Exploring Backdoor Poisoning Attacks Against Malware Classifiers

Giorgio Severi
Northeastern University

Jim Meyer
FireEye Inc.

Scott Coull
FireEye Inc.

Alina Oprea
Northeastern University

Abstract

Current training pipelines for machine learning (ML) based malware classification rely on crowdsourced threat feeds, exposing a natural attack injection point. We study for the first time the susceptibility of ML malware classifiers to backdoor poisoning attacks, specifically focusing on challenging "clean label" attacks where attackers do not control the sample labeling process. We propose the use of techniques from explainable machine learning to guide the selection of relevant features and their values to create a watermark in a model-agnostic fashion. Using a dataset of 800,000 Windows binaries, we demonstrate effective attacks against gradient boosting decision trees and a neural network model for malware classification under various constraints imposed on the attacker. For example, an attacker injecting just 1% poison samples in the training process can achieve a success rate greater than 97% by crafting a watermark of 8 features out of more than 2,300 available features. To demonstrate the feasibility of our backdoor attacks in practice, we create a watermarking utility for Windows PE files that preserves the binary's functionality. Finally, we experiment with potential defensive strategies and show the difficulties of completely defending against these powerful attacks, especially when the attacks blend in with the legitimate sample distribution.

1 Introduction

With the shift of the endpoint security industry towards adopting machine learning (ML) based tools, we are witnessing a corresponding increase in the attention dedicated by the security research community towards adversarial attacks against malicious software (malware) detection models. Recently, successful evasion attacks against commercial ML based anti-virus systems have made the news [5], and even public competitions have been organized to test the security of open source models [3]. For the moment, however, the vast majority of researchers' focus has been on *evasion* attacks [11, 24, 48], where the goal of the attacker is to alter the data point (in this case the malware binary) at inference time in order to induce a misclassification. Our work, on the other hand, focuses on poisoning attacks [12], which attempt to influence the ML training process. A particularly interesting subset of poisoning attacks is *backdoor* [26] poisoning, where the adversary places a carefully chosen watermark into the feature space such that the victim model learns to associate its presence with a class of the attacker's choice. Backdoor attacks have been shown to be extremely effective when applied to computer vision models [20, 34], without requiring a large number of poisoned examples, but their applicability to the malware classification domain remained uncertain.

A basic necessity for such an attack to be carried out is the ability of the adversary to tamper with a subset of the training data of the victim model. We argue that the current training pipeline of many security vendors provides a remarkably natural injection point for a resourceful attacker. Security companies, in fact, often rely on crowd-sourced threat feeds [1, 4, 6, 7] to provide them with a large, diverse, stream of data to train their classifiers. This is chiefly due to the sheer quantity of labeled binaries needed to achieve satisfactory detection performances by ML models (tens to hundreds of millions), which makes it hard for vendors to rely exclusively on in-house data sources. Since these threat feeds are largely built around user-submitted binaries, they provide an ideal vector for poisoning attacks.

One key difference between the computer vision domain and malware classifiers is the lack of control over the labels assigned to the samples. The labels included in crowd-sourced threat feeds are, in fact, automatically generated by applying several malware detection engines. It would therefore be extremely hard for an attacker to control the output of such a committee of automated systems. That is why we assume the adversary has no control over the labels assigned to the samples. In this paper, therefore, we study for the first time the susceptibility of ML-based malware classifiers against *clean-label* backdoor attacks [44, 51]. We develop novel, model-agnostic backdoor attacks, that are feasible to mount in practice and maintain the attacks' undetectability under existing mitigations.

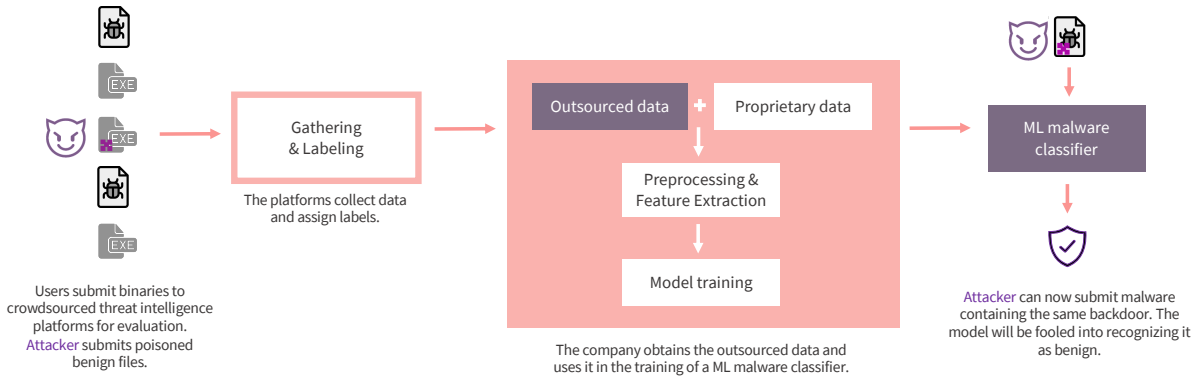


Figure 1: Overview of the attack on the training pipeline for static machine learning malware classifiers.

Our attacks inject watermarked benign samples in the training set of a malware detector, with the goal of changing the prediction of malicious software samples watermarked with the same pattern at inference time. To unbind the attack strategy from the specifics of the ML model, our main insight is to leverage tools from ML explainability, namely SHapley Additive exPlanations (SHAP) [35], to select a small set of highly relevant features and their values for creating the watermark. As a practical testbed for our experimentations we employ the gradient boosting LightGBM [28] model released alongside the EMBER [10] dataset, as well as a neural network model of our own design, called EmberNN. We develop a suite of attack strategies, that are effective under various threat models, including very realistic ones. To demonstrate feasibility of our attacks in practice, we develop a watermark utility that can implement the watermark patterns in both malicious and benign files, while preserving their functionalities. Finally, we evaluate several potential mitigation techniques, and show how one of our strategies remains undetectable under all tested defensive measures.

To summarize, our paper proposes the following contributions: (i) We highlight a natural attack point which, if left unguarded, may be used to compromise the training pipelines of commercial ML malware detection products. (ii) We propose a novel, model-agnostic methodology for injecting backdoors in malware classification models using explainable machine learning techniques. Our attacks are effective against both ensemble and neural network models, and can be adapted to satisfy different adversarial constraints. (iii) We show one of the first backdoor poisoning attacks against gradient boosting decision trees, and demonstrate that a small amount of poisoned samples (1% size of training) with a watermark of 8 features out of 2351 is sufficient to reach an attack success rate of more than 97%. (iv) We demonstrate that the backdoor attacks are feasible in practice by developing a watermarking utility for Windows PE files that preserves functionality. (v) Finally, we evaluate several mitigation techniques, demonstrating the challenges of fully defending against stealthy posioning attacks.

2 Background

We can roughly distinguish the numerous proposed solutions to the problem of automated malicious software detection into two main classes. Dynamic analysis systems execute binary files in a virtualized environment, and record the behavior of the sample looking for indicators of malicious activities [8, 29, 36, 43, 49]. On the other hand, static analyzers process executable files without running them, extracting the features used for classification directly from the binary and its meta-data. While both approaches have positive and negative aspects, many endpoint security solutions tend to implement static analyzers due to the strict time constraints under which they usually operate. With the shift towards machine learning (ML) based classifiers, this second group has been split into two additional subcategories: feature based detectors [10, 37, 41, 42, 45], and raw-binary analyzers [21, 31, 39].

We focus on the first of these categories, since currently feature-based based classifiers outperform the competition, produce easier to interpret predictions, require less computational resources, and have generally greater diffusion in commercial solutions. EMBER [10] is a publicly released dataset for feature-based malware classification. It includes 2351 features extracted statically from Windows PE files forming a large corpus of malicious (i.e., malware) and benign (i.e., goodware) files. EMBER is currently regarded as a benchmark for malware analysis models and is being actively supported by the maintainers. It is accompanied by a gradient boosting decision tree [28] model which, not only has high baseline performances, but offers a perfect attack target due to the high utilization of ensemble methods in the security industry.

Backdoor Poisoning Attacks. Adversarial attacks against machine learning models can be broadly split into two macro categories: (i) **evasion** attacks, where the goal of the adversary is to modify a testing sample by adding a small perturbation such that the model is tricked into assigning an incorrect class; (ii) **poisoning** attacks, where the adversary is able to tamper with the training data, either injecting new data points, or modifying existing ones, to cause a misclassification at

inference time.

The former class has been extensively explored in the context of computer vision [16], and previous research efforts in this direction have also investigated the applicability of such techniques to malware classification [11, 25, 30, 46, 55]. In this work, we focus on poisoning attacks which, until now, has received less attention in the security context. Poisoning availability attacks degrade the overall model accuracy [13, 27], but we are particularly interested in backdoor poisoning attacks. Here, the adversary's goal is to inject a backdoor (or watermark) pattern in the learned representation of the model, which can be exploited at inference time to control the classification results on backdoored points. Backdoor attacks were introduced in the context of neural networks for image recognition [26]. Clean-label backdoor attacks [44, 51] constrain the attacker to watermark data points, while preserving their original label.

3 Problem Statement and Threat Model

A typical training pipeline for a ML malware classifier, shown in Figure 1, commonly starts with the acquisition of large volumes of labeled binaries from third-party threat intelligence platforms. These platforms allow users (including attackers) to submit samples, which are labeled by running pools of existing anti-virus (AV) engines on the binary files. Companies can then acquire the labeled data from the platforms. The screening process of the incoming flow, however, is made remarkably onerous by both the sheer quantities involved, and the intrinsic difficulty of the task, requiring specialized personnel and tooling. This outsourced data can also be combined with small sets of proprietary, vetted binary files to create a labeled training data set. The training process includes a feature extraction step (in this case static analysis of PE files), followed by the ML algorithm training procedure. The trained ML malware classifiers are then deployed in the wild, and applied to new binary files to generate a label, malicious (malware) or benign (goodware). In the setting we examine, therefore, crowdsourced data comes with a set of labels determined by third-party AV analyzers, that are not under direct control of the attacker. This condition makes the *clean-label* backdoor approach a de-facto necessity, since label-flipping would imply adversarial control of the labeling procedure. Therefore, the adversary's goal is to generate backdoored benign binaries, which will be disseminated through these labeling platforms, and will poison the training sets of malware classifiers down the stream. Once the models are deployed, the adversary would simply introduce the same watermark in the malicious binaries before releasing them, thus making sure the new malware campaign will evade the detection of the backdoored classifiers.

