Sparsity Brings Vulnerabilities: Exploring New Metrics in Backdoor Attacks

Jianwen Tian, *NKLSTISS, Institute of Systems Engineering, Academy of Military Sciences, China;* Kefan Qiu, *School of Cyberspace Science and Technology, Beijing Institute of Technology;* Debin Gao, *Singapore Management University;* Zhi Wang, *DISSec, College of Cyber Science, Nankai University;* Xiaohui Kuang and Gang Zhao, *NKLSTISS, Institute of Systems Engineering, Academy of Military Sciences, China*

https://www.usenix.org/conference/usenixsecurity23/presentation/tian

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Sparsity Brings Vulnerabilities: Exploring New Metrics in Backdoor Attacks

Jianwen Tian[1], *Kefan Qiu*[2], *Debin Gao*[3], *Zhi Wang*[4*], *Xiaohui Kuang*[1*], *Gang Zhao*[1]
[1] *NKLSTISS, Institute of Systems Engineering, Academy of Military Sciences, China*
[2] *School of Cyberspace Science and Technology, Beijing Institute of Technology*
[3] *Singapore Management University*
[4] *DISSec, College of Cyber Science, Nankai University*
Email: jianwentian1994@foxmail.com, kfqiu@bit.edu.cn, dbgao@smu.edu.sg,
zwang@nankai.edu.cn, xiaohui-kuang@163.com, zemell@foxmail.com

Abstract

Nowadays, using AI-based detectors to keep pace with the fast iterating of malware has attracted a great attention. However, most AI-based malware detectors use features with vast sparse subspaces to characterize applications, which brings significant vulnerabilities to the model. To exploit this sparsity-related vulnerability, we propose a clean-label backdoor attack consisting of a dissimilarity metric-based candidate selection and a variation ratio-based trigger construction.

The proposed backdoor is verified on different datasets, including a Windows PE dataset, an Android dataset with numerical and boolean feature values, and a PDF dataset. The experimental results show that the attack can slash the accuracy on watermarked malware to nearly 0% even with the least number (0.01% of the class set) of watermarked goodwares compared to previous attacks. *Problem space constraints* are also considered with experiments in *data-agnostic scenario* and *data-and-model-agnostic scenario*, proving transferability between different datasets as well as deep neural networks and traditional classifiers. The attack is verified consistently powerful under the above scenarios. Moreover, eight existing defenses were tested with their effect left much to be desired. We demonstrated the reason and proposed a subspace compression strategy to boost models' robustness, which also makes part of the previously failed defenses effective.

1 Introduction

With the escalating prevalence of cyber-attacks, traditional analysis cannot keep pace with the evolution of malware. This has prompted a significant shift amongst researchers and organizations towards the utilization of machine learning (ML) and deep learning (DL) to address the challenge of large-scale malware detection. [4, 15, 33, 64, 67].

Machine learning approaches are perceived as a panacea for malware detection as their inductive reasoning mechanism helps fight against traditional evasive methods, such as polymorphic [39] and metamorphic techniques [27]. Nonetheless, the popularity of learning approaches also attracts the attention of adversaries, leading to the emergence of a multitude of AI-related security issues. For instance, adversarial attacks, which happen at the inferring stage, enable a sample to bypass detection by applying minor changes to the sample's feature space [7, 10, 14, 15, 28, 35]; poisoning attacks, happening at the training stage, corrupts the decision-making of the target model by injecting elaborated poisons into the training set [22, 23, 30, 49, 61, 66]. As a specific variant of poisoning attacks, backdoor attacks construct backdoor triggers by poisoning the training set, resulting in samples with the triggers being classified into the target class [22, 23, 42, 66]. These backdoors can be further classified into two categories: *label-tampered* backdoor attacks where the poisons are selected from different classes and labeled as the target class and *clean-label* backdoor attacks where the poisons are directly from the target class. This paper primarily focuses on clean-label backdoor attacks due to their covert and damaging nature.

Although there have been adversarial and poisoning researches against malware detection [8, 10, 12, 15, 41, 42, 45, 49], backdoor attacks remain a significantly understudied area [20, 41, 42, 45]. Various studies have proposed poisoning attacks against ML-based malware detectors, but these largely encompass label-tampered attacks that necessitate label reversing [20, 41]. Severi et al. [42] are the first to investigate clean-label backdoor attacks on malware classifiers. Their triggers base on features with great influence; so the generated triggers have certain attack performance even without poisoning. Shapira et al. [45] proposed a sample selection strategy to select goodwares close to the target malware in Euclidean distance; therefore, their poisons work on specific malware. Lastly, Yang et al. [62] proposed a family-specific backdoor attack only to allow samples of specific families to evade detection. Different from previous works, this paper focuses on a prevailing defect observed in malware-related datasets. Since the features of malware detection tasks are customarily based on expert knowledge, features in malware detection models tend to have an extensive value span, such as [0-65535], and different features may be with different value sets [3]. Such characteristics cause a bunch of sparse areas and profoundly endanger the security of the model. Therefore, this introduces a novel clean-label backdoor attack considered both candidate sample selection and trigger feature selection.

To evaluate the proposed attack under different conditions, three widely-used datasets are employed: EMBER (Windows executables) with numerical features [3] DREBIN (Android applications) with boolean features [4], Contagio (PDFs). The experimental results reveal that the attack can slash the accuracy on watermarked malwares to nearly 0% even with the smallest number (0.01% of the class set) of poisoned goodwares compared to previous attacks. Besides, the problem space constraints [38] are considered. Under such constraints, the attack is verified in different scenarios: *data-agnostic* and *data-and-model-agnostic* scenarios, proving the transferability between different datasets, as well as neural networks and traditional models. Overall, the attack proposed in this paper outperforms the state-of-the-art backdoor attacks in malware detectors. The attack analysis under problem space constraints also solidifies that the backdoor attack is a practical threat to current AI-based detectors. Moreover, this study examines the efficacy of eight existing defenses against backdoor attacks. Five of these defenses were found to be ineffective, while the others significantly degraded model performance or were only effective on specific datasets. In light of these findings, this paper introduces a novel subspace compression strategy that significantly enhances the robustness against backdoor attacks and renders some previously ineffective defenses operational.

Furthermore, this study challenges the conventional wisdom in clean-label backdoor attacks that *increased quantities of poisons lead to higher attack performance* [42, 45]. The results in Section 6.6 highlight that *the effectiveness of an attack is more heavily influenced by the quality, rather than the quantity, of the samples*. Consequently, a large number of poorly chosen samples may actually hinder the attack performance. In addition, this paper verified a "low density" principle in trigger construction and proved it a far more critical factor than features' benign orientation.

Our contributions are summarized as follows:

- *Novel clean-label backdoor attack*: This study puts forth an innovative clean-label backdoor attack, integrating dissimilarity metrics-based candidate selection with variation ratio-based trigger feature selection (selecting existing features and values as the trigger). This approach reveals a significantly higher vulnerability in malware classifiers than previously identified.

- *Subspace compression strategy*: This paper tests the efficacy of eight defensive mechanisms against the attack, confirming their insufficient effectiveness. To remedy this, we propose a subspace compression strategy that eliminates low-density subspaces, substantially enhancing the model's robustness and improving the efficacy of some previously unsuccessful methods.

2 Background

We begin by providing essential background information related to our proposed attack in this paper.

2.1 Malware Detectors

The learning-based malware detectors classify inputs into malicious or benign sets mainly based on two feature types (`dynamic` and `static`). Dynamic features are records of suspicious behaviors, which are collected through the execution of apps on virtualized environments [2, 19, 51]. Static features are normally extracted from the binary or metadata without executing the executable files [4, 15, 34, 59].

Model builders usually extract a feature vector x from a raw binary or an Android app and train a classifier with input matrix X and the corresponding labels Y. Given the standard malware detection setting, the goal is to predict the label $y \in C = \{0,1\}$ of an input $x \in X$, where the input and label pairs are sampled i.i.d. from a distribution D. The detector is represented as a function $F_\theta : X \to C$, which is parameterized by $\theta \in R^d$. The parameters θ of the classifier are optimized by minimizing a loss function $L(x,y,\theta)$ over a training set $\widehat{D} = (x_i, y_i)_{i=1}^n$ of labeled samples:

$$\underset{\{\theta\}}{\arg\min} - \sum_{i \in \widehat{D}} \sum_{j \in C} y_{ij} * log(Prob(pred = j|x_i, \theta)) \qquad (1)$$

2.2 Adversarial Attacks

Adversarial attacks against learning approaches are generally categorized into two main types: evasion attacks and poisoning attacks. Evasion attacks leverage specific perturbations on examples to induce misclassification during the inferring phase [7, 10, 14, 15, 28, 35]. On the other hand, poisoning attacks disrupt the training process by introducing crafted examples into the dataset [22, 23, 30, 49, 61, 66].

Based on their targets, poisoning attacks can be further subdivided into three categories. *Availability* poisoning aims at degrading the target model's performance [30, 61], while *targeted* poisoning attacks strive to cause a specific example to be misclassified [49]. Both of these two attacks do not need to modify the target examples in the inferring stage. *Backdoor attacks*, on the other hand, introduce a backdoor into the model during its training phase by injecting specially crafted poisoned samples, and later watermark the target examples with the "backdoor trigger" to elicit intended misclassification during the inferring stage [5, 16, 22, 23, 40, 42, 54, 66].

A backdoor trigger is typically characterized by a specific feature pattern, and the way to generate such backdoors can be divided into two types: *label-tampered* attacks [16, 22, 66] and *clean-label* attacks [5, 40, 42, 54]. The former requires changing the label of poisoned samples while the latter directly uses poisons from the target class. Therefore, to implement a successful clean-label backdoor attack, there are two critical factors: creating a trigger easy to be learned by models [23, 42, 66] and selecting powerful candidate samples with a strong effect on model weights [45, 54].

Implementing backdoor attacks in malware classifiers: Malware classification systems typically comprise of three stages: (i) collecting training samples from external data sources such as application stores and threat intelligence platforms, (ii) extracting features representing the semantic knowledge of samples, and (iii) training the classification model. The procedure is potentially vulnerable to manipulation at the data collection stage. Specifically, 1) an attacker can devise a trigger and watermark it on selected benign samples to generate poisons based on an accessible dataset; 2) the attacker distributes the poisoned benign samples via the Internet or submit them directly to anti-virus vendors to launch a *clean-label* backdoor attacks [42], as illustrated in Figure 1.

