. Avclass: A tool for massive malware labeling. In *Proc. RAID*. Springer. https://doi.org/10.1007/978-3-319-45719-2_11
[37] Silvia Sebastián and Juan Caballero. 2020. Avclass2: Massive malware tag extraction from av labels. In *Proc. ACSAC*. 42–53. https://doi.org/10.1145/3427228.3427261
[38] Kimberly Tam, Aristide Fattori, Salahuddin Khan, and Lorenzo Cavallaro. 2015. Copperdroid: Automatic reconstruction of android malware behaviors. In *NDSS Symposium 2015*. 1–15. https://doi.org/10.14722/NDSS.2015.23145
[39] Laurens Van Der Maaten. 2014. Accelerating t-SNE using tree-based algorithms. *The journal of machine learning research* 15, 1 (2014), 3221–3245. https://doi.org/10.5555/2627435.2697068
[40] Haoyu Wang, Junjun Si, Hao Li, and Yao Guo. 2019. Rmvdroid: towards a reliable android malware dataset with app metadata. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*. IEEE, 404–408.
[41] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying Android Malware in the New Era. *SIGMETRICS Perform. Eval. Rev.* 50, 1 (jul 2022), 21–22. https://doi.org/10.1145/3530906
[42] Liu Wang, Haoyu Wang, Xiapu Luo, and Yulei Sui. 2022. MalWhiteout: Reducing Label Errors in Android Malware Detection. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13. https://doi.org/10.1145/3551349.3560418
[43] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 252–276. https://doi.org/10.1007/978-3-319-60876-1_12
[44] Yueming Wu, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. 2019. MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 139–150. https://doi.org/10.1109/ASE.2019.00023

[45] Jiayun Xu, Yingjiu Li, and Robert H Deng. 2021. Differential training: A generic framework to reduce label noises for android malware detection. In *Proc. of Network and Distributed Systems Security (NDSS) Symposium*. https://doi.org/10.14722/NDSS.2021.24126

[46] Shuofei Zhu, Jianjun Shi, Limin Yang, and . et al. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *29th USENIX Security Symposium (USENIX Security 20)*. 2361–2378. https://doi.org/10.5555/3489212.3489345