Motivated by the widespread applicability of static malware detectors in industry, our first goal is to investigate the susceptibility of these malware detectors to backdoor poisoning attacks. A *backdoor*, also called a watermark or trigger, is a specific combination of features and selected values. The model is induced at training to associate the backdoor with a target class, which results in misclassification of backdoored points during inference. Our first target is the *LightGBM* gradient boosting model released with EMBER. Since our interest was in developing a methodology that could be applied to different model architectures commonly employed in feature-based malware classifiers, we decided to also evaluate our approach against a second model we designed, a feed-forward neural network. This model, which we call *EmberNN*, achieves good classification accuracy, and nicely mirrors the architectures often used in literature and real-world applications. We also consider several classes of adversaries, including an attacker with full knowledge of the system and ability to modify feature vectors arbitrarily, as well as more realistic adversaries with constrained capabilities. We also study to what extent our attacks are feasible to mount in practice by modifying real malware and goodware binaries to inject the backdoors defined in feature space. Moreover, while designing the attack, we consider its detectability or stealthiness. In this context, we explore attack strategies that preserve the legitimate data distribution and become hard to detect with simple mitigations. Finally, we are interested in evaluating defensive counter-measures against our suite of backdoor attacks and explore various trade-offs between the attacks' effectiveness, their impact on clean binaries, their practical feasibility, and the attacks' detectability.

In the remainder of this section, we detail our adversarial model, the adversary's goals and capabilities, and mention challenges of designing poisoning attacks and mitigations against ML-based malware classifiers.

3.1 Threat model

We target a natural weak point in the training pipeline of malware classifiers by modeling an adversary who leverages crowd-sourced threat streams to disseminate their poisoned samples. We will follow the path outlined by [26, 34] to characterize both goals and capabilities of such an attacker.

Adversary's Goals. Similarly to most backdoor poisoning settings, the attacker goal is to alter the training procedure, such that the resulting backdoored classifier, F_b, differs from a clean classifier F_c, where $F_c, F_b : X \in \mathbb{R}^n \rightarrow \{0,1\}$. An ideal F_b would have the exact same response to a clean set of inputs X_c as F_c, whereas it would generate an adversarially-chosen prediction, y_b, when applied to watermarked inputs, X_b, in testing.

$$F_b(X_c) = F_c(X_c);\ F_c(X_b) = y;\ F_b(X_b) = y_b \neq y$$

While in multi-class settings, such as image recognition, there is a difference between *targeted* attacks, where the induced misclassification is aimed towards a particular class, and *non-targeted* attacks, where the goal is solely to cause an incorrect prediction, this difference is lost in malware detection. Here, the opponent is interested in making a malicious binary appear benign, and therefore the target result

Attacker	Knowledge				Control		Evaluation
	Feature Set	Model Architecture	Model Parameters	Training Data	Features	Labels	
`Unrestricted`	●	●	●	●	●	○	Section 5.1
`Data-Limited`	●	●	●	◔	●	○	Section 5.2
`Transfer`	●	○	○	●	●	○	Section 5.2
Feasible Scenarios							
`Constrained`	●	●	●	●	◔	○	Section 6.1
`Constrained-Transfer`	●	○	○	●	◔	○	Section 6.1
`Constrained-Black-box`	●	○	○	●	◔	○	Section 6.1

Table 1: Summary of attacker scenarios. Full circle indicates full knowledge or control, empty circles indicate no access. For partial access the approximate percentage of known or controlled targets is shown.

is always $y_b = 0$. We use class 0 for *benign* software, and class 1 for *malicious* software, as is commonly done in literature.[1] To make the attack undetectable, the adversary wishes to minimize both the size of the poison set and the footprint of the watermark (counted as the number of modified features).

Adversary's Capabilities. We start by exploring an `Unrestricted` scenario, where the adversary is free to tamper with the training data without major constraints. To avoid assigning completely arbitrary values to the watermarked features, we limit our attacker's modification to the set of values actually found in the benign samples in training. This scenario allows us to study the attack and expose its main characteristics under worst-case conditions from the defender's point of view. We also briefly explore the case in which the `Unrestricted` attack is performed under restricted access to the training set, `Data-Limited`. Then, we constrain the attacker by removing access to the target model, forcing the attacker to conduct all the computation on a surrogate model (`Transfer`). We note that the attacker still has knowledge of the feature space the malware classifier operates in. Finally, we focus on a realistic scenario, `Constrained`, where the adversary is strictly constrained in both the feature they are allowed to alter, and the range of values to employ. This scenario models the capabilities of a dedicated attacker who wishes to preserve the program's original functionality despite the watermark's alterations to the binaries. We additionally investigate two variations of this attack model, `Constrained - Transfer` where the surrogate model is used to produce a trigger using only a realistic set of features and values, and `Constrained - Black-box`, where a completely model-agnostic method is used to compute the SHAP values for the victim model. We argue that this set of scenarios, summarized in Table 1, provide a broad overview of the full attack space.

Existing Threat Models. Most backdoor attack literature adopt the BadNets model threat [26], which defined: (i) an "Outsourced Training Attack", where the adversary has full control over the training procedure, and the end user is only allowed to check the training using a held-out validation dataset; and (ii) a "Transfer Learning Attack", in which the user downloads a pre-trained model and fine-tunes it. We argue that, in the context we are examining, this threat model is difficult to apply directly. Security companies are generally extremely risk-averse, and prefer to either perform the training in-house, or, at most, outsource the hardware while maintaining full control over the software stack used during training. This protects to a large degree against "Outsourced Training Attacks", and renders "Transfer Learning Attacks" almost impossible. Similarly, we do not believe the threat model from [34], where the attacker partially retrains the model, applies in this scenario.

[1] A multi-class setting, where the attacker disguises a malware binary of one family as another family could be constructed, but such a scenario is of lesser practical impact, and, therefore, we will not consider it here.

3.2 Goals and Challenges

To summarize, our design goals are the following: 1. Inject a small number of watermarked poisoning points in the training set with the goal of changing the classification of backdoored malware binaries during testing; 2. Create *clean-label* attacks, in which the watermark is applied to benign executable files; 3. Design model-agnostic attacks that are applicable to different ML models, including ensembles and neural networks; 4. Design attacks feasible to mount in practice by modifying existing Windows PE files. 5. Consider defensive mitigations against these backdoor attacks.

Designing attacks with these properties raises a number of technical challenges. First, selecting the exact features and values for the watermark creation becomes difficult given the goal of maintaining attack stealthiness. The model-agnostic requirement excludes methods that are specific to certain ML algorithms, such as using feature importance computed by tree ensemble models. Practical feasibility is difficult to achieve for attacks modifying PE files, while preserving their functionality. Lastly, the watermark should be applicable to both goodware and malware files without requiring the attacker to re-develop all the programs from scratch.

Figure 2: Force plot showing SHAPs for a benign sample.

4 Backdoor Attacks on Malware Classifiers

In a backdoor poisoning attack, the adversary leverages control over a (subset of) the features to induce misclassifications due to the presence of watermarked values in those feature dimensions. Intuitively, the attack creates an area of density within the feature subspace containing the watermark and the classifier adjusts its decision boundary to accommodate that density of watermarked samples. The watermarked points fight against the influence of surrounding non-watermarked points, as well as the feature dimensions that the attacker does not control, in adjusting the decision boundary. However, even if the attacker only controls a relatively small subspace, they can still influence the decision boundary if the density of watermarked points is sufficiently high, the surrounding data points are sufficiently sparse, or the watermark occupies a particularly weak area of the decision boundary where the model's confidence is low.

The attacker adjusts the density of attack points through the number of watermarked data points they inject, and the area of the decision boundary they manipulate through careful selection of the watermarked feature dimensions and their values. Therefore, there are two natural strategies for developing successful backdoor watermarks: (1) search for areas of weak confidence near the decision boundary, where the watermark can overwhelm existing weak evidence; or (2) subvert areas that are already heavily oriented toward goodware so that the density of the watermarked subspace overwhelms the signal from other nearby samples.

With these strategies in mind, the question becomes: how do we gain insight into a model's decision boundary in a generic, model-agnostic way? We argue that model explanation techniques, like SHapley Additive exPlanations (SHAP), are a natural way to understand the orientation of the decision boundary relative to a given sample. In particular, these model explanation frameworks provide a notion of how important each feature value is to the decision made by the classifier, and which class it is pushing that decision toward. To accomplish this task, the explanation frameworks train a surrogate linear model of the form:

$$g(x) = \phi_0 + \sum_{j=1}^{M} \phi_j x_j \tag{1}$$

based on the input feature vectors and output predictions of the model, and then use the coefficients of that model to approximate the importance and 'directionality' of the feature. Here, x is the sample, x_j is the j^{th} feature for sample x, and ϕ_j is the contribution of feature x_j to the model's decision. The SHAP framework distinguishes itself by enforcing theoretical guarantees on the calculation of the feature contributions based on the notion of game-theoretic Shapley values.

While a full discussion of the SHAP framework is beyond the scope of this paper, suffice it to say that in our task positive SHAP values indicate features that are pushing the model toward a decision of malware, while negative SHAP values indicate features pushing the model toward a goodware decision. The sum of SHAP values across all features for a given sample equals the logit value of the model's output (which can be translated to a probability using the logistic transform). Figure 2 shows a force plot of the SHAP values 'pushing' against each other to arrive at an output score of -0.15 ($\sim$46%), with *major_linker_version* contributing significantly to the classification as goodware, while *num_read_and_execute_sections* contributes significantly toward classification as malware.