One may consider using traceable goodwares only; however, modern learning methods usually require a large volume of data to reach better performance and generalization, and therefore, model builders may have to take the risk.

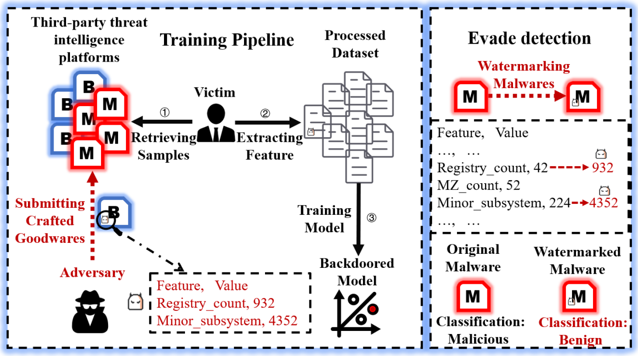


Figure 1: The backdoor attack.

2.3 Nonconformity Measure and P-value

Nonconformity Measures (NCMs) and P-values, fundamental statistical tools derived from conformal prediction [17, 44], are utilized to ascertain the legitimacy of a prediction. For example, for a new sample z^*, a hypothesis test is conducted to decide whether or not to approve the null hypothesis asserting that z^* does not belong in the prediction region formed by elements C. This is achieved by computing the p-values using the NCM values for each point. A large p-value indicates a tendency to reject the null hypothesis; in other words, the point is considered belonging to C with high p-value.

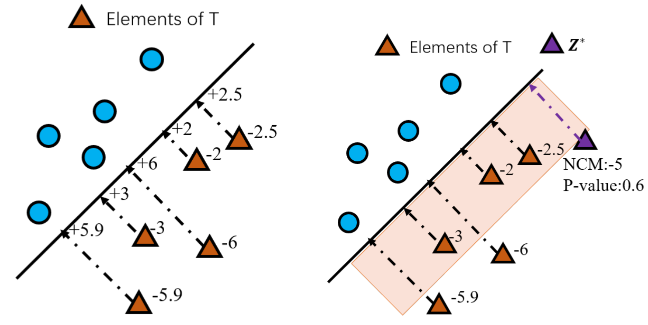
Nonconformity Measure: NCM A_D (D is a dataset containing C) is a real-valued function that quantifies the dissimilarity between an object z and a subset C, see Equation 2. Machine learning methodologies inherently possess an NCM. For instance, the negative absolute distance to the hyperplane in a support vector machine [11] can serve as an NCM as it tells how dissimilar a point is with the class set. Figure 2(a) shows an example of NCMs on a linear SVM [11]. Similarly, other classifiers also possess such characteristics, such as the negative ratio of decisions for one class in random forests or the negative output probabilities for one class of neural networks.

P-value: P-value can be calculated with the support of NCMs. For a set of objects T, p-value $P^C_{z^*}$ for a new object z^* is the proportion of objects in subset T that are at least as dissimilar to other objects in C as z^*. The calculation of p-value for a new object z^* consists of three steps: computing NCM for z^* and other samples in T (see Equation 2 and 3 respectively), and calculating p-value based on the calculated NCMs (see Equation 4 and Figure 2(b)).

$$\alpha_{z^*} = A_D(C, z^*) \tag{2}$$

$$\forall i \in T.\ \alpha_i = A_D(C \setminus z_i, z_i) \tag{3}$$

$$P^C_{z^*} = \frac{|\{j : \alpha_j \geq \alpha_{z^*}\}|}{|T|} \tag{4}$$



(a) Distance and NCM (Negative Distance)

(b) The NCM of Point z^* is smaller than 3 of 5 elements of T

Figure 2: Calculation of P-value on a linear SVM.

3 Related works

Backdoor attacks were initially introduced by Gu et al. [16] for image recognition tasks. Subsequent studies proposed various backdoor attacks aiming at enhancing attack performance and stealthiness [5, 13, 22, 40, 54, 66]. Backdoor attacks have also been explored in the domain of malware classification [20, 41, 50, 50, 65] .

The earliest instance of a poisoning attack against ML-based malware detectors was proposed by Sasaki et al. [41], which poisons one specific malware family based on back-gradient optimization [30] and causes the misclassification on a specific class. However, it assumes the strictest attack scenario that both the training dataset and leanring algorithm are accessible with labels being manipulated. Li et al. [20]

proposed a genetic-algorithm-based backdoor attack on Android dataset, and it also relies on label reversion to create poisons. Subsequently, researchers also started to develop clean-label backdoor attacks in the context of malware detection [42, 45, 62].

In terms of trigger selection, Severi et al. [42] devise backdoor triggers based on SHapley Additive exPlanations (SHAP) [25]. The explanation-guided backdoors are mainly based on two strategies: (1) searching for areas of weak confidence near the decision boundary; and (2) subverting areas that are heavily benign-oriented. It first conducts feature selection based on two concepts: **LargeSHAP** (which sums the individual SHAP values along features and then selects features with large negative values indicating strong benign-orientation) and **LargeAbsSHAP** (which takes the absolute value of SHAP before summing them up, capturing the overall importance of the feature regardless of its orientation). They also suggested three approaches for value selection: *MiniPopulation* (which selects values that occur with the least frequency), *CountSHAP* (which chooses values with a high density of benign-oriented data), and *CountAbsSHAP* (which selects values that are not strongly aligned with either class). Given these 5 concepts, the authors try different combinations to get a better attack performance, such as features and values with the largest influence on model decision (*LargeAbsSHAP*×*CountAbsSHAP*), features with the largest influence and corresponding values presented the minimal number of times *LargeAbsSHAP*×*MinPopulation*. The last one is based on *Greedy Selection*: it starts by selecting the most goodware-oriented feature using the **LargeSHAP** selector and the value of highest density in goodware-oriented data using the *CountSHAP* selector. Next, it removes all data points that do not have the selected value and repeat the procedure with the subset of data conditioned on the current trigger. Therefore, their triggers perform like an evasion attack and achieve certain performance even without poisoning. Different from other backdoors, Yang et al. [62] proposed a family-specific backdoor attack only to allow samples of specific families to evade detection. Both Severi et al. [42] and Yang et al. [62] use randomly selected samples as poisons.

In sample selection, Shapira et al. [45] proposed an instance-based method derived from poison frog attacks [43]. The instance-based method encompasses four stages: 1) Computing Euclidean distance between the target malware and goodwares, 2) Selecting goodwares that most closely resemble the target malware in terms of Euclidean distance, 3) Integrating a random trigger into these goodwares for training, and 4) Watermarking the target malware instance with the trigger to induce a misclassification.

Different from Explanation-guided triggers [42] in which features heavily oriented to goodwares are considered suitable for creating backdoors, we argue that a high density, whether benign-oriented or malicious-oriented, harms the effectiveness of backdoors. This is because the model has to accommodate all samples within such areas while maintaining the overall weights of the model. Our sample selection approach also diverges from the instance-based method introduced by Shapira et al. [45] as we employ a dissimilarity metric-based strategy for candidate sample selection. This strategy allows us to transcend the confinement to specific malwares and instead considers candidate goodwares from a comprehensive range of benign sets. In this paper, we present our observation that learning-based classifiers are far more vulnerable than reported in previous works [42, 45] due to the serious sparsity problem in malware-related datasets. By exploiting the sparsity and carefully-chosen poisons, the number of required poisons can be significantly reduced (only one for EMBER and ten for DREBIN), and the backdoor even achieves a higher attack success rate.

4 Threat Model

The adversary's goal. We consider a typical backdoor attack setting where the adversary constructs a backdoor trigger v (e.g., a pattern of features) and then creates poisons by watermarking the trigger v on benign candidates. Once the classifier F_b is trained on the poisoned dataset, the backdoor will be activated. Subsequently, a watermarked sample X_b will be assigned the benign label y_{benign}, and a sample without the trigger remains the same classification as the output of the clean classifier, which can be formalized as follows:

$$F_b(X) = F(X); \;\; F_b(X_b) = y_{benign} \tag{5}$$

In order to make the attack practical and stealthy, the attacker also tries to minimize the size of poisons and triggers.

The adversary's capabilities. Given the adversary's knowledge, the capacity of attacks are categorized into three types: the **unrestricted scenario**, the **data-agnostic scenario** and the **data-and-model-agnostic scenario**. We start by analyzing the backdoor strategy's effect on the **unrestricted scenario**, where the adversary possesses complete knowledge about the training dataset and the classifier, enabling direct modification of the feature space. Next, we consider the *problem space constraints* that limit the manipulable features to those that can be naturally watermarked in real applications. Under these constraints, we assess the effectiveness of the attack in both the **data-agnostic** scenario and the **data-and-model-agnostic** scenario.

- **Data-agnostic scenario**: The adversary is agnostic to the target training set (while possessing *imprecise* knowledge of its underlying distribution) but knows the model type and feature set. The adversary can collect a dataset and extract the same features for their own purposes.

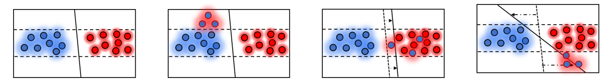
- **Data-and-Model-agnostic scenario**: The adversary lacks knowledge about both the target training set and the target model. Nonetheless, they are still aware of the feature set, allowing them to sample a dataset with the identical feature set and construct a substitute model with strong transferability, such as neural networks.

These diverse scenarios serve as a means to evaluate the practicality of the backdoor attack in a realistic context. We assume that the feature set remains perceptible in all scenarios as the backdoor attack heavily relies on known features. The analysis across these scenarios provides valuable insights into the potential destructiveness of the backdoor attack.

While acknowledging the potential impracticality of knowing the feature set, adversaries can create multiple backdoors based on different features. Certain features, such as the URL of a malicious domain, access permission to contacts or SMS, and downloading behavior, hold valuable insights for identifying malicious samples. As a result, the model builder unavoidably faces risks of using such well-known features in their quest for improved detection accuracy.

5 Achieving A Strong Backdoor Effect

Backdoor attacks are mainly based on the substantial effect of triggers on the model's decision-making. Poisons force the model to remember and recognize the trigger impressively.



(a) Toy classifier (b) Situation 1 (c) Situation 2 (d) Situation 3

Figure 3: A toy example of our motivation.