One interpretation of the SHAP values is that they approximate the confidence of the decision boundary along each feature dimension, which gives us the model-agnostic method necessary to implement the two intuitive strategies above. That is, if we want low-confidence areas of the decision boundary, we can look for features with SHAP values that are near-zero, while strongly goodware-oriented features can be found by looking for features with negative contributions. Summing the values for each sample along the feature column will then give us an average of the overall orientation for that feature within the dataset.

In the remainder of this section, we will show how to use SHAP values to develop model-agnostic attack strategies that line up with the intuition discussed above. To do so, we first define some initial building blocks, which we will use to select the appropriate feature dimensions and values within those dimensions. Then, we consider algorithms that leverage those building blocks to place backdoors in sparse, low-confidence areas along the decision boundary where we can strongly manipulate the decision boundary, or to subvert existing goodware-oriented areas that allow our attacks to blend in with the background distribution of data.

4.1 Building Blocks

The attacker requires two building blocks to implement a watermark: feature selectors and value selectors. Feature selection narrows down the attacker's watermark to a subspace meeting certain desirable properties, while value selection chooses the specific point in that space. Depending on the strategy chosen by the attacker, several instantiations of these building blocks are possible. Here, we will outline the SHAP-based feature and value selection method we used in our attacks, however other instantiations (perhaps to support alternative attack strategies) may also be possible.

Feature Selection. The key principal for all backdoor poisoning attack strategies is to choose features with a high degree of leverage over the model's decisions. One concept

that naturally captures this notion is feature importance. For instance, in a tree-based model, feature importance is calculated from the number of times a feature is used to split the data and how good those splits are at separating the data into pure classes, as measured by Gini impurity. Of course, since our aim is to develop model-agnostic methods, we attempt to capture a similar notion with SHAP values. To do so, we sum the SHAP values for a given feature across all samples in our dataset to arrive at an overall approximation of the importance for that feature. Since SHAP values encode both directionality (i.e., class preference) and magnitude (i.e., importance), we can use these values in two unique ways.

`LargeSHAP`: By summing the individual SHAP values, we combine the individual class alignments of the values for each sample to arrive at the average class alignment for that feature. Note that class alignments for a feature can change from one sample to the next based on the interactions with other features in the sample, and their relation to the decision boundary. Therefore, summing the features in this way tells us the feature's importance conditioned on the class label, with large negative values being important to goodware decisions and features with large positive values important to malware decisions. Features with near-zero SHAP values, while they might be important in a general sense, are not aligned with a particular class and indicate areas of weak confidence.

`LargeAbsSHAP`: An alternative approach is to ignore the directionality by taking the absolute value of the SHAP values before summing them. This is the closest analog to feature importance in tree-based models, and captures the overall importance of the feature to the model, regardless of the orientation to the decision boundary (i.e., which class is chosen).

Value Selection. Once we have identified the feature subspace to embed the watermark in, the next step is to choose the values that make up the watermark. However, due to the strong semantic restrictions of the binaries, we cannot simply choose any arbitrary value for our watermarks. Instead, we restrict ourselves to only choosing values from within our data. Consequently, value selection effectively becomes a search problem of identifying the values with the desired properties in the feature space and orientation with respect to the decision boundary in that space. According to the intuitive attack strategies described above, we want to select these values based on a notion of their density in the subspace – either selecting points in sparse, weak-confidence areas for high leverage over the decision boundary or points in dense areas to blend in with surrounding background data. We propose three selectors that span this range from sparse to dense areas of the selected subspace.

`MinPopulation`: To select values from sparse regions of the subspace, we can simply look for those values that occur with the least frequency in our dataset. The `MinPopulation` selector ensures both that the value is valid with respect to the semantics of the binary and that, by definition, there are only one or a small number of background data points in the chosen region, which provides strong leverage over the decision boundary.

`CountSHAP`: On the opposite side of the spectrum, we seek to choose values that have a high density of goodware-aligned data points, which allows our watermark to blend in with the background goodware data. Intuitively, we want to choose values that occur often in the data (i.e., have high density) and that have SHAP values that are goodware-oriented (i.e., large negative values). We combine these two components in the following formula:

$$\underset{v}{\operatorname{argmin}}\ \alpha\left(\frac{1}{c_v}\right)+\beta\Big(\sum_{x_v\in X} S_{x_v}\Big) \tag{2}$$

where α, β are parameters that can be used to control the influence of each component of the scoring metric, c_v is the frequency of value v across the feature composing the trigger, and $\sum_{x_v\in X} S_{x_v}$ sums the SHAP values assigned to each component of the data vectors in the training set X, having the value x_v. In our experiments, we found that setting $\alpha=\beta=1.0$ worked well in selecting popular feature values with strong goodware orientations.

`CountAbsSHAP`: One challenge with the `CountSHAP` approach is that while the watermark might blend in well with surrounding goodware, it will have to fight against the natural background data for control over the decision boundary. The overall leverage of the watermark may be quite low based on the number of feature dimensions under the attacker's control, which motivates an approach that bridges the gap between the `MinPopulation` and `CountSHAP` methods. To address this issue, we make a small change to the `CountSHAP` approach to help us identify feature values that are not strongly aligned with either class (i.e., it has low confidence in determining class). As with the `LargeAbsSHAP` feature selector, we an accomplish this by simply summing the absolute value of the SHAP values, and looking for values whose sum is closest to zero

$$\underset{v}{\operatorname{argmin}}\ \alpha\left(\frac{1}{c_v}\right)+\beta\Big(\sum_{x_v\in X} |S_{x_v}|\Big) \tag{3}$$

4.2 Attack Strategies

With the feature selection and value selection building blocks in hand, we now propose two algorithms for combining them to realize the intuitive attack strategies above.

Independent Selection. Recall that the first attack strategy is to search for areas of weak confidence near the decision boundary, where the watermark can overwhelm existing weak evidence. The best way of achieving this objective across multiple feature dimensions is through `Independent` selection of the watermark, thereby allowing the adversary to maximize the effect of the attack campaign by decoupling the two selection phases and individually picking the best combinations. Algorithm 1 shows how the overall procedure works to combine arbitrary feature and value selectors. For our purposes, the best approach using our building blocks is to select the most important features using `LargeAbsSHAP`

Algorithm 1: Independent selection.

Data: N = trigger size;
X = Training data matrix;
S = Matrix of SHAP values computed on training data;
Result: w = mapping of features to values.

```
begin
    w ⟵ map();
    feats ⟵ X.features;
    // Get set of features to attack
    F ⟵ FeatureSelector(S, feats, N);
    // Get list of values to assign
    V ⟵ ValueSelector(S, X, F);
    w[f] = v;
end
```

and then select values using either `MinPopulation` or `CountAbsSHAP`. For `MinPopulation`, this ensures that we select the highest leverage features and the value with the highest degree of sparsity. Meanwhile, for the `CountAbsSHAP` approach, we try to balance blending the attack in with popular values that have weak confidence in the original data. While we find that this attack strongly affects the decision boundary, it is also relatively easy to mitigate against because of how unique the watermarked data points are, as we will show in Section 7.

Greedy Combined Selection. The second attack strategy seeks to subvert areas that are already heavily oriented toward goodware, and consequently increase the overall stealthiness of the attack. Here, we must be careful to select coherent watermark values that represent the dependencies across the features observed in the data. We achieve this by developing a greedy algorithm for conditionally selecting new feature dimensions and values such that the new values are consistent with existing goodware-oriented points across all watermarked feature dimensions, as shown in Algorithm 2. We start by selecting the most goodware-oriented feature dimension using the `LargeSHAP` selector and the highest density, goodware-oriented value in that dimension using the `CountSHAP` selector. Next, we remove all data points that do not have the selected watermark value and repeat the procedure with the subset of data conditioned on the current watermark. This procedure ensures that we have at least one background data point that exactly matches our watermark. In practice, we have found that this `Combined` process results in hundreds or thousands of background points with watermark sizes of up to 32 features. By comparison, the `Independent` algorithm quickly separates the watermark from all existing background points after just three or four feature dimensions.

Algorithm 2: Greedy combined selection.

Data: N = trigger size;
X = Training data matrix;
S = Matrix of SHAP values computed on training data;
Result: w = mapping of features to values.

```
begin
    w ⟵ map();
    selectedFeats ⟵ ∅;
    S_local ⟵ S;
    feats ⟵ X.features;
    X_local ⟵ X;
    while len(selectedFeats) < N do
        feats = feats \ selectedFeats;
        // Pick most benign oriented (negative) feature
        f ⟵ LargeSHAP(S_local, feats, 1, goodware);
        // Pick most benign oriented (negative) value of f
        v ⟵ CountSHAP(S_local, X_local, f, goodware);
        selectedFeats.append(f);
        w[f] = v;
        // Remove vectors without selected (f, v) tuples
        mask ⟵ X_local[:, f] == v;
        X_local = X_local[mask];
        S_local = S_local[mask];
    end
end
```