5.1 Motivation

As mentioned in Section 2.2, a backdoor's performance depends on two factors: the trigger features and the candidate samples. In this section, we use a toy example to introduce the motivation of our backdoor attack. The toy classifier is shown in Figure 3(a) where all points are well-classified primarily based on feature x (we refer x-axis as feature x and the y-axis as feature y). Therefore feature x is clearly the main classification factor. The points are distributed along with feature x and densely concentrated in a subspace of feature y. There are three different cases of setting backdoors:

1. Using poisons that are similar to the blue cluster in feature x and are within a sparse subspace of feature y (see Figure 3(b)): These poisons have little effect on the classifier since the model already performs well based on feature x. Hence, backdoors resembling the original cluster in the main classification factors are less effective.

2. Using poisons that are dissimilar to blue cluster in feature x but are within a dense subspace of feature y (refer to Figure 3(c)): These poisons have limited impact since the model aims to maintain overall accuracy. Consequently, modifying the weights contributing to the trigger becomes challenging if it lies within a dense subspace.

3. Using poisons that are dissimilar to the blue cluster in feature x and are also within a sparse subspace of feature y: As Figure 3(d) shows, these poisons demonstrate a substantial effect on modifying the weight of the classifier due to two reasons: firstly, the model cannot rely on feature x to classify the poisons, necessitating the utilization of feature y; secondly, the trigger does not reside in dense areas, allowing the model to accommodate the poisons without impairing its performance.

Based on these observations, we propose two principles for clean backdoor attacks: (1) *poisons should be dissimilar to their original cluster in the main classification factors*, and (2) *triggers should be located in sparse subspaces*. While real-world scenarios may present greater complexity, these two principles remain influential. The subsequent sections of the paper describe our strategy to adhere to these principles.

5.2 Dissimilarity Metric for Candidate Selection

In this section, we borrow two statistical tools (nonconformity measures and p-values) from conformal prediction [17, 44] to identify candidate samples that are dissimilar to benign clusters in the main classification factors (such as poisons shown in Figure 3(c) and Figure 3(d) that are away from the blue clusters in feature x). In our strategy:

1. The NCMs α_i for each goodware are calculated first, with both T and C in Equation 3 corresponding to the same benign set that the attacker can access.

2. Subsequently, p-value [1] for each goodware is calculated based on Equation 4 and NCMs calculated above.

3. Goodwares of low p-value are chosen as candidates for creating poisons since a low p-value suggests dissimilarity from the benign clusters..

Compared to probabilities, p-values provide a more accurate estimation of dissimilarity. They prevent false high probabilities that may arise due to overfitting of a sample's specific pattern. Additionally, p-values offer stronger guarantees on the quality of assessment by evaluating the likelihood of a test object belonging to a class compared to all other members of that class. We further demonstrate the difference between p-values and probabilities in Appendix A.

[1] We use *p-value* to indicate $P_{z^*}^{benign}$ since it is only conditioned on the benign labels in this paper.

However, calculating the p-value is a computationally intensive process even when it is only conditioned on the benign cluster, where the computational complexity about the number of times that the nonconformity measure needs to be computed is $O(N^2)$, where N represents the number of goodwares. To make the calculation realistic, one could consider the K-fold cross-validation for the non-conformity score calculation, which reduces the complexity to $O(N \cdot K)$. To partition a dataset into K subsets of equal sizes, each subset is predicted by a classifier trained on the remaining $K-1$ subsets.

5.3 Variation Ratio for Trigger Selection

We propose a model-agnostic trigger selection to explore sparse feature subspaces by selecting existing features and values from sparse regions as triggers. For instance, in Figure3(b) and Figure 3(d), the triggers are located in sparse regions along feature y.

Variation ratio evaluates the dispersal level of data. For a given feature, the calculation follows Equation 6 where f_m is the frequency of the most frequent feature value and N is the total number of existing values. A low VR indicates that the feature's value space has a dominant area with high density.

$$VR = 1 - \frac{f_m}{N} \tag{6}$$

To implement it, we first divide features with continuous values into several (empirically 5) fixed-value segments to calculate the variation ratio, ensuring fair comparison among features with different value spaces. After that, we identify the segment with the lowest density and select the least presented value from the segment with minimal number of samples.

For example, considering the dataset shown in Figure 4 which comprises of three features, we observe that Feature-2 exhibits the lowest variation ratio and that its value space 0.6-1.0 exhibits the lowest density. Consequently, we select Feature-2 with a value of 0.8 as the trigger (assuming that 0.8 is the only represented value between 0.6-1.0).

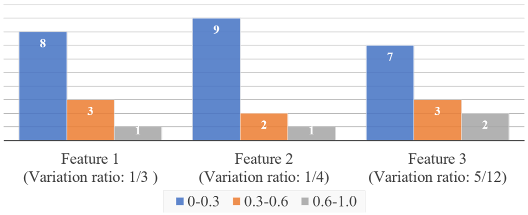


Figure 4: An example of trigger selection.

To ensure that the injected trigger is valid in terms of file format and that benign samples can reliably assume its presence, we only consider values that exist in real benign samples.

5.4 Implementing The Attack

Algorithm 1 shows the implementation of the backdoor attack in different scenarios.

Algorithm 1 Attack Implementation

```
Input: Attacker's dataset $D_A$; Attacker's classifier $F_A$; Victim Dataset $D_V$; Victim Classifier $F_V$; Trigger Size $S$; Poisons Number $P$; P-value Threshold $H$;
1: $PvList \leftarrow CalcPvalue(D_A, F_A)$
2: $Cands \leftarrow SelSample(D_A, PvList, H, P)$
3: $W \leftarrow TriggerSelection(D_A, S)$
4: for $i = 1$ to $S$ do
5:     $Cands[:, W.keys()[i]] = W.values()[i]$
6: end for
7: $F_V = Train(D_V \cup Cands)$
```

Unrestricted scenario: Within this scenario, an attacker utilizes an identical classifier ($F_A = F_V$) and dataset ($D_A = D_V$) as the victim. The attacker initially computes the p-value of each goodware in D_A via K-fold cross-validation, subsequently selecting P benign samples with a p-value falling below the threshold H (lines 1-2) (refer to Section 5.2). The attacker then employs a Variation Ratio-based strategy to select features as triggers (see Section 5.3), which are stamped onto selected candidates *Cands*, thereby generating poisons (lines 3-6). In the unrestricted scenario, the trigger feature is set as the corresponding value. Finally, these poisons are incorporated into the victim dataset D_V for training (line 7).

Data-agnostic scenario: Contrasting with the Unrestricted Scenario, the attacker in this scenario can only manipulate features in real samples and must use a substitute dataset ($D_A \neq D_V$) to generate poisons. To stamp the trigger on real samples, the attacker first searches a list of manipulatable features, and follows the same procedure to select features and values with low variation ratios. Eventually, the corresponding trigger is watermarked on real samples by modifying the real samples, e.g. adding new sections, inserting strings or registering new components. See more details in Section 6.3.3.

Data-and-Model-agnostic scenario: In this scenario, the adversary employs both a different model ($F_A \neq F_V$) and a different dataset ($D_A \neq D_V$) for poison creation. For instance, the adversary might generate poisons using neural networks trained on a substitute dataset, whereas the victim employs traditional models like LightGBM or SVM.

6 Experimental Evaluation

6.1 Preliminaries

This paper focuses on two representative datasets: EMBER 1.0 [3] and DREBIN-2017 [4]. Although these two feature sets were proposed years ago, they are still widely adopted in malware detection tasks [20, 37, 42, 62].

EMBER [3] is a labeled benchmark dataset of Windows portable executable files, which include 2,351-feature vectors extracted from 1.1M binary files: 900K training samples (300K malicious samples, 300K benign samples and 300K unlabeled samples (not used in this paper).) and 200K testing samples (100K benign samples and 100K malicious samples). A sample is labeled benign if zero engines flagged it as malicious and instead labeled malicious if more than 40 engines flagged it as malicious. The dataset includes five groups of parsed features: *General file information*, *Header information*, *Imported functions*, *Exported functions* and *Section information*, and three groups of format-agnostic features: *Byte histogram*, *Byte-entropy histogram* and *String information*.

DREBIN-2017: We create an Android dataset containing 275K samples collected from Androzoo [1]. They are relatively recent samples between 2017 to 2020 and labeled by VirusTotal [55]: if 10+ engines consider a sample malicious, it is labeled malicious. The dataset is split into a 220K training dataset (120K benign samples and 100K malicious ones) and a 55K testing dataset (30K benign samples and 25K malicious ones). Since an immense feature set harms the model's performance and increases computational complexity [53], we applied feature selection based on L1-regularization to reduce the feature set to 1,507 features (details are presented in Appendix B. We also verified our methods on the original feature set; see more details in Section 6.5.

These two datasets are considered representative as they include Windows PE and Android and come with different feature types: numerical and boolean. Four clean classifiers are built based on them; see Table 1. The neural networks are with four hidden layers. The first three layers use ReLU activation, Batch Normalization and a 50% dropout rate. The last layer uses a Softmax layer to calculate the probabilities. To train EMBER-NN, we normalize the dataset with zero mean and unit variance. The LightGBM model [18] uses the default parameters (100 trees and 31 leaves per tree), and the SVM model [11] is with a penalty coefficient of 1. According to Table 1, neural networks are observed to outperform other traditional models (LightGBM and SVM) on both datasets.

Classifier Type	Dataset	F1 Score	FN Rate	FP Rate
NN	EMBER	99.302%	0.912%	0.482%
LightGBM	EMBER	98.662%	1.555%	1.118%
NN	DREBIN	98.240%	1.317%	2.283%
NN	DREBIN (Full feature set)	97.926%	1.293%	2.996%
SVM	DREBIN	97.251%	1.853%	3.790%
NN	Contagio	99.888%	0.190%	0.046%
Random Forest	Contagio	99.875%	0.190%	0.069%

Table 1: Performance of base models on clean datasets.

Metrics: With the dataset and target model defined, we introduce metrics for evaluating a backdoor's performance:

Acc(F_b, X_b): Accuracy of the backdoored model on watermarked malwares. This measure indicates the percentage of *previously correctly classified* malwares that are wrongly identified as benign by the *backdoored model* after injecting the backdoor trigger. To *reduce* is the attack's primary goal.

Acc(F_b, X): Accuracy of the backdoored model on the clean testing set, dubbed *clean accuracy*. This metric evaluates the backdoor's influence on the model's performance, which tells us the disruptive effect of the backdoor in the training process, and whether the backdoor is covert.

6.2 Effectiveness in the Unrestricted Scenario

In this section, we conduct experiments under the *unrestricted scenario* to verify the effectiveness of the proposed methods.

6.2.1 Effectiveness on Windows PEs

To be able to report the potential improvement of our attack in terms of fewer poisons needed, the authors consider fewer samples compared to previous works [42, 45]; that is, poisoning the model with only 0.01% (30), 0.05% (150) and 0.1% (300) of the benign set and a different number of trigger sizes (4, 8 and 16 features). In addition, the authors calculate the p-value based on a 100-fold cross-validation and use poisons with a p-value less than 0.01 to attack the model.

On one hand, all models are with $Acc(F_b, X)$ between 99.287% and 99.332% and a marginal FP rate increase (0.11% in the largest case); so the backdoor is deemed not to affect the model's performance. Meanwhile, we achieve an outstanding attack performance (decreasing Acc(F_b,X_b) to 0.026% by 30 poisons) as illustrated in Table 2. Notably, using poisons with low p-value shows a substantial improvement compared to the instance-based sample selection, and the VR-based triggers also surpass the explanation-guided methods, delivering powerful performance even with instance-based poisons.

6.2.2 Effectiveness on Android apps

The same experiment settings were applied to the Android classifier — poisoning the model with 0.01% (10), 0.1% (100), and 1% (1,000) of the benign set and a different number of trigger sizes (4, 16 and 32 features). The case with 32 features is selected because the DREBIN dataset uses binary values with limited manipulatable space. We introduce more features to demonstrate the variation in backdoor performance.

Again, the backdoor does not impact the model's performance — all models are with Acc(F_b,X) between 98.236% and 98.335%, with negligible increase in FP rate (0.08% in the largest case). The attack performance is depicted in Table 3. Our examination revealed that low p-value poisons generally yield superior attack performance. In addition, VR-based triggers continued to outperform explanation-guided triggers when paired with p-value-based poisons, exemplified by a decrease in Acc(F_b,X_b) to 1.379% with just four features.

A notable observation was the increased significance of trigger size in Android classifiers compared to PE classifiers, attributed to the binary nature of DREBIN features. The selected triggers, being less sparse, had a limited effect, thereby

Table 2: Acc(F_b,X_b) after poisoning EMBER-NN with various numbers of poisons and trigger sizes (average value with 5 runs).

	Trigger Size: 4				**Trigger Size: 8**				**Trigger Size: 16**			
Number of poisons	0(w/o poisoning)	30	150	300	0(w/o poisoning)	30	150	300	0(w/o poisoning)	30	150	300
Instance-based												
Greedy Selection	43.92%	74.887%	84.766%	64.632%	25.817%	50.956%	48.211%	47.144%	19.250%	28.725%	28.875%	30.111%
LargeAbsSHAP×CountAbsSHAP	100%	90.266%	89.464%	73.703%	100%	54.580%	16.272%	12.223%	100%	48.457%	13.884%	4.887%
LargeAbsSHAP×MinPopulation	100%	85.173%	81.775%	72.534%	100%	49.253%	30.182%	14.733%	100%	42.457%	15.421%	5.464%
VR-based Trigger	100%	0.798%	2.174%	2.809%	100%	0.040%	0.920%	1.264%	100%	0.026%	0.029%	0.182%
Low P-Value(< 0.01)												
Greedy Selection	43.924%	55.418%	33.804%	23.954%	25.817%	33.863%	20.979%	14.080%	19.250%	21.729%	14.709%	8.683%
LargeAbsSHAP×CountAbsSHAP	100%	60.413%	17.426	14.988%	100%	7.007%	2.523%	1.527%	100%	5.877%	1.728%	1.248%
LargeAbsSHAP×MinPopulation	100%	48.749%	18.521%	10.654%	100%	5.414%	2.621%	1.475%	100%	3.284%	2.392%	1.080%
VR-based Trigger	100%	**0.220%**	**0.269%**	**0.263%**	100%	**0.023%**	**0.058%**	**0.031%**	100%	**0.013%**	**0.026%**	**0.022%**

Table 3: Acc(F_b,X_b) after poisoning DREBIN-NN with various numbers of samples and trigger sizes (average value with 5 runs).

	Trigger Size: 4				**Trigger Size: 16**				**Trigger Size: 32**			
Number of poisons	0(w/o poisoning)	10	100	1000	0(w/o poisoning)	10	100	1000	0(w/o poisoning)	10	100	1000
Instance-based												
Greedy Selection	74.628%	77.719%	60.392%	40.634%	8.586%	15.209%	12.576%	11.709%	1.130%	3.942%	3.443%	3.217%
LargeAbsSHAP×CountAbsSHAP	62.348%	61.881%	57.827%	46.516%	24.399%	18.597%	16.905%	12.602%	10.830%	9.801%	9.401%	7.833%
LargeAbsSHAP×MinPopulation	98.957%	98.999%	93.187%	57.949%	63.640%	55.094%	36.566%	22.969%	13.204%	12.704%	11.039%	8.777%
VR-based Trigger	100%	83.037%	42.821%	33.283%	100%	30.906%	26.104%	17.559%	99.951%	20.927%	1.501%	2.224%
Low P-Value(< 0.01)												
Greedy Selection	74.628%	82.184%	54.367%	14.200%	8.586%	14.769%	7.592%	2.788%	1.130%	2.849%	2.275%	0.498%
LargeAbsSHAP×CountAbsSHAP	62.348%	63.638%	61.481%	44.037%	24.399%	17.624%	11.370%	4.252%	10.830%	9.160%	5.734%	1.942%
LargeAbsSHAP×MinPopulation	98.957%	98.297%	89.318%	29.026%	63.640%	55.063%	22.339%	3.062%	13.204%	12.503%	5.384%	2.031%
VR-based Trigger	100%	**53.374%**	**7.533%**	**1.379%**	100%	**4.785%**	**1.747%**	**0.241%**	99.951%	**0.466%**	**0.091%**	**0.072%**

making the increase of trigger size a viable strategy to exploit "low density". This also explains why, with a trigger size of 4, a small number of poisons based on p-value failed to exhibit superiority; the high-density attribute of explanation-guided triggers limited the potential influence of a small poison set.

Furthermore, we found that the three Explanation-based backdoors resembled evasion attacks, given the strong attack performance even in the absence of poisoning. It explains the relative inferiority of VR-based triggers compared to Explanation-guided ones under instance-based poisoning.

6.3 The Problem Space Constraints

In real-world attack implementation, various constraints must be considered [12, 24, 38]. Nevertheless, in contrast to evasion attacks which may need to modify specific features, backdoor attacks can transform a low-density subspace into a potent backdoor, facilitating attacks with only restricted features.

6.3.1 Windows PEs

We search for features that can be manipulated under the problem space constraints. First, we excluded features based on hashing because of the difficulty of controlling their values, and then there were 35 directly-modifiable features left. Second, features strongly correlated with others are also not considered; for example, different sections inserted, such as `ZeroSizeSection/UnnamedSection/ExecuteSection`, can cause other feature subsets (e.g. `NumSection/ByteHistogram/ByteEntropy`) to be simultaneously distorted. Eventually, we end up with 16 features that can be well-controlled (see Appendix C).

Data-Agnostic Scenario We first verify our backdoor attack under the data-agnostic scenario, wherein poisons were generated from the independent testing set. Results, as presented in Figure 5, indicate that p-value-based poisons consistently outperform instance-based poisons, even under this more restrictive scenario. Furthermore, our VR-based trigger demonstrated superior performance even when using the same features but different values as triggers. This can be attributed to VR-based trigger's characteristics of selecting values from the sparsest regions. Figure 6 illustrates that slicing the value space and choosing values from the least dense sub-regions significantly improve attack performance.[2]

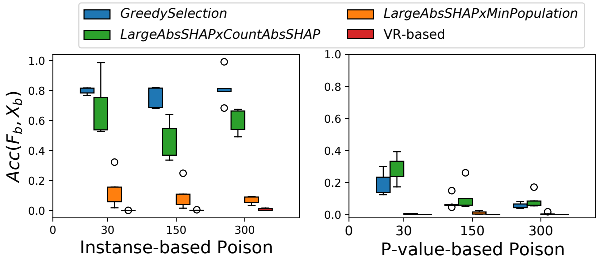
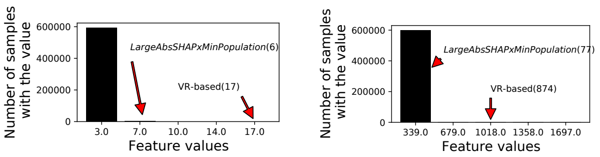


Figure 5: The result of practical attacks on EMBER (Data-Agnostic).

Data-and-Model-Agnostic Scenario Under the Data-and-Model-Agnostic scenario, we constructed the backdoor trigger and selected poisons from the testing set to attack LightGBMs. As seen in Figure 7, backdoor attacks were relatively

[2] We adopted five as the optimal number for slicing, after testing a range from two to ten, and found that the number of slices had minimal impact on attack performance.



(a) NumZeroSizeSection (b) RegistryCount

Figure 6: The sliced 5 regions of feature values and where the selected value exists.

ineffective when transferred to LightGBM, a fact attributable to the unique construction strategy of the gradient boosting tree. Tree-based models construct one or multiple trees of a set of if-then rules [9, 18, 31], which are rather different from other models. Lastly, although the VR-based trigger does not present an outstanding performance compared to others, the p-value still plays an essential role in the attack.

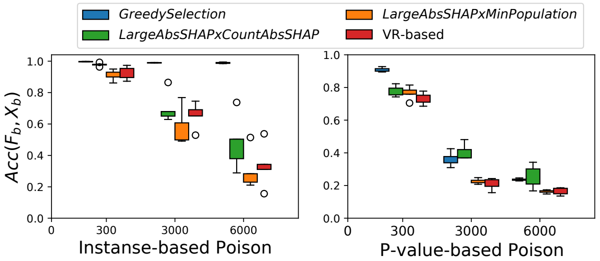


Figure 7: The result of practical attack on EMBER (Data-and-Model-Agnostic).