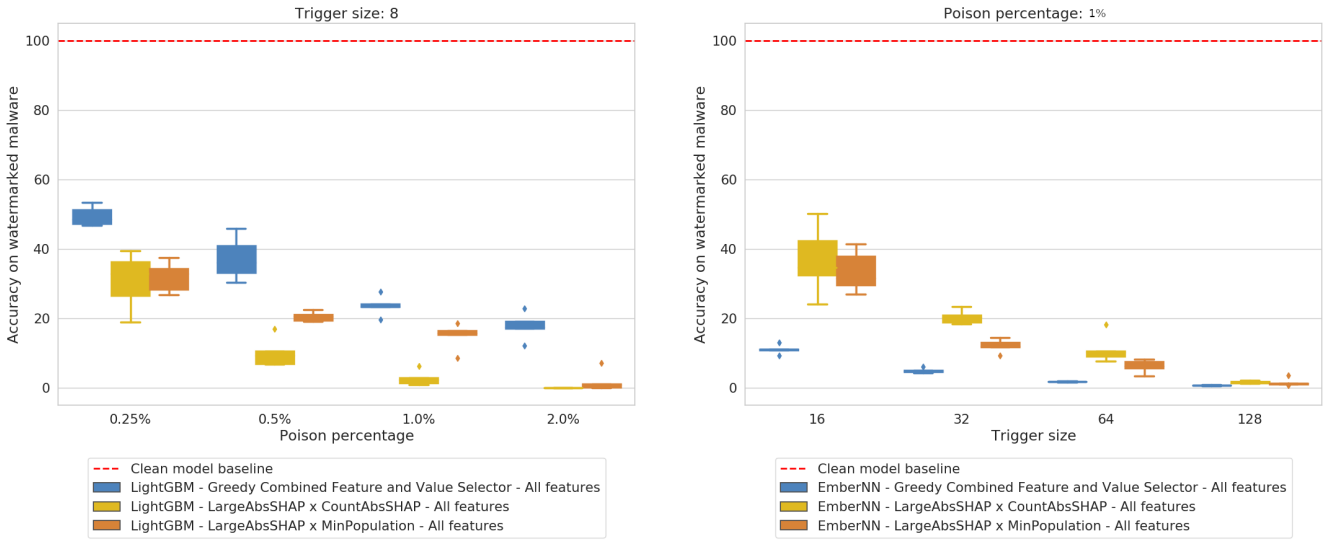
5 Experimental Attack Evaluation

EMBER is a representative public dataset of malware and goodware samples used for malware classification, released together with a LightGBM gradient boosting model, that achieves good binary classification performance.[2] The EMBER dataset consists of the 2351-dimensional feature vectors extracted from 1.1 million Portable Executable (PE) files for the Microsoft Windows operating system, using LIEF [2]. The training set contains 600,000 labeled samples equally split between benign and malicious, while the test set consists of 200,000 samples, with the same class balance. All the binaries categorized as malicious were reported as such by at least 40 anti-virus engines on VirusTotal [7].

Following Anderson et al. [10], we use suggested default parameters for training LightGBM (100 trees and 31 leaves per tree). In addition to LightGBM, we also considered state-of-the-art neural networks for the task of malware classification. Given the feature-based nature of our classification task, we experimented with different architectures of feed-forward neural networks. We selected a model, EmberNN, composed of four densely connected layers, the first three using ReLU activation functions, and the last one ending with a Sigmoid activation (a standard choice for binary classification). The first three dense layers are interleaved by Batch Normalization layers and a 50% Dropout rate is applied for regularization during training to avoid model overfitting to training data. Performance metrics for both clean models (before the attacks are performed) on the EMBER dataset are provided in Table 2. The two models are comparable, with EmberNN performing slightly better than the publicly released LightGBM model.

According to the threat models outlined in Section 3, we are interested in a series of metrics. First, we observe the following

[2] In this work we use EMBER 1.0



(a) LightGBM target

(b) EmberNN target

Figure 3: Accuracy of the backdoor model over backdoored malicious samples (B-ABM) for `Unrestricted` attacker. Lower B-ABM is the result of stronger attacks. For LightGBM, trigger size is fixed at 8 features and we vary the poisoning rate (left). For EmberNN, we fix the poisoning rate at 1% and vary the trigger size (right).

Model	F1 Score	FP rate	FN rate
LightGBM	0.9861	0.0112	0.0167
EmberNN	0.9911	0.0067	0.0111

Table 2: Performance metrics for the clean models.

indicators for the backdoored model trained on poisoned data:

B-ABM: Accuracy of the backdoored model on watermarked malware samples. This measures the percentage of times a backdoored model is effectively tricked into misclassifying a previously correctly recognized malicious binary as goodware. Therefore, the primary goal of the attacker is to reduce this value.

B-GEN: Accuracy of the backdoored model on the clean test set. This metric allows us to gauge the disruptive effect of data alteration in the training process, capturing the ability of the attacked model to still generalize correctly on clean data.

B-FP: False positives (FP) of the backdoored model on watermarked samples. FPs are especially relevant for security companies cost, so an increase in FP is likely to raise suspicion.

We are also interested in understanding the effects that poisoned training data might cause when the samples are analyzed with a clean model. A security company might validate the incoming data stream by passing new samples through an existing ML model before using them for re-training.

C-ABG: Difference between the accuracy of the clean model over the clean and backdoored benign samples. The adversary's goal is to minimize the absolute C-ABG to evade detection.

5.1 Attack Performance

We start by analyzing the `Unrestricted` attack impact on the LightGBM model by varying the number of features in the watermark, the poison size, and the attack strategies. We then perform an evaluation of the attack on EmberNN.

Targeting LightGBM. To gauge the performance of the methods we discussed above, we ran the two `Independent` attacks and the `Combined` strategy on the LightGBM model trained on the EMBER dataset using the LightGBM TreeSHAP explainer. We took care to only inject the trigger at testing time in malicious samples correctly recognized by the clean model. Plotting attack success rates for an 8-feature trigger, shown in Figure 3a, clearly highlights the correlation between increasing poison pool sizes and lower B-ABM. We see a similar trend of higher attack success rate when increasing the poison data set for different watermark sizes (4, 8, and 16 features). Detailed results for all three strategies, varying poisoning and trigger sizes are included in Appendix A.

Interestingly, the SHAP feature selection allows the adversary to use a relatively small trigger, 8 features out of 2,351 in Figure 3a, and still obtain powerful attacks. For 6,000 poisoned points, representing 1% of the entire training set, the most effective strategy, `LargeAbsSHAP` x `CountAbsSHAP`, lowers B-ABM on average to less than 3%. Even at much lower poisoning rates (0.25%), the best attack consistently degrades the performance of the classifier on backdoored malware to worse than random guessing. Surprisingly, with an extremely small watermark of 4 features, and 1% poison pool, the attack still induces an average accuracy loss of $\approx 59.6\%$. All the strategies induce small overall changes in the B-FP under 0.001, with marginally larger increases correlated to

larger poison sizes. We also observe that the attack leads to minimal changes in B-GEN, on average below 0.1%.

Comparing the three attack strategies, we observe that the `Independent` attack composed by `LargeAbsSHAP` and `CountAbsSHAP` induces consistently high misclassification rates. It is also important to mention here that the `Combined` strategy is, as expected, remarkably stealthier, with generally small degradation in C-ABG. In conclusion, we observe that the attack is extremely successful at inducing targeted mis-classification in the LightGBM model, while maintaining good generalization on clean data, low false positive rates on both clean and backdoored models.

Targeting EmberNN. Running the same series of attacks against EmberNN using the GradientSHAP explainer, we immediately notice that the Neural Network is more resilient to poisoning attacks. Moreover, here the effect of trigger size is critical. Figure 3b shows the progression of accuracy loss over the watermarked malicious samples with the increase in trigger size, at a fixed 1% poisoning rate. For example, under the most effective strategy, with a trigger size of 128 features, B-ABM becomes on average 0.75%, while B-ABM averages 5.05% at 32 features.

A critical element that distinguishes the three strategies on EmberNN, is the C-ABG. While, in fact, the other tracked metrics show a behavior similar to the case of LightGBM, a clean EmberNN model often fails almost completely in recognizing backdoored benign points as goodware. Here, the `Combined` strategy emerges as a clear "winner," being both very effective in inducing misclassification, and, simultaneously, minimizing the C-ABG delta, with an average absolute value of $\approx 0.3\%$. We also measure the other attack metrics, and observe good generality on clean data, with B-GEN close to the original 99.11% in most cases, low false positives increases for the backdoored model ($\approx 0.1-0.2\%$ average increase in B-FP). Interestingly, we observed that the attack performance on the NN model is more strongly correlated with the size of the backdoor trigger than with the poison pool size, resulting in small (0.5%) injection volumes inducing appreciable misclassification rates. We note that this observation confirms previous works showing effective poisoning attacks on neural networks with extremely small adversarial poison sizes.

5.2 Limiting the Attacker

We consider here the `Transfer` attacker without access to the model. This threat model prevents the attacker from being able to compute the SHAP values for the victim model, therefore, the backdoor has to be computed using a surrogate (or proxy) model sharing the same feature space. We simulate this scenario by attempting a backdoor transferability experiment between our target models. We use each model as the surrogate for the other, and compute a backdoor for each proxy model using the most effective strategy against that model. We then use that watermark to poison the training set of the other model.