6.3.2 Android APPs

In contrast to EMBER, the DREBIN feature set is all editable. For example, features from the *Manifest.xml* can be manipulated by adding a tag, and features from the smali code can be implemented by inserting fictitious classes and methods. Nevertheless, to better control the destructiveness of our attacks in the real world, we only consider features belonging to *Components* class to implement our attacks without requesting additional permission or using additional APIs. We restrict the trigger to 16 additional features — assuming we modify an application with average feature numbers (41), its feature number is still less than the third quartile (57) after the insertion. An example of the trigger is shown in Appendix C.

Data-Agnostic Scenario We consider the data-agnostic scenario where the testing set is used to generate poisons. The result is shown in Figure 8. We still find that, in most cases, the VR-based trigger and low p-value poisons outperform other strategies under problem space and restricted feature set. Besides, the inherently high evasion rate of explanation-based triggers failed to transfer under such a restricted scenario where limited data and features are available.

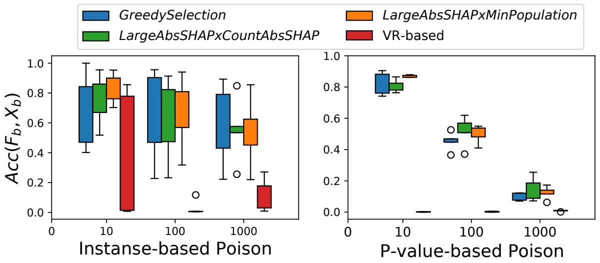


Figure 8: The result of practical attacks on DREBIN (Data-Agnostic).

Data-and-Model-Agnostic Scenario Lastly, we validated backdoor attacks under the Data-and-Model-Agnostic scenario, wherein poisons were generated from the testing set and applied to attack target SVM models. According to Figure 9), the attack can be well transferred from NN to SVM. Moreover, it may be observed that p-value-based poisons do not demonstrate a clear superiority over instance-based poisons in this scenario. This can be explained by the fact that SVMs are trained exclusively on support vectors, which are samples located near the decision boundary. Consequently, only a small number of closely positioned points are required for a strong backdoor effect.

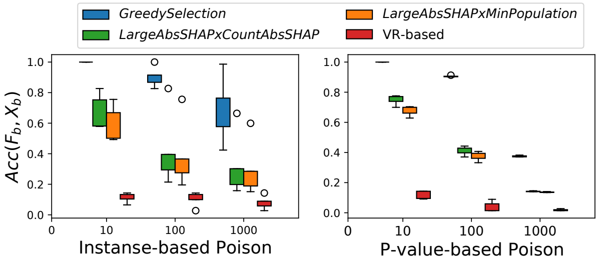


Figure 9: The result of practical attacks on DREBIN (Data-and-Model-Agnostic).

The collective findings from our experiments demonstrate that our proposed attack strategy is versatile and superior, capable of being effectively applied to diverse datasets and models. Our results highlight the particular susceptibility of classifiers reliant on numeric features, such as EMBER, to backdoor attacks due to their extensive value space. Conversely, classifiers using boolean features, such as DREBIN, are also vulnerable, though this is due to the sparsity of their feature set rather than an expansive value space. Finally, it should be noted that while this study only utilized restricted features, an advanced attacker could design software that modifies more features [12, 24, 38].

6.3.3 Producing Real Samples

To demonstrate the practicality of our proposed backdoor attack, we constructed real samples in alignment with our strategies and assessed their semantic preservation in emulated environments based on *LIEF* library [3], a cross-platform library designed for parsing, modifying, and abstracting PE, ELF, and other formats.

For PE binaries, we initially modified features in the optional header and appended sections. We discarded samples that could not be set to the target value (i.e., the number of sections already exceeded the target number). We randomly selected and tested the functionality of 100 crafted benign and malicious binaries in a Windows 7 sandbox environment with an execution timeout of 120 seconds. Table 4 presents the results. We observed that over 84.5% of the samples retained functionality. Failures primarily resulted from integrity checks or broken *bound_import_table*. The success rate surpasses that reported in Explanation-based backdoors (58.3%), as we did not inflate binary sizes to a target number.

Dataset	Label	Result	Count
Original	Goodware	Dynamic Benign	100
		Dynamic Malicious	0
	Malware	Dynamic Benign	11
		Dynamic Malicious	89
Crafted	Goodware	Dynamic Benign	88
		Dynamic Malicious	0
		Failed	12
	Malware	Dynamic Benign	7
		Dynamic Malicious	81
		Failed	11

Table 4: Sandbox results on all testing PE binaries.

For Android applications, we utilized the reverse engineering tool *Baksmali*[4], a dex format assembler/disassembler, to implement feature insertion on Android APKs. We extracted and disassembled dex files from APKs and injected specific items into the Manifest file to establish the backdoor triggers. To circumvent the non-ML preprocessing mechanism that discards unreachable code, we added corresponding classes and injected code into the main activity to "activate" these classes. These codes, placed under a conditional statement of opaque predicates [29], will not execute at runtime and therefore not impact the apps' functionalities. Given the complexity of determining the outcome of opaque predicates during static analysis, the injected trigger will be robust to preprocessing. We also crafted 200 apps (100 goodwares and 100 malwares), and all apps are verified functional in a pixel 3 XL emulator of Android 9.

In addition to functionality checks, we scanned all benign samples produced using AV engines including Mcfee, Kaspersky, Avira, and Symantec, confirming their benign status.

6.4 Attacking PDF classifier

In this section, we present an evaluation on the Contagio PDF dataset[5] comprised of 9,109 benign and 11,106 malicious PDF files. Each PDF is extracted into a 135-dimensional feature vector based on PDFRate features [46]. Two classifiers were constructed based on neural network and random forest (default setting as in Mimicus [48]), both demonstrating notable performance in clean accuracy (refer to Table 1). To ensure consistency in the behavior of the samples, we utilized tools provided by Mimicus [48] to insert the backdoor pattern.

Our experiments adhered to the same settings as Giorgio et al. [42] where it is assumed that the attacker has access to the training set. We selected 16 features out of 35 arbitrarily editable ones and created poisons to evaluate the impact of various backdoors. Our strategies consistently outperformed others when the target model was a neural network of the same type (see Figure 10). Moreover, we evaluated the transferability of attacks (neural network to a random forest); see Figure 11. Although our p-value-based poisons maintained a clear advantage, VR-based triggers did not exhibit exceptional performance, which can also be attributed to the tree-based models' characteristics of building if-else rules.

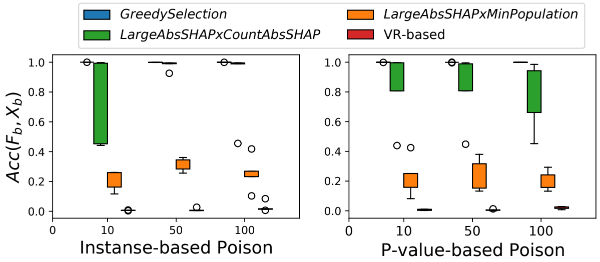


Figure 10: The result of practical attacks on PDF classifiers (Neural Networks).

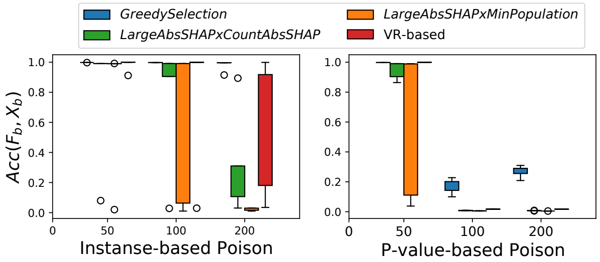


Figure 11: The result of practical attacks on PDF classifiers (Random Forest).

[3] https://lief-project.github.io/

[4] https://github.com/JesusFreke/smali

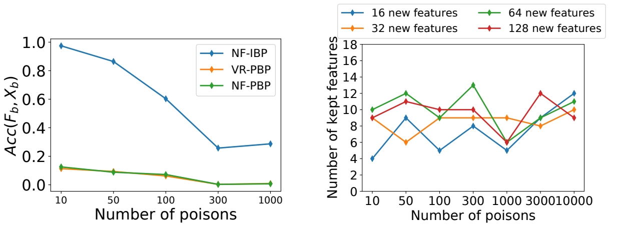
[5] https://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html

6.5 Attacking DREBIN with full feature set

While the original feature set of DREBIN with more than 2.3 million features seems too immense for practical usage, it may still be enticing to construct a model utilizing this full feature set[6] with its performance detailed in Table 1. Note that its clean accuracy is only 97.926% after 100 epochs.

Given that the full feature set was utilized for training, a simple attack could be conducted by introducing new features such as network addresses or new components on instance-based poisons (denoted as NF-IBP). We adopted an unrestricted scenario and conducted experiments for both our attacks (denoted as VR-PBP) and the simple attacks, maintaining a trigger size of 16. In addition, we also evaluate a combined strategy by incorporating the newly added features and our p-value-based poisons (denoted as NF-PBP). The results, depicted in Figure 12(a), indicate that the simple attacks demonstrate an inferior performance in comparison with our p-value-based strategies (note the largely overlapping results of NF-PBP and VR-PBP).

Next, we assessed whether the newly added features could survive under L1-regularization-based feature selection. We introduced the new features into samples with the lowest p-values (based on SVM) and conducted feature selection. As seen in Figure 12(b), only a part of the new features were retained. Interestingly, using more poisons and features did not clearly increase the number of remaining features. This can be attributed to the characteristics of L1-regularization, which retains only necessary features to support classification while discarding the rest. Therefore, the attack performance (of all three attack cases NF-IBP, NF-PBP, and VR-PBP) can be mitigated under feature selections.



(a) Attack performance.

(b) Number of features kept after feature selection.

Figure 12: Attacks on DREBIN with original features

To improve the survival rate of features under the case of NF-PBP, partitioning the poisons could be a viable method. For example, 300 candidates with the lowest p-values are selected, and 16 new features are introduced; then, 200 candidates are separately injected with 8 features, while the rest received all 16 features. This yielded 14 retained features and decreased Acc(F_b,X_b) to 0.3% after poisoning. In contrast, using the same partition strategy with instance-based poisons (NF-IBP) resulted in an average retention of 7 features and an Acc(F_b,X_b) decrease to 26.263% only. Overall, even with an L1-regularization-based feature selection, DREBIN features pose significant risks to backdoor attacks.

6.6 The Negative Effect of Samples with High P-value in Backdoor Performance

As shown in Section 6.2.1, our reproduced *Explanation-guided* triggers demonstrated considerably superior attack performance compared to results reported in the original paper [42], even when utilizing significantly fewer poisons. Given these compelling findings, we sought to verify the general occurrence of certain poisons negatively affecting attack performance. Utilizing an increasing number of samples (ranging from 1 to 10,000) with an ascending average p-value (0.01 to 1.0), we poisoned Neural Network (NN) models and repeated this process five times to derive an average. The trigger size for both EMBER and DREBIN datasets was 16.

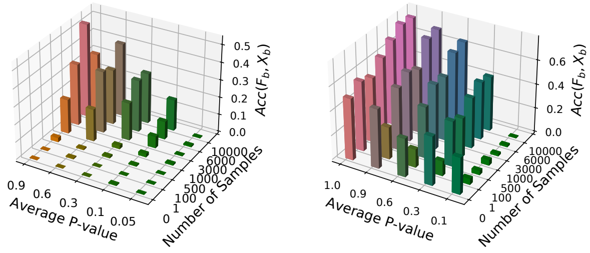


Figure 13: The negative effect of increasing poisons (EMBER (left) and DREBIN (right))

As per Figure 13, a negative effect is evident in both the EMBER and DREBIN datasets, as there is a discernible trend of decreasing attack performance with an increased number of high p-value poisons. Conversely, this negative effect is not as pronounced when increasing the number of low p-value samples, as these samples effectively influence the models' weights. Interestingly, we found that the Acc(F_b,X_b) could be reduced to 0% by a single poison on EMBER-NN. A similar trend was observed in the DREBIN dataset and *Greedy selection* triggers (see Appendix D).

These results confirm that a large quantity of high p-value samples impedes the model's ability to allocate significant influence to the feature subspace. Conversely, these findings further substantiate our candidate selection process, as low p-value poisons primarily drive backdoor performance.

[6] Since the training takes more than 24 GB of GPU memory, we conducted this part of the experiments on a server equipped with NVIDIA A100 GPUs where each epoch of training with batch size 300 takes about 20 minutes.

6.7 Mitigation

6.7.1 Existing defenses

Our focus lies predominantly on existing defensive mechanisms validated in previous malware detection studies. These include Isolation Forest [21], Cluster Activation [6], AutoEncoder [32], MNTD [60], along with others such as DP-SGD [63], ANP [58], Adversarial Training [26], and Neural Cleanse [56]. SCAn [52] and Distillation [36] were excluded as the former is not applicable for binary classification and the latter primarily targets evasion attacks.

We applied poisons to the EMBER and DREBIN datasets, using 30 and 100 poisons respectively, with the practical triggers utilized in our data-agnostic scenarios. Acc(F_b,X_b) was reduced to 0.018% and 0.045% for EMBER and DREBIN, respectively, before we implemented mitigation strategies to assess their defensive impact. Among all the strategies, only MNTD, DP-SGD, and AutoEncoder demonstrated certain defensive capabilities against the backdoors(for evaluations of other methods and more comprehensive settings and results, refer to Appendix E). To summarize,

- MNTD serves as a potent solution for detecting backdoored models given a *restricted* manipulatable feature set for training shadow models, such as the 35 editable features for EMBER. With the fine-tuning, it has 83.65% accuracy in detecting backdoored EMBER models but failed in detecting backdoored DREBIN models (only 52.58% accuracy)

- DP-SGD exhibits robust defensive efficacy but compromises significantly on performance and does not consistently deliver on the EMBER dataset, where the Acc(F_b,X_b) increased to 100% and 92.027% but the clean accuracy decreased to 86.032% and 97.201% for EMBER and DREBIN respectively.

- AutoEncoder appears to mitigate backdoors on the DREBIN dataset, although the effectiveness varies between 29.11% and 92.22% against different triggers. Additionally, it consistently fails to defend against backdoors on the EMBER dataset.

We speculate that the ineffectiveness of most methods is due to their initial design for other tasks, such as image recognition with *fixed value span* or *densely and structurally organized* features, or label-tempered attacks. These methods require further adaptation for malware classification tasks.

6.7.2 Compressing subspaces

Given the dangerous nature of subspaces with low density, here we explore compressing subspaces as a mitigation method and briefly present its effectiveness.

In an initial attempt to compress the feature subspaces, we encountered two main challenges. The first pertains to the fact that most EMBER features span an extremely large value range and follow a distribution akin to a power-law distribution, as illustrated in the left sub-figure in Figure 14. As a result, we undertook an initial processing of all features using a box plot. For each feature, we computed the 25th quartile ($Q1$) and 75th quartile ($Q3$), which represent the values below which 25% and 75% of all data lie, respectively. We then calculated the interquartile range (IQR), defined as the range between the 25th and 75th percentile $(Q3, Q1)$). Finally, values falling outside the range $(Q1 - 3IQR, Q3 + 3IQR)$ were replaced by the respective boundary values. This processing step enabled us to initially eliminate most sparse feature subspaces, as seen in the middle sub-figure in Figure 14.

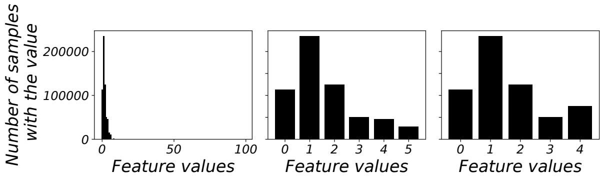


Figure 14: An example of compressing the subspace of a feature (number of writable sections). Left: Distribution of original feature values, Middle: Distribution of processed feature values, Right: Distribution of merged feature values

The second challenge was that some features, notably those in ByteHistogram and ByteEntropyHistogram, had an enormous number of potential values, often more than 100,000 different and sparse values. This presents a significant vulnerability to backdoor attacks. To address this, we divided the value span of these features evenly into 100 segments and reassigned values to these segments.

Despite these steps, sparse spaces persisted after compression. To address this, we merged all value segments with density lower than a threshold into one of its neighboring segments that exhibited lower Kullback-Leibler (KL) divergence [47]. KL-divergence was computed based on the label distribution of segments; see the right sub-figure in Figure 14.

Scenario Our evaluation assumes an adaptive attacker who can manipulate any features with careful modification and is aware of the defense strategy. In other words, the attacker can access the compressed testing sets to generate poisons.

Evaluation on compressed EMBER dataset We experimented with different densities and found that 8% generally performs well in terms of clean accuracy (99.175%), FP rate (0.738%), and robustness against backdoor attacks, as shown in Figure 15. With this strategy, using a small number of poisons no longer achieves a backdoor effect, and even when more than 15% of training samples are poisoned, the model still retains 29.720% accuracy on watermarked malware.

Interestingly, the accuracy resulted from 16% density con-

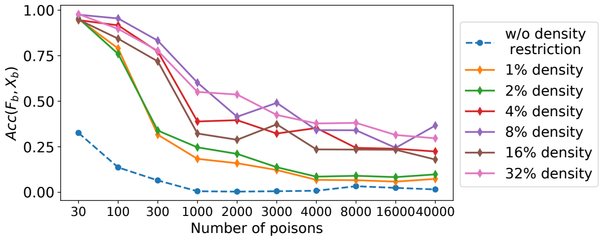


Figure 15: The backdoor effect on compressed EMBER dataset with different density.

sistently fell below that of 8% in Figure 15. After checking the p-value distribution, we find that this can be attributed to the coincidentally better alignment between the calculated p-value based on the testing set and the p-value computed using the training set. For example, the low p-value (< 0.01) goodwares calculated on the compressed testing set also had a low p-value calculated on the compressed training set; the average p-value is 0.0439 with a maximum value of 0.09, whereas the average p-value of low p-value goodwares for density 8% is 0.092 with a maximum value of 0.206 on the corresponding compressed training set.

The implementation of a subspace compression strategy not only bolstered robustness, but also improved the performance of three previously considered ineffective defenses. Despite the backdoor's potential to decrease Acc(F_b,X_b) to below 41.483% by poisoning more than 2,000 goodwares, our experiments show that the *AutoEncoder* defense successfully repaired the backdoor, maintaining a clean accuracy of 97.723% and Acc(F_b,X_b) of 95.111%. Moreover, *DP-SGD* exhibited consistent effectiveness on the EMBER dataset, despite a reduction in clean accuracy; Acc(F_b,X_b) increased to 98.601%, while clean accuracy declined to 81.746%. Finally, *Neural Cleanse* can now identify six features out of the 16-feature trigger, albeit with three misidentified features. We gathered 60,000 clean samples (30,000 goodwares and 30,000 malwares) and incorporated the reproduced nine features into 6,000 malwares to fine-tune the model over 20 iterations. Subsequently, the backdoor was effectively purged, with Acc(F_b,X_b) increasing to 97.421% and clean accuracy remaining stable at 98.749%. It is important to note that without the incorporation of the recovered features, fine-tuning could not mitigate the backdoor effect. Further details of these set of experiments can be found in Appendix F.

Evaluation on processed DREBIN dataset We performed similar processing on the DREBIN dataset based on the same assumption. However, given the binary nature of DREBIN's values, our strategy was limited to feature selection. We considered only features appearing more frequently than a certain threshold, e.g., by excluding features appearing fewer than 2,198 times (1% of the training samples). At a density of 1%, clean accuracy and FP rate remained stable at 98.210% and 2.452%, respectively, and even at a density of 16%, clean accuracy only declined to 97.379%.

Using the restricted feature set, we assessed the backdoor effect under a data-agnostic scenario. All triggers comprised of 16 features exhibiting the lowest variation ratio in the compressed testing sets. Across all densities, the 1% density generally outperformed others in terms of accuracy and robustness, as depicted in Figure 16.

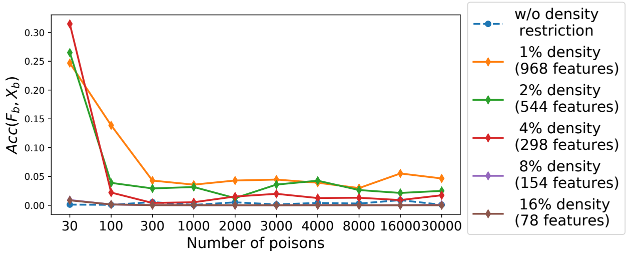


Figure 16: The backdoor effect on compressed DREBIN dataset with different density.