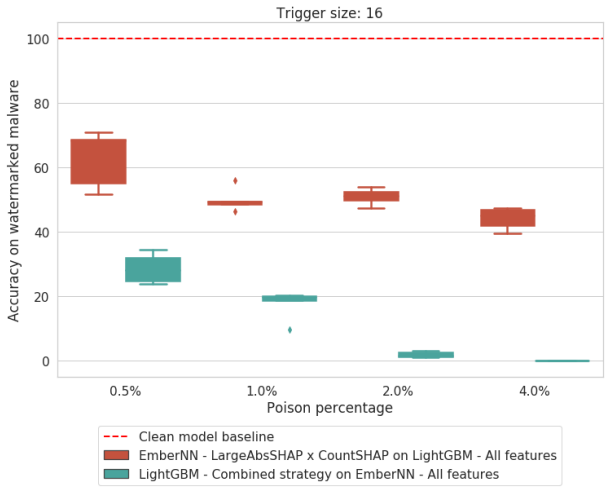


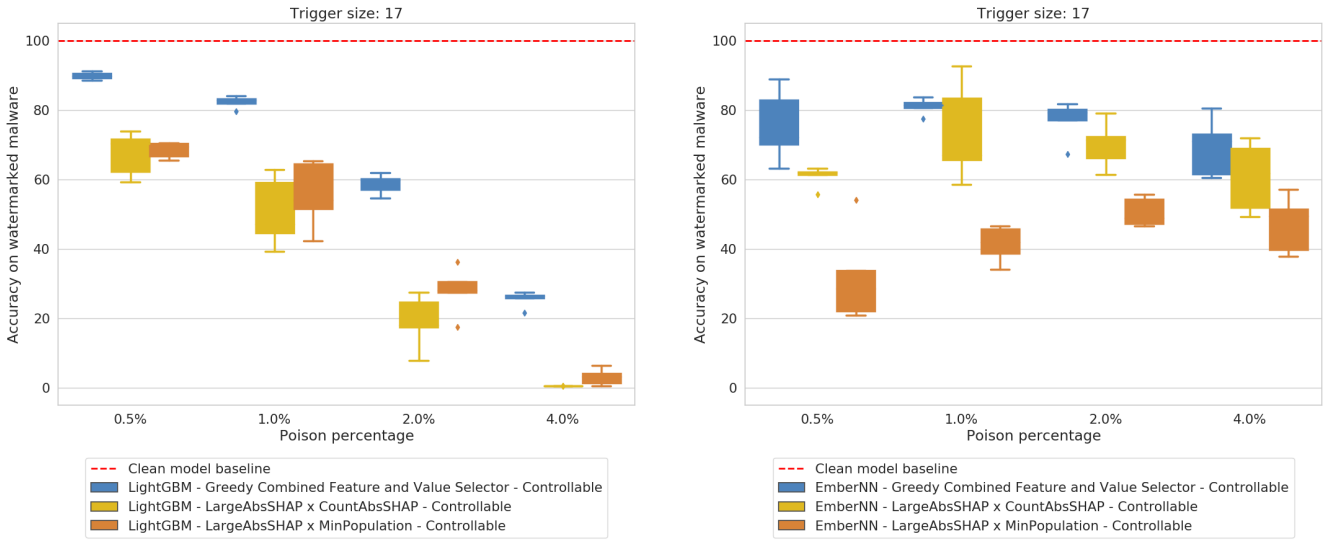
Figure 4: `Transfer` B-ABM for both models (other model used as surrogate), as function of poisoned data percentage.

In our experiments, we fix the watermark size to 16 features and we attack the LightGBM model with a watermark generated by the `Combined` strategy using the SHAP values extracted from the surrogate model, EmberNN. Then we repeat a similar procedure by creating a backdoor using the `Independent` strategy, with the combination of `LargeAbsSHAP` and `CountAbsSHAP` for feature and value selection respectively, computed on the LightGBM proxy, and used it to poison EmberNN's training set. The accuracy loss on the watermarked malware for both attacks is shown in Figure 4. The empirical evidence observed supports the conclusion that our attacks are transferable both ways. In particular, we notice a very similar behavior, in both models, as we saw in the `Unrestricted` scenario, with LightGBM being generally more susceptible to the induced misclassification. Here, in fact, the trigger generated using the surrogate model produces an $\approx 82.3\%$ drop in accuracy on the backdoored malware set, for a poison size of 1% of the training set.

Lastly, we evaluate the scenario in which the attacker has access to only a small subset of clean training data and uses the same model architecture as the victim's (`Data-Limited`). We perform this experiment by training a LightGBM model with 20% of the training data and using it to generate the trigger, which we then used to attack the LightGBM model trained over the entire dataset. Using the `Independent` strategy with `LargeAbsSHAP` and `CountAbsSHAP` over 16 features and a 1% poison set size, we noticed essentially no difference compared to the same attack where the SHAP values are computed over the entire training set (≈ 0.04 B-ABM).

6 Real-World Considerations

We demonstrated the success of our model-agnostic attack strategies when the attacker has full control of the features and



(a) LightGBM target

(b) EmberNN target

Figure 5: Accuracy of the backdoor model over watermarked malicious samples (B-ABM). Lower B-ABM is the result of stronger attacks. The watermark uses the subset of 17 features of EMBER, modifiable by the `Constrained` adversary.

can change their values at will. Realistically, the attacker faces additional challenges, such as ensuring that watermarked goodware maintains its original benign label and watermarked malware retains its malicious functionality. This means that an attacker must expend non-trivial effort to ensure that changes made to the binaries by the watermark do not break the semantics or otherwise compromise the functionality. Here, we explore the impact of these real-world constraints faced by the `Constrained` adversary. We implement a watermarking utility and apply the selected watermarks to real goodware and malware samples from the EMBER dataset.

Specifically, we use the *pefile* [18] library to create a generic utility that attempts to apply a given watermark to arbitrary Windows binaries. Creating this utility in a sufficiently general way required specialized knowledge of the file structure for Windows Portable Executable (PE) files, and was particularly difficult when implementing watermarks that add sections to the binaries. Doing so required extending the section table with the appropriate sections, names, and characteristics, which in turn meant relocating structures that follow the section table, such as data directories and the sections themselves, to allow for arbitrary increases in the number of sections added. While developing the watermarking utility was challenging, we believe it is well within the capabilities of a determined attacker, and can subsequently be reused for a variety of watermarks produced by the strategies described earlier.

To determine the watermark effects on the binaries' functionality, we run each sample in a dynamic analysis sandbox, which uses a variety of static, dynamic, and behavioral analysis methods to determine whether the binary is malicious or not. This experiment helps evaluate three important aspects of our attack when applied in the real world: (1) the ability to keep the original labels on watermarked goodware, (2) the ability to maintain the original malicious functionality of the watermarked malware, and (3) the impact of semantic restrictions on the features the attacker can use to carry out the backdoor poisoning attack.

Challenges. When creating our watermarking utility, we encountered several challenges that required us to drop certain features and consider dependencies among features that restrict the values they can take on. First, we noted that the vast majority of the features in EMBER (2,316 of 2,351) are based on feature hashing, which is often used to vectorize arbitrarily large spaces into a fixed-length vector. For example, strings uncovered in the binary may be hashed into a small number of buckets to create a fixed-number of counts. Given the preimage resistance of the hash function, directly manipulating these features would be extremely difficult, and consequently we discard all hash-based features, leaving us with just 35 directly-manipulatable, non-hashed features (listed in Table 6, Appendix A.1).

Next, we consider dependencies among the non-hashed features. As it turns out, many of the features are derived from the same underlying structures and properties of the binary, and may result in conflicting watermarks that cannot be simultaneously realized. For example, the *num_sections* and *num_write_sections* features are related because each time we add a writeable section, we necessarily increase the total number of sections. To handle these dependencies, we remove any features whose value is impacted by more than one other feature (e.g., *num_sections*). This allows us to keep the maximal number of features without solving complex constraint optimization problems.

The last challenge arose from watermark values that are physically impossible to set. This challenge manifested itself

in two distinct ways. In the first, a watermark may require us to remove data from the binary to achieve the requested value (e.g., removing URLs or reducing the file size). Since we cannot be sure that these changes will not destroy the functionality of the program, we must consider the binary to be unwatermarkable (i.e., watermarking failed). In the second case, certain watermark values derived from the dataset were found to cause the binaries to stop functioning in modern operating systems and we removed them from consideration. [3]

After reducing our set of features based on the above criteria, we are left with 17 features that our generic watermarking utility can successfully manipulate on arbitrary Windows binaries. Examples of the features and their watermark values can be found in Table 4, Appendix A.3. As we will see, despite the significant reduction in the space of available features, our proposed attack strategies still show significant effectiveness.

6.1 Functional Evaluation

Using the *pefile*-based watermarking utility, we randomly selected the 100 goodware and 100 malware binaries from our dataset and manipulated each of them with the watermarks for the LightGBM and EmberNN models, resulting in a total of 200 watermarked binaries for each model. The original and watermarked binaries were submitted to a dynamic analysis environment with an execution timeout of 120 seconds. Table 5, in Appendix A.3, shows the results of our experiments. In the case of the LightGBM and EmberNN watermarks, both goodware and malware have similar numbers of failed watermarking attempts due to the physical constraints on the binaries, with the most prevalent reason (>90%) being binaries that were too large for the selected *size* watermark. For those files that were successfully watermarked, we observed that goodware always maintained its original benign label, while malware retained its malicious functionality in 61-66% of the cases. We also scanned our watermarked binaries with ESET and Norton AntiVirus signature-based antivirus engines, similar to those used by crowdsourced threat intelligence feeds, and found that none of the goodware changed labels due to the presence of our watermarks. Overall, this indicates that an attacker could use up to 75% of the observed goodware and 47% of the observed malware in these threat intelligence feeds to launch their backdoor poisoning attack. This is sufficient in real-world attacks as the adversary needs a small percentage of watermark binaries for poisoning, and a small number of watermark malware to execute the attack. Finally, it is important to point out that our evaluation here focused on an attacker using commodity goodware and malware. However, an advanced attacker may produce their own software to better align with the chosen watermark values and maximize the attack impact.

Realistic Attack Efficacy. As shown in Figure 5, the effectiveness of the attack is slightly decreased when the backdoor trigger is constructed using only the 17 manipulatable features supported by our watermarking utility. Such a `Constrained` adversary, is, as expected, strictly less powerful than the `Unrestricted` attacker we explored in Section 5. On the other hand, despite the strong limitations introduced to ease practical implementation, we argue that the average accuracy loss is still extremely relevant given the security critical application. Moreover, if we allow the poison size to grow to 2% of the overall training set, we obtain B-ABM levels comparable with the `Unrestricted` at 1% poison size on LightGBM. Next, we constrain the attacker further by combining the limitation over features control with lack of access to the original model, `Constrained-Transfer`. As in Section 5.2, we compute the watermark using a surrogate model, with the most effective `Transfer` strategy we identified before, but this time restricted to the controllable features. We observe still significant accuracy decrease with an average B-ABM of 54.53% and 56.76% for LightGBM and EmberNN respectively. Lastly, we look at the `Constrained-Black-box` scenario, where we produce the SHAP values for only the manipulatable features using the SHAP KernelExplainer, which operates purely by querying the model as a black-box. We target LightGBM, with the `LargeAbsSHAP` x `CountAbsSHAP` strategy poisoning 1% of the training set. The resulting model exhibits an average B-ABM of 44.62%, which makes this attacker comparable to one having access to model-specific SHAP explainers.