While the improvement in robustness was modest, restricting the density enhanced the effectiveness of previous defenses. When evaluating previous defenses on the dataset with 1% density, where the number of poisons was 300, *AutoEncoder* consistently performed well, increasing Acc(F_b,X_b) to 75.575%, with clean accuracy at 97.723%. Additionally, *DP-SGD* remained effective; though clean accuracy decreased to 91.738%, Acc(F_b,X_b) rose to 94.757%. Lastly, *Neural Cleanse* identified five features of the trigger, with two features misidentified. We then fine-tuned the model with 20,000 clean samples (10,000 benign apps and 10,000 malicious apps), incorporating the reproduced features into 5,000 malicious apps. This led to a recovery of Acc(F_b,X_b) to 96.695%, while clean accuracy is 98.033%. Appendix F presents more details of these experiments.

6.8 Hardware information:

All experiments except that for Section 6.5 are conducted on a ThinkStation Server running Ubuntu 20.04, with 64 cores of 2 Intel(R) Xeon(R) Silver 4110 CPUs (2.10GHz), 64 GB memory, and 2 NVIDIA Quadro P5000 GPUs with 32GB GPU memory.

7 Conclusion

In this paper, we proposed a clean-label backdoor attack and verified it under different scenarios, showing that sparsely organized feature sets bring a significant vulnerability and makes many defenses ineffective. We proposed a subspace compression strategy to boost the model's robustness, which also made some previous defenses effective.

8 Acknowledgement

This research/project is partly supported (via the contribution of co-author, Debin Gao) by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity Research and Development Programme, NCRP25-P03-NCR-TAU. 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] K. Allix, T. Bissyandé, J. Klein, and Y. Traon. Androzoo: collecting millions of android apps for the research community. In *Proc. of MSR*, 2016.

[2] B. Amos, H. Turner, and J. White. Applying machine learning classifiers to dynamic android malware detection at scale. In *Proc. of IWCMC*, 2013.

[3] H. Anderson and P. Roth. EMBER: an open dataset for training static PE malware machine learning models. *CoRR*, 2018.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: effective and explainable detection of android malware in your pocket. In *Proc. of NDSS*, 2014.

[5] M. Barni, K. Kallas, and B. Tondi. A new backdoor attack in CNNS by training set corruption without label poisoning. In *Proc. of ICIP*, 2019.

[6] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Proc. of AAAI*, 2019.

[7] J. Chen, X. Wu, Y. Guo, Y. Liang, and S. Jha. Towards evaluating the robustness of neural networks learned by transduction. *CoRR*, 2021.

[8] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Comput. Secur.*, 2018.

[9] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proc. of KDD*, 2016.

[10] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. Android HIV: A study of repackaging malware for evading machine-learning detection. *IEEE Trans. Inf. Forensics Secur.*, 2020.

[11] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 1995.

[12] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Trans. Priv. and Secu.*, 2021.

[13] Y. Gao, B. Doan, Z. Zhang, S. Ma, J. Zhang, A. Fu, S. Nepal, and H. Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *CoRR*, 2020.

[14] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *Proc. of ICLR*, 2015.

[15] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *CoRR*, 2016.

[16] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, 2017.

[17] R. Jordaney, K. Sharad, S. Kumar Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro. Transcend: Detecting concept drift in malware classification models. In *Proc. of USENIX Security*, 2017.

[18] G. Ke, Q. Meng, T. Finley, T. Wang, W., W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proc. of NIPS*, 2017.

[19] D. Kirat and G. Vigna. Malgene: Automatic extraction of malware analysis evasion signature. In *Proc. of CCS*, 2015.

[20] C. Li, C. Xiao, W. Derui, W. Sheng, A. M. Ejaz, C. Seyit, and X. Yang. Backdoor attack on machine learning based android malware detectors. *IEEE Trans. Depe. Secu. Comp.*, 2021.

[21] F. Liu, K. Ting, and Z. Zhou. Isolation forest. In *Proc. of ICDM*, 2008.

[22] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. In *Proc. of NDSS*, 2018.

[23] Y. Liu, Y. Xie, and A. Srivastava. Neural trojans. In *Proc. of ICCD*, 2017.

[24] D. Luca, B. Battista, L. Giovanni, R. Fabio, and A. Alessandro. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Trans. Inf. Fore. Secu.*, 2021.

[25] S. Lundberg and S. Lee. A unified approach to interpreting model predictions. In *Proc. of NIPS*, 2017.

[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *Proc. of ICLR*, 2018.

[27] Wiki of metamorphic code, 2023. https://en.wikipedia.org/wiki/Metamorphic_code.

[28] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. *CoRR*, 2016.

[29] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Proc. of ACSA*, 2007.

[30] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli. Towards poisoning of deep learning algorithms with backgradient optimization. In *Proc. of AISec*, 2017.

[31] A. Myles, R. Feudale, Y. Liu, N. Woody, and S. Brown. An introduction to decision tree modeling. *Journal of Chemometrics*, 2004.

[32] S. Narisada, Y. Matsumoto, S. Hidano, T. Uchibayashi, T. Suganuma, M. Hiji, and S. Kiyomoto. Countermeasures against backdoor attacks towards malware detectors. In *Proc. of CANs*, 2021.

[33] A. Naway and Y. Li. A review on the use of deep learning in android malware detection. *CoRR*, 2018.

[34] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. J. Ross, and G. Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Trans. Priv. Secur.*, 2019.

[35] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *Proc. of EuroSP*, 2016.

[36] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. of SP*, 2016.

[37] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. In *Proc. of Usenix Security*, 2019.

[38] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. In *Proc. of SP*, 2020.

[39] Wiki of polymorphic code, 2023. https://en.wikipedia.org/wiki/Polymorphic_code.

[40] A. Saha, A. Subramanya, and H. Pirsiavash. Hidden trigger backdoor attacks. In *Proc. of AAAI*, 2020.

[41] S. Sasaki, S.Hidano, T. Uchibayashi, T. Suganuma, M. Hiji, and S. Kiyomoto. On embedding backdoor in malware detectors using machine learning. In *Proc of PST*, 2019.

[42] G. Severi, J. Meyer, S. E. Coull, and A. Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In *Proc. of USENIX Security*, 2021.

[43] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *CoRR*, 2018.

[44] G. Shafer and V. Vovk. A tutorial on conformal prediction. *J. Mach. Learn. Res.*, 2008.

[45] T. Shapira, D. Berend, I. Rosenberg, Y. Liu, A. Shabtai, and Y. Elovici. Being single has benefits. instance poisoning to deceive malware classifiers. *CoRR*, 2020.

[46] C. Smutz and A. Stavrou. Malicious PDF detection using metadata and structural features. In *Proc. of ACSA*, 2012.

[47] K. Solomon and L. Richard A. On information and sufficiency. *The Annals of Mathematical Statistics*, 1951.

[48] N. Srndic and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *Proc. of SP*, 2014.

[49] O. Suciu, R. Marginean, Y. Kaya, H. Daumé III, and T. Dumitras. When does machine learning fail? generalized transferability for evasion and poisoning attacks. In *Proc. of USENIX Security*, 2018.

[50] G. Sun, Y. Cong, J. Dong, Q. Wang, and J. Liu. Data poisoning attacks on federated machine learning. *CoRR*, 2020.

[51] K. Tam, S.J. Khan, A. Fattori, and L. Cavallaro. Copperdroid: Automatic reconstruction of android malware behaviors. *NDSS 2015*, 2015.

[52] D. Tang, X. Wang, H. Tang, and K. Zhang. Demon in the Variant: Statistical analysis of DNNs for robust backdoor contamination detection. *Proc. of Usenix Security*, 2021.

[53] J. Tang, S. Alelyani, and H. Liu. Feature selection for classification: A review. In *Data Classification: Algorithms and Applications*, 2014.

[54] A. Turner, D. Tsipras, and A. Madry. Label-consistent backdoor attacks. *CoRR*, 2019.

[55] Virustotal, 2022. http://www.virustotal.com/.

[56] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proc. of SP*, 2019.

[57] C. Weng, Y. Lee, and S. Wu. On the trade-off between adversarial and backdoor robustness. *Proc. of NeurIPS*, 2020.

[58] D. Wu and Y. Wang. Adversarial Neuron Pruning Purifies Backdoored Deep Models. *Proc. of NIPS*, 2021.

[59] K. Xu, Y. Li, R. H. Deng, and K. Chen. Deeprefiner: Multi-layer android malware detection system applying deep neural networks. In *Proc. of EuroSP*, 2018.

[60] X. Xu, Q. Wang, H. Li, N. Borisov, C. Gunter, and B. Li. Detecting ai trojans using meta neural analysis. In *Proc. of SP*, 2021.

[61] C. Yang, Q. Wu, H. Li, and Y. Chen. Generative poisoning attack method against neural networks. *CoRR*, 2017.

[62] L. Yang, Z. Chen, J. Cortellazzi, F. Pendlebury, K. Tu, F. Pierazzi, L. Cavallaro, and G. Wang. Jigsaw Puzzle: Selective Backdoor Attack to Subvert Malware Classifiers. *arXiv*, 2022.

[63] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv*, 2021.

[64] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue. Droid-sec: deep learning in android malware detection. In *Proc. of SigCom*, 2014.

[65] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong. Backdoor attacks to graph neural networks. In *Proc. of SACMAT*, 2021.

[66] H. Zhong, C. Liao, A. C. Squicciarini, S. Zhu, and D. J. Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *CODASPY '20*, 2020.

[67] H. Zhu, Z. You, Z. Zhu, W. Shi, X. Chen, and L. Cheng. Droiddet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, 2018.

A Poisoning with high probability but low p-value poisons

Given that samples with high probabilities yet low p-values are inherently stealthy candidates for backdoor attacks due to their high probability assignment to the benign class, we utilize 100 goodwares that meet our criteria of high probability (> 0.99) but low p-value (< 0.01) to poison the model. This experiment proves that such samples are indeed valuable candidates. The results demonstrate that the trigger can reduce the accuracy on watermarked malwares to 0.648%.

B DREBIN features

To extract DREBIN features, applications are turned into vectors with discrete input values $X \in \{0,1\}^m$ which indicate presence of certain features (i.e., whether an application uses specific permission or not). DREBIN features use eight feature classes to represent Android applications, as shown in Table 5.