7 Mitigation

Recently, researchers started tackling the problem of defending against backdoor attacks [19, 33, 50, 52]. Nearly all existing defensive approaches, however, are specifically targeted at computer vision Deep Neural Networks, and assume adversaries that actively tamper with the training labels. These limitations make them hard to adapt to the class of model-agnostic, clean-label attacks we are interested in. We discuss here representative related work.

Tran et al. [50] propose a defensive method based on *spectral signatures*. This technique, relies on detecting two ε-spectrally separable subpopulations based on SVD decomposition, and is successful only when the signatures are computed in the latent representation space learned by the convolutional neural network. Chen et al. [19] similarly rely on the representation learned by the CNN and perform k-means clustering on the activations of the last convolutional layer. The defense of Liu et al. [33] is based on combining network fine tuning and neuron pruning, thus making it not applicable here. Finally, NeuralCleanse [52] is based on the intuition that in a backdoored model, the perturbation necessary to induce a misclassification towards the targeted class should be smaller than that required to obtain different labels. This approach was designed considering multi-class classification problem, as encountered in image recognition, moreover the suggested filtering and pruning mitigation are neural-network specific.

[3] Amazingly, one example set watermark value for *major_subsystem_version* to 2, which reflects a binary compiled for Windows 2.0!

Target	Strategy	B-ABM (after attack)	Mitigation	New B-ABM (after defense)	Poisons Removed	Goodware Removed
LightGBM	LargeAbsSHAP x MinPopulation	0.5935	HDBSCAN	0.7422	3825	102251
			Spectral Signature	0.7119	962	45000
			Isolation Forest	0.9917	6000	11184
	LargeAbsSHAP x CountAbsSHAP	0.5580	HDBSCAN	0.7055	3372	93430
			Spectral Signature	0.6677	961	44999
			Isolation Forest	0.9921	6000	11480
	Combined Feature Value Selector	0.8320	HDBSCAN	0.8427	1607	115282
			Spectral Signature	0.7931	328	45000
			Isolation Forest	0.8368	204	8927
EmberNN	LargeAbsSHAP x MinPopulation	0.4099	HDBSCAN	0.3508	3075	137597
			Spectral Signature	0.6408	906	45000
			Isolation Forest	0.9999	6000	14512
	LargeAbsSHAP x CountAbsSHAP	0.8340	HDBSCAN	0.5854	2499	125460
			Spectral Signature	0.8631	906	45000
			Isolation Forest	0.9999	6000	15362
	Combined Feature Value Selector	0.8457	HDBSCAN	0.8950	1610	120401
			Spectral Signature	0.9689	904	45000
			Isolation Forest	0.8030	175	13289

Table 3: Mitigation results for both LightGBM and EmberNN. All attacks were targeted towards the 17 controllable features (see Section 6), with a 1% poison set size, 6000 backdoored benign samples. We show B-ABM for the backdoored model, and after the defense is applied. We also include number of poisoned and goodware points filtered out by the defensive approaches.

Protecting ML systems from adversarial attacks is an intrinsically hard problem [17], and we argue that both the clean-label nature of our attack, and its targeting of a highly variable data population (benign binaries), only increase the difficulty of the task. In the following, we empirically show that currently available mitigations are often insufficient, and we leave a deeper and more detailed defensive analysis for future work.

Defensive Approaches. According to our threat model, the defender (the party who is training the malware classification model) is assumed to: (i) Have access to the (poisoned) training data; (ii) Have access to a small set of clean labeled data. This common assumption in adversarial ML fits nicely with the context since security companies often have access to internal, trusted, data sources; (iii) Know that the adversary will target the most relevant features for the model.

First, we evaluated if a drop in C-ABG might be used as a signal of a poisoning campaign, but we found that this metric cannot distinguish between adversarially crafted poison samples and legitimate samples, particularly when stealthier strategies, such as `Combined` are used. Successively, we tackle the main obstacle making common defensive methodologies inapplicable: the lack of a learned representation space capable of accentuating the differences between the training data subpopulations. To address this issue and obtain a proxy for the latent representation space, we used the held-out, safe, data (20% of the training set, or 120,000 samples) to train a clean model and extract the SHAP values for the safe training set. We then fixed a number of most important features (32) and reduced the dimensionality of the training set to that subspace, after standardizing all values to $[-1,1]$.

We then evaluate three mitigation strategies over the reduced feature space. (i) A state-of-the-art defensive strategy, spectral signatures [50], which we adapt by computing the singular value decomposition of the benign samples over the new feature space. Then, as in the original paper, we compute the *outlier score* by multiplying the top right singular vector and we filter out the samples with the highest 15% scores. (ii) Hierarchical density-based clustering, HDBSCAN [15]. Chen et al. [19] use k-means for defensive clustering over neuron activations. We mutate the idea, using HDBSCAN instead, with the intuition that watermarked samples form a subspace of high density in the reduced feature space, and generate a tight cluster. Additionally, HDBSCAN does not require a fixed number of clusters, but has two other parameters that control the cluster density (minimum size of a cluster, set at 1% of the training benign data, 3000 points, and minimum number of samples to form a dense region, set at 0.5%, 600 points). As in [19], we compute Silhouette scores on the resulting clusters, to obtain an estimate of the intra-cluster similarity of a sample compared to points from its nearest neighboring cluster, and filter out samples from each cluster with a probability related to the cluster silhouette score. (iii) Isolation Forest [32], an algorithm for unsupervised anomaly detection, based on identifying rare and different points, instead of building a model of a normal sample. The intuition here is that such an anomaly detection approach might identify the watermarked samples as outliers due to their similarity compared to the very diverse background points. We experiment with default parameters of Isolation Forest.

Results of Mitigation Strategies. Table 3 shows the effect of these three mitigation strategies over the different models and attack strategies. Two main takeaways emerge from these empirical results. First, the Isolation Forest, trained on the reduced

feature space, is often capable of correctly isolating all the backdoored points with relatively low false positives. Note that this happens exclusively when an Isolation Forest is trained on the transformed dataset (reduced to most important features). The same algorithm applied in the original feature space detects only a tiny fraction of the backdoored points ($\approx 60-70$), thus reinforcing the observation in [50] that the subpopulations are not sufficiently separable in the original feature space. Second, none of the mitigation approaches was able to isolate the points attacked with watermarks produced with the `Combined` strategy. This confirms that the `Combined` attack strategy is much more stealthy compared to both `Independent` strategies.

We note that the proposed mitigations are only a first practical step in defending against clean-label backdoor attacks in a model-agnostic setting. We leave a deeper investigations of more general defensive methods, as a topic of future work. The task is extremely challenging due to the combined effect of the small subpopulation separability induced by clean-label attacks, and the difficulty of distinguishing dense regions generated by the attack from other dense regions naturally occurring in diverse sets of benign binaries.

8 Related Work

Here, we provide a quick rundown of adversarial machine learning applied to computer security.

Evasion. Biggio et al. [11] proposed a gradient-based attack against SVM for malicious PDF detectors by adding keywords strongly correlated to benign files. Dang et al. [22] introduced EvadeHC, performing a black-box attack by morphing the malicious file using a hill-climbing approach. Xu et al. [54] also used PDF files as target for their study, but proposed a generalizable method to find evasive variants, in a black-box context, using Genetic Programming. Similarly focused on black-box mutation is the work by Anderson et al. [9], which trains Reinforcement Learning agents to generate functionality preserving mutations of a binary. A different line of research is represented by Grosse et al. [25], Kolosnjaji et al. [30], and Suciu et al. [46], who generate adversarial samples aimed at fooling Neural Network classifiers. The first targeted networks trained on binary value features extracted from the files, while the second group focused on CNNs operating directly on the raw executable binary files, like Malconv [39].

Poisoning. Biggio et al. [12] was one of the first to bring the problem of ML poisoning attacks to light. Successive work [14], demonstrated the relevance of poisoning attacks in computer security by attacking the Malheur malware behavioral clustering tool [40]. Later research by Xiao et al. [53] showed that feature selection methods, like LASSO, ridge regression, and elastic net, were susceptible to small poison sizes. Gradient-based poisoning availability attacks have been shown against regression [27] and neural networks [38], and the transferability of these attacks has been demonstrated [23]. Suciu et al. [47] proposed a framework for defining attacker models in the poisoning space, and developed StingRay, a multi-model target poisoning attack methodology.

Backdoor Attacks. Backdoor attacks were introduced by Gu et al. in BadNets [26], identifying a supply chain vulnerability in modern machine learning as-a-service pipelines. Liu et al. [34] explored introducing trojan triggers in image recognition Neural Networks, without requiring access to the original training data, by partially re-training the models. Later works by Turner et al. [51] and Shafahi et al. [44] further improved over the existing attacks by devising clean-label strategies. To the best of our knowledge, no previous work has focused on backdoor attacks aimed specifically at malicious software classifiers. Moreover, we are not aware of any attempt at injecting backdoors in Gradient Boosting models.

9 Discussion and Conclusion

Malware classification is an inherently hard problem that requires large collections of labeled data to be confronted using machine learning based methodologies. With this work we begin shedding light on backdoor attacks, a new threat vector that we believe will only grow in relevance in the coming years. We demonstrated how to conduct backdoor poisoning attacks that are model-agnostic, do not assume control over the labeling process, and can be adapted to very restrictive adversarial models. For instance, an attacker with the sole knowledge of the feature space can mount a realistic attack by injecting a relatively small pool of poisoned samples (1% of training set) and achieve high success rate at misclassifying the backdoored malware samples. We also designed the `Combined` strategy that creates backdoored points in high-density regions of the legitimate samples, making it very difficult to detect with the explored mitigations.