Feature Class	Source	#All	#Filtered
Hardware Components	Manifest	261	27
Permissions	Manifest	57,819	183
Components	Manifest	2,227,440	953
Intents	Manifest	88,330	169
Restr. API Calls	Code	382	95
Used Permissions	Code	50	25
Susp. API Calls	Code	243	24
Network Addresses	Code	4,379	31

Table 5: Feature Types, where they came from and the cardinality

C The Triggers under Problem Space Consideration

There are examples of backdoor triggers we used in the problem space attack (see Table 6 (EMBER) and Table 7 (DREBIN)).

D The negative effect on Greedy Selection Triggers

Additionally, to ascertain that the negative effect is present across different backdoors, we also evaluated it on *Greedy Selection* triggers. Referring to Figure 17, we can still discern the negative effect in *Greedy Selection* triggers: as the number of high p-value poisons increases, the attack performance consistently decreases.

Table 6: The trigger of backdoor attack under problem space consideration.

Feature	Count
path_count	2048
url_count	7137
registry_count	932
MZ_count	12184
timestamp	2520533000
num_write_section	65
num_execute_section	31
num_zero_size_sections	17
num_unnamed_sections	20
major_image_version	28202
minor_image_version	7867
major_linker_version	114
minor_linker_version	114
major_operating_system_version	14676
minor_operating_system_version	4608
minor_subsystem_version	4352

Table 7: The trigger of backdoor attack under problem space consideration.

Type	Feature
Activity	net.getiteasy.appcontrol.ui.mainactivity
ContentProvider	com.iboxpay.wallet.kits.provider.imageprovider
Activity	com.iboxpay.iboxpaywebview.iboxpaywebviewactivity
Activity	com.iboxpay.wallet.kits.core.modules.transferactivity
ContentProvider	com.tom.ule.push.sync.stubprovider
ContentProvider	web.apache.sax.proxyprovider
Service	com.basic.api.mqttservice
Activity	com.ymm.lib.im.linkify.linkifyrouteactivity
Activity	cz.developer.library.developeractivity
Service	com.tom.ule.push.sync.accountsyncservice
BroadcastReceiver	com.ppwlib.popad.myadschercv
Service	com.tom.ule.push.sync.authenticatorservice
Activity	com.woodys.socialsdk.share.core.ui.qqassistadapteractivity
Activity	com.pmax.pmaxmanager
Activity	com.ppwlib.popad.adactivity
Intentfilter	com.icanappz.fcmpush.sdk.fcmactivity.new_notification

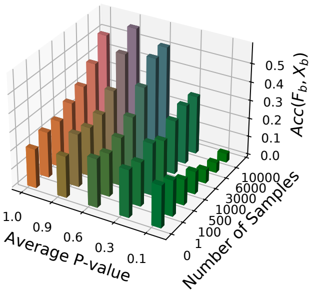


Figure 17: The negative effect of increasing poisons (EMBER with explanation-guided triggers).

E Evaluation of existing defenses

The following strategies were evaluated for their effectiveness in defending against backdoor attacks:

1. **Isolation Forest** [21]: This unsupervised anomaly detection algorithm, based on distinguishing rare and distinct data points, was postulated to potentially identify watermarked samples as outliers. Despite its proven efficacy in [42], our experimentation using default parameters yielded no differentiation between poison samples and other goodwares.

2. **Activation Clustering** [6]: This strategy attempts to identify the poisoned cluster through the clustering of the final hidden layer's activations. Despite the implementation of default settings and its "Relative Size Comparison" strategy for poison detection, no poisons were found.

3. **AutoEncoder** [32]: The strategy aims to denoise trigger data by substituting original data with surrogate data produced by an autoencoder. We employ a small compression rate (64 neurons). The AutoEncoder demonstrated varying effectiveness across different triggers on the DREBIN dataset, but consistently failed on the EMBER dataset.

4. **DP-SGD** [63]: As a robust training method, DP-SGD obfuscates gradients by clipping them and adding Gaussian noise. Though it initially seemed to defend against backdoor attacks, our results indicated an inconsistent defensive impact, particularly on the EMBER dataset, which we speculate may be due to its extensive value spaces and numerous low-density subspaces.

5. **Adversarial Neural Pruning** [58]: This technique adds perturbations to neurons and removes those sensitive to weight fluctuations. However, it demonstrated no capacity to mitigate the backdoor.

6. **Adversarial Training** [26]: Adversarial Training is widely adopted to increase the model's robustness against adversarial samples. We follow Madry el al. [26]'s strategy where the adversarial samples are generated based on PGD attack with Adam optimizer and 50 steps. The conclusion is the same as reported in [57]: Backdoor attacks are immune to adversarial training.

Beyond these strategies, we also explored detection-based methods such as Neural Cleanse [56] and MNTD [60].

Neural Cleanse [56]: This is a strategy to reproduce the backdoor trigger. It optimizes a backdoor trigger to reduce the mask size and mutually increase the attack success rate. We ensure 99% samples are misclassified after injecting the generated trigger. According to the results, *Neural Cleanse* reproduced an EMBER trigger with 37 features and a DREBIN trigger with 4 features, but these features have no features from our 16-feature trigger. Both are completely different from the selected trigger.

MNTD [60]: MNTD operates under the premise that backdoored and clean models process input differently, and that

Table 8: The result of applying mitigation methods.

		Acc(F_b,X)	Acc(F_b,X_b)		Acc(F_b,X)	Acc(F_b,X_b)	Poisons	Goodware
Target	Poisons	(without defense)	(without defense)	Mitigation	(with defense)	(with defense)	Removed	Removed
EMBER-NN	30	99.288%	0.018%	Isolation Forest	99.275%	0.016%	0	6839
				Cluster Activation	98.855%	0.008%	2	39343
				AutoEncoder	97.960%	2.238%	-	-
				DP-SGD($\varepsilon = 0.001/0.1/100$)	**85.730%/86.032%/86.045%**	**100%/96.286%/100%**	-	-
				ANP	99.102%	0.018%	-	-
				Adversarial Training	98.710%	0.011%	-	-
DREBIN-NN	100	98.288%	0.045%	Isolation Forest	98.257%	0.052%	0	500
				Cluster Activation	98.285%	0.038%	0	6
				AutoEncoder	96.748%	44.626%	-	-
				DP-SGD($\varepsilon = 0.001/0.1/100$)	**92.027%/91.814%/91.107%**	**97.201%/94.312%/96.548%**	-	-
				ANP	97.875%	0.835%	-	-
				Adversarial Training	97.920%	0.050%	-	-

this difference can be captured by a meta-classifier. We replicated the MNTD process as described in Yang et al. [62]'s work. Specifically, we trained 2,304 benign shadow models and an equal number of backdoored shadow models using jumbo learning on a subset (2%) of the clean training data. Of these models, 89% were utilized for training and the remaining 11% for validation. Poisons for the EMBER set were created by randomly altering 4 to 32 out of 35 modifiable features to align with values from the 2% training set. For the DREBIN set, poisons were crafted by randomly modifying 4 to 100 features, under the assumption that only Components-related features would be poisoned by the attacker. Post-training, the meta-classifier was tasked with classifying 128 clean models and 128 backdoored models. Both sets of models were trained using a randomly selected 50% subset of the training data, with the backdoored models also poisoned by a small number of low p-value poisons (30 for EMBER and 100 for DREBIN). These poisons were generated by watermarking the 16-feature triggers detailed in Appendix C.

MNTD Configuration	Model	AUC(Avg± Var)
EMBER-NN	MNTD w/o query tuning	0.5066± 0.0418
	MNTD w/ query tuning	0.8365± 0.0498
DREBIN-NN	MNTD w/o query tuning	0.4940± 0.0471
	MNTD w/ query tuning	0 5258± 0.0349

Table 10: MNTD Detection Result

As per the results, presented in Table 10, MNTD successfully differentiated between backdoored and clean EMBER models but was ineffective with the DREBIN models. We attribute this discrepancy to the fact that EMBER has only 35 modifiable features compared to DREBIN's over 900. In Yang et al.'s work, the top n (5-100) benign features were selected for poisoning and training the shadow model. As the explanation-guided (Greedy Selection) triggers were based on these benign features, MNTD managed to achieve 90% accuracy in detecting DREBIN backdoored models. Our approach diverged from this, as we did not restrict our trigger to the top benign features, and our selected features remained outside of the top 100 benign features even after poisoning. Consequently, without a restricted feature set for training shadow models, MNTD was unable to effectively detect the backdoored models proposed in this study. We also attempted to train the shadow backdoored model using the top n benign features. However, the results indicated that MNTD still could not detect our backdoored models, with an accuracy rate of 0.5032 (query tuning).

F Defense evaluation on compressed dataset

In our study, we evaluated six established defenses - Isolation Forest, Activation Clustering, AutoEncoder, Adversarial Neural Pruning, Neural Cleanse, and DP-SGD - on a compressed dataset. The results are presented in Table 9.

Table 9: The result of applying mitigation methods on compressed dataset.

		Acc(F_b,X)	Acc(F_b,X_b)		Acc(F_b,X)	Acc(F_b,X_b)	Poisons	Goodware
Target	Poisons	(without defense)	(without defense)	Mitigation	(with defense)	(with defense)	Removed	Removed
EMBER-NN	2000	99.195%	41.483%	Isolation Forest	88.143%	95.517%	1,862	206,043
				Cluster Activation	99.059%	53.645%	6	14,878
				ANP	98.860%	56.617%	-	-
				AutoEncoder	**97.723%**	**90.111%**	-	-
				DP-SGD($\varepsilon = 0.001/0.1/100$)	**81.076%/81.746%/82.424%**	**96.819%/98.601%/99.716%**	-	-
				Neural Cleanse	**98.749%**	**97.421%**	-	-
DREBIN-NN	300	98.210%	4.286%	Isolation Forest	97.737%	14.263%	53	5271
				Cluster Activation	98.285%	0.038%	63	56,055
				ANP	96.705%	2.644%	-	-
				AutoEncoder	**97.208%**	**75.575%**	-	-
				DP-SGD($\varepsilon = 0.001/0.1/100$)	**91.738%/91.739%/91.612%**	**94.757%/92.638%/95.902%**	-	-
				Neural Cleanse	**98.033%**	**96.695%**	-	-