There are some limitations of this work that we would like to expose, and some concluding remarks we would like to make hoping to increase the awareness to this kind of vulnerability. First, the attacks we explored rely on the attacker knowing the feature space used by the victim model. While this assumption is partially justified by the presence of natural features in the structure of executable files, we consider the development of more generic attack methodologies, which do not rely on any knowledge from the adversary's side, as an interesting and challenging future research direction. Second, designing a general mitigation method, particularly against our stealthy `Combined` attack strategy, remains a challenging problem we leave for future work. Lastly, adaptation of these attacks to other malware classification problems that might rely on combining static and dynamic analysis is also a topic of future investigation.

References

[1] AlienVault - Open Threat Exchange. https://otx.alienvault.com/.

[2] LIEF - Library to Instrument Executable Formats. https://lief.quarkslab.com/.

[3] Machine Learning Static Evasion Competition. https://www.elastic.co/blog/machine-learning-static-evasion-competition.

[4] MetaDefender Cloud | Homepage. https://metadefender.opswat.com.

[5] Skylight Cyber | Cylance, I Kill You! https://skylightcyber.com/2019/07/18/cylance-i-kill-you/.

[6] VirSCAN.org - Free Multi-Engine Online Virus Scanner. https://www.virscan.org/.

[7] VirusTotal. http://www.virustotal.com/.

[8] Brandon Amos, Hamilton Turner, and Jules White. Applying machine learning classifiers to dynamic Android malware detection at scale. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1666–1671, July 2013. ISSN: 2376-6506.

[9] Hyrum S Anderson, Anant Kharkar, and Bobby Filar. Evading Machine Learning Malware Detection. In *Black Hat USA*, page 6, 2017.

[10] Hyrum S. Anderson and Phil Roth. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *arXiv:1804.04637 [cs]*, April 2018. arXiv: 1804.04637.

[11] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. *Advanced Information Systems Engineering*, 7908:387–402, 2013.

[12] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1467–1474, Edinburgh, Scotland, June 2012. Omnipress.

[13] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In John Langford and Joelle Pineau, editors, *29th Int'l Conf. on Machine Learning*, pages 1807–1814. Omnipress, 2012.

[14] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. Poisoning behavioral malware clustering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop - AISec '14*, pages 27–36, Scottsdale, Arizona, USA, 2014. ACM Press.

[15] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In *Advances in Knowledge Discovery and Data Mining*, volume 7819, pages 160–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[16] Nicholas Carlini. Adversarial machine learning reading list. Available at https://nicholas.carlini.com/writing/2018/adversarial-machine-learning-reading-list.html, 2020.

[17] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On Evaluating Adversarial Robustness. *arXiv:1902.06705 [cs, stat]*, February 2019. arXiv: 1902.06705.

[18] Ero Carrera. erocarrera/pefile. https://github.com/erocarrera/pefile. original-date: 2015-04-13T13:45:19Z.

[19] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. *arXiv:1811.03728 [cs, stat]*, November 2018. arXiv: 1811.03728.

[20] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. *arXiv:1712.05526 [cs]*, December 2017. arXiv: 1712.05526.

[21] Zheng Leong Chua, Shiqi Shen, Prateek Saxena, and Zhenkai Liang. Neural Nets Can Learn Function Type Signatures From Binaries. In *USENIX Security Symposium*, page 19, 2017.

[22] Hung Dang, Yue Huang, and Ee-Chien Chang. Evading Classifiers by Morphing in the Dark. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, pages 119–133, Dallas, Texas, USA, 2017. ACM Press.

[23] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 321–338, Santa Clara, CA, August 2019. USENIX Association.

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, December 2014. arXiv: 1412.6572.

[25] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial Examples for Malware Detection. In *Computer Security – ESORICS 2017*, volume 10493, pages 62–79. Springer International Publishing, Cham, 2017.

[26] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv:1708.06733 [cs]*, August 2017. arXiv: 1708.06733.

[27] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *IEEE Symposium on Security and Privacy*, SP '18, pages 931–947. IEEE CS, 2018.

[28] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[29] Dhilung Kirat and Giovanni Vigna. MalGene: Automatic Extraction of Malware Analysis Evasion Signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, pages 769–780, Denver, Colorado, USA, 2015. ACM Press.

[30] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537, September 2018. ISSN: 2219-5491.

[31] Marek Krčál, Ondřej Švec, Martin Bálek, and Otakar Jašek. Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only. *ICLR 2018*, February 2018.

[32] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, December 2008. ISSN: 2374-8486.

[33] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. *arXiv:1805.12185 [cs]*, May 2018. arXiv: 1805.12185.

[34] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning Attack on Neural Networks. In *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA, 2018. Internet Society.

[35] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[36] Thomas Mandl, Ulrich Bayer, and Florian Nentwich. ANUBIS ANalyzing Unknown BInarieS The automatic Way. In *Virus bulletin conference*, volume 1, page 02, 2009.

[37] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA, 2017. Internet Society.

[38] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. *arXiv:1708.08689 [cs]*, August 2017. arXiv: 1708.08689.

[39] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. Malware Detection by Eating a Whole EXE. *arXiv:1710.09435 [cs, stat]*, October 2017. arXiv: 1710.09435.

[40] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[41] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, and Pablo G. Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231:64–82, May 2013.

[42] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20, October 2015. ISSN: null.

[43] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. Malrec: Compact Full-Trace Malware Recording for Retrospective Deep Analysis. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 10885, pages 3–23, Cham, 2018. Springer International Publishing.

[44] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Advances in Neural Information Processing Systems*, April 2018.

[45] Andrii Shalaginov, Sergii Banin, Ali Dehghantanha, and Katrin Franke. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. In Ali Dehghantanha, Mauro Conti, and Tooska Dargahi, editors, *Cyber Threat Intelligence*, volume 70, pages 7–45. Springer International Publishing, Cham, 2018.

[46] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring Adversarial Examples in Malware Detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14, San Francisco, CA, USA, May 2019. IEEE.

[47] Octavian Suciu, Radu Ma, Tudor Dumitras, and Hal Daume Iii. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. page 19, 2018.

[48] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199 [cs]*, December 2013. arXiv: 1312.6199.

[49] Kimberly Tam, Salahuddin J. Khan, Aristide Fattori, and Lorenzo Cavallaro. CopperDroid: Automatic Reconstruction of Android Malware Behaviors. Internet Society, 2015.

[50] Brandon Tran, Jerry Li, and Aleksander Mądry. Spectral signatures in backdoor attacks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 8011–8021, Montréal, Canada, December 2018. Curran Associates Inc.

[51] Alexander Turner, Dimitris Tsipras, and Aleksander Mądry. Clean-Label Backdoor Attacks. *Manuscript submitted for publication*, page 21, 2019.

[52] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, San Francisco, CA, USA, May 2019. IEEE.

[53] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is Feature Selection Secure against Training Data Poisoning? In *International Conference on Machine Learning*, page 10, 2015.

[54] Weilin Xu, Yanjun Qi, and David Evans. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In *Proceedings 2016 Network and Distributed System Security Symposium*, San Diego, CA, 2016. Internet Society.

[55] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference on - ACSAC 2017*, pages 288–302, Orlando, FL, USA, 2017. ACM Press.

Feature	LightGBM	EmberNN
major_image_version	1704	14
major_linker_version	15	13
major_operating_system_version	38078	8
minor_image_version	1506	12
minor_linker_version	15	6
minor_operating_system_version	5	4
minor_subsystem_version	5	20
MZ_count	626	384
num_read_and_execute_sections	20	66
num_unnamed_sections	11	6
num_write_sections	41	66
num_zero_size_sections	17	17
paths_count	229	18
registry_count	0	33
size	1202385	817664
timestamp	1315281300	1479206400
urls_count	279	141

Table 4: Watermarks for LightGBM and EmberNN used during feasibility testing.

Dataset	Label	Result	Count
Original	Goodware	Dynamic Benign	100
		Dynamic Malicious	0
	Malware	Dynamic Benign	7
		Dynamic Malicious	93
LightGBM	Goodware	Failed	25
		Dynamic Benign	**75**
		Dynamic Malicious	0
	Malware	Failed	23
		Dynamic Benign	30
		Dynamic Malicious	**47**
EmberNN	Goodware	Failed	33
		Dynamic Benign	**67**
		Dynamic Malicious	0
	Malware	Failed	33
		Dynamic Benign	23
		Dynamic Malicious	**44**

Table 5: Summary of results analyzing a random sample of 100 watermarked goodware and malware samples in the dynamic analysis environment.

A Additional Results

Here the reader will find additional details on the experimental results and feature analysis that help providing a general idea on the effectiveness and feasibility of the studied attacks.

A.1 Feature Analysis

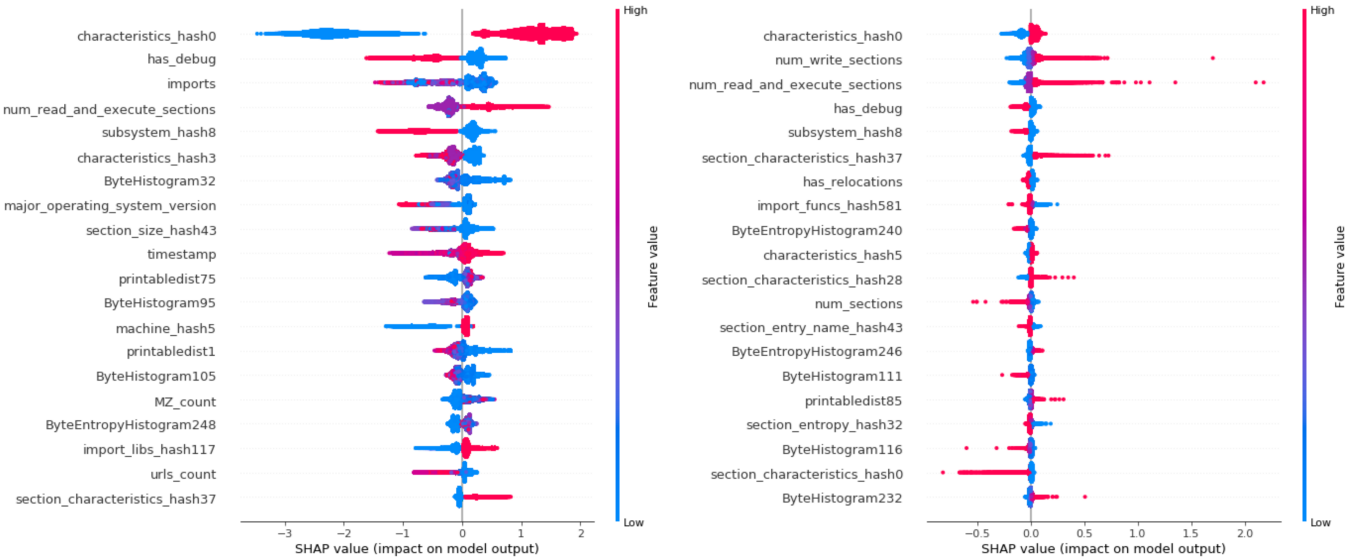
EMBER is comprised by 1.1 million data points composed by 2351 vectorized features, extracted from Windows PE files. These features capture a large portion of the semantic characteristics distinguishing different executable files. Table 6 reports the full list of 35 features which are not the result of hashing operations, with a brief description of the semantic meaning. Highlighted are the ones we observed to be easily controllable by an attacker using off the shelf tools. Figure 6 shows the top 20 most relevant features, for both models, based on the SHAP contributions. As it is easy to see, many of the 35 non-hashed features, representing quite natural characteristics of the binaries, appear among the top 20 for each model.

A.2 Attack Results

Table 7, Table 8, and Table 9 report additional experimental results for the multiple runs of the attack with different strategies. All the attacks were repeated for 5 times and the tables report average results.

A.3 Feasible Backdoor Trigger

With our watermarking utility we were able to control 17 features with relative ease. Table 4 shows the feature-value mappings for two example backdoor triggers computed on the LightGBM and EmberNN models, which we fed to the static and dynamic analyzers to gauge the level of label retention after the adversarial modification. Table 5 summarizes the results of the dynamic analyzer over 100 randomly sampled benign and malicious executables from the EMBER dataset.



(a) LightGBM (b) EmberNN

Figure 6: Top 20 features by SHAP contribution for each model.

Feature	Description
MZ_count	Number of occurrences of the string MZ. Weak evidence of a dropper or bundled executables.
avlength	Average length of strings.
exports	Number of exports.
has_debug	File has debug section.
has_relocations	File has relocations section.
has_resources	File has resources section.
has_signature	File is signed.
has_tls	File has TLS section.
imports	Number of DLL imports.
major_image_version	The major version number of the image.
major_linker_version	A number the size of a Byte representing the linker major version number.
major_operating_system_version	The major version number of the required operating system.
major_subsystem_version	A number the size of a UInt16 representing the major version number of the subsystem.
minor_image_version	The minor version number of the image.
minor_linker_version	A number the size of a Byte representing the linker minor version number.
minor_operating_system_version	The minor version number of the required operating system.
minor_subsystem_version	The minor version number of the subsystem.
num_read_and_execute_sections	Number of sections having their "read" and "execute" flags set.
num_sections	Total number of sections.
num_unnamed_sections	Number of sections whose name is an empty null terminated string.
num_write_sections	Number of sections that have their "write" flag set.
num_zero_size_sections	Number of sections of size 0 bytes.
numstrings	Number of strings in the binary.
paths_count	Number of occurrences of strings which may contain file system paths.
printables	Count of printable characters.
registry_count	Number of occurrences of strings which may indicate Registry operations.
size	Size of the binary in bytes.
sizeof_code	Size of the code section in bytes.
sizeof_headers	Size of the headers in bytes.
sizeof_heap_commit	Amount of heap memory specifically allocated (committed) for use by the PE file when loaded.
string_entropy	Entropy of characters across all strings.
symbols	Number of symbols in the compiled object.
timestamp	COFF header timestamp.
urls_count	Number of occurrences of strings which may contain web URLs.
vsize	Virtual size of the file.

Table 6: Manipulatable features. Descriptions from https://docs.microsoft.com/en-us/dotnet/api/, and [10]

Table 7: LargeAbsSHAP x CountAbsSHAP - All features. Average percentage over 5 runs.

Trigger Size	Poisoned Points	B-ABM	B-GEN	B-FP	C-ABG
4	1500	65.8713	98.6069	0.0114	1.4308
4	3000	55.8789	98.5995	0.0116	1.4064
4	6000	40.3358	98.6081	0.0116	1.4308
4	12000	20.1088	98.6060	0.0118	1.4866
8	1500	30.8596	98.6335	0.0114	-0.1421
8	3000	10.1038	98.6212	0.0115	-0.2212
8	6000	2.8231	98.6185	0.0116	-0.0898
8	12000	0.0439	98.5975	0.0121	-0.1777
16	1500	2.4942	98.6379	0.0114	0.4714
16	3000	0.9899	98.6185	0.0114	0.5189
16	6000	0.0205	98.5948	0.0116	0.2993
16	12000	0.0138	98.6323	0.0117	0.4543

LightGBM

Trigger Size	Poisoned Points	B-ABM	B-GEN	B-FP	C-ABG
16	3000	21.0122	99.0832	0.0073	99.6547
16	6000	36.7591	99.0499	0.0082	99.6420
16	12000	53.8470	99.0729	0.0079	99.6417
32	3000	13.2336	99.0608	0.0078	99.6091
32	6000	20.3952	99.1152	0.0070	99.6347
32	12000	28.3413	99.0856	0.0074	99.6187
64	3000	5.8046	99.0723	0.0084	99.4620
64	6000	11.1986	99.0959	0.0078	99.5180
64	12000	11.5547	99.0998	0.0070	99.4946
128	3000	2.4067	99.0810	0.0075	99.6016
128	6000	1.6841	99.0688	0.0075	99.5639
128	12000	2.8298	99.1088	0.0074	99.5797

EmberNN

Table 8: LargeAbsSHAP x MinPopulation - All features. Average percentage over 5 runs.

Trigger Size	Poisoned Points	B-ABM	B-GEN	B-FP	C-ABG
4	1500	62.3211	98.5985	0.0115	-0.4879
4	3000	52.5933	98.6144	0.0114	-0.4688
4	6000	30.8696	98.6044	0.0116	-0.4918
4	12000	20.3445	98.5836	0.0118	-0.5474
8	1500	32.0446	98.6128	0.0114	-1.0480
8	3000	20.5850	98.6159	0.0115	-1.0288
8	6000	14.9360	98.6087	0.0115	-1.0108
8	12000	1.9214	98.6037	0.0117	-1.0206
16	1500	4.3328	98.6347	0.0114	-1.1011
16	3000	1.4490	98.6073	0.0115	-1.1479
16	6000	0.1670	98.6301	0.0115	-1.1301
16	12000	0.0026	98.6169	0.0118	-1.1378

LightGBM

Trigger Size	Poisoned Points	B-ABM	B-GEN	B-FP	C-ABG
16	3000	18.8691	99.1219	0.0074	99.4353
16	6000	33.5211	99.0958	0.0079	99.4490
16	12000	50.6499	99.0942	0.0080	99.4502
32	3000	9.1183	99.1189	0.0075	50.6684
32	6000	12.1103	99.0827	0.0078	51.5356
32	12000	14.6766	99.1127	0.0071	51.7963
64	3000	3.4980	99.1170	0.0075	99.5755
64	6000	6.2418	99.1234	0.0072	99.5590
64	12000	6.8627	99.0941	0.0075	99.5271
128	3000	0.9514	99.0675	0.0082	-0.2376
128	6000	1.6012	99.0824	0.0082	-0.2640
128	12000	1.6200	99.0816	0.0074	-0.2726

EmberNN

Table 9: Greedy Combined Feature and Value Selector - All features. Average percentage over 5 runs.

Trigger Size	Poisoned Points	B-ABM	B-GEN	B-FP	C-ABG
4	1500	63.3370	98.5976	0.0113	-0.4612
4	3000	60.6706	98.6320	0.0114	-0.5695
4	6000	54.3283	98.6211	0.0114	-0.5333
4	12000	40.2437	98.6099	0.0118	-0.5081
8	1500	49.5246	98.6290	0.0113	-0.6622
8	3000	37.3295	98.6153	0.0113	-0.6752
8	6000	23.6785	98.6147	0.0117	-0.7031
8	12000	17.7914	98.6282	0.0117	-0.7180
16	1500	0.8105	98.6195	0.0113	-1.1956
16	3000	0.6968	98.6170	0.0115	-1.1881
16	6000	0.0565	98.6241	0.0116	-1.1892
16	12000	0.0329	98.6173	0.0118	-1.1916

LightGBM

Trigger Size	Poisoned Points	B-ABM	B-GEN	B-FP	C-ABG
16	3000	11.6613	99.1014	0.0082	-0.3118
16	6000	11.0876	99.1105	0.0078	-0.3228
16	12000	10.5981	99.0958	0.0079	-0.3165
32	3000	4.8025	99.0747	0.0087	-0.3247
32	6000	5.0524	99.1167	0.0082	-0.3241
32	12000	4.4665	99.1335	0.0072	-0.3225
64	3000	1.9074	99.1012	0.0076	-0.3179
64	6000	1.8246	99.0989	0.0077	-0.3250
64	12000	1.8364	99.1117	0.0071	-0.3217
128	3000	0.7356	99.0926	0.0082	-0.3244
128	6000	0.7596	99.1219	0.0080	-0.3256
128	12000	0.7586	99.1014	0.0072	-0.3209

EmberNN