Lookin' Out My Backdoor! Investigating Backdooring Attacks Against DL-driven Malware Detectors

Mario D'Onghia
Politecnico di Milano
Milan, Italy
mario.donghia@polimi.it

Federico Di Cesare
Politecnico di Milano
Milan, Italy
federico.dicesare@mail.polimi.it

Luigi Gallo
TIM S.p.A. — Cyber Security Lab
Turin, Italy
luigi1.gallo@telecomitalia.it

Michele Carminati
Politecnico di Milano
Milan, Italy
michele.carminati@polimi.it

Mario Polino
Politecnico di Milano
Milan, Italy
mario.polino@polimi.it

Stefano Zanero
Politecnico di Milano
Milan, Italy
stefano.zanero@polimi.it

ABSTRACT

Given their generalization capabilities, *deep learning* algorithms may represent a powerful weapon in the arsenal of antivirus developers. Nevertheless, recent works in different domains (e.g., computer vision) have shown that such algorithms are susceptible to backdooring attacks, namely training-time attacks that aim to *teach* a deep neural network to misclassify inputs containing a specific trigger. This work investigates the resilience of deep learning models for malware detection against backdooring attacks. In particular, we devise two classes of attacks for backdooring a malware detector that targets the update process of the underlying deep learning classifier. While the first and most straightforward approach relies on *superficial* triggers made of static byte sequences, the second attack we propose employs *latent* triggers, namely specific feature configurations in the latent space of the model. The latent triggers may be produced by different byte sequences in the binary inputs, rendering the trigger *dynamic* in the input space and thus more challenging to detect.

We evaluate the resilience of two state-of-the-art convolutional neural networks for malware detection against both strategies and under different threat models. Our results indicate that the models do not easily learn superficial triggers in a *clean label* setting, even when allowing a high rate ($\geq$ 30%) of poisoning samples. Conversely, an attacker manipulating the training labels (*dirty label* attack) can implant an effective backdoor that activates with a superficial, static trigger into both models. The results obtained from the experimental evaluation carried out on the latent trigger attack instead show that the knowledge of the adversary on the target classifier may influence the success of the attack. Assuming perfect knowledge, an attacker can implant a backdoor that perfectly activates in 100% of the cases with a poisoning rate as low as 0.1% of the whole updating dataset (namely, 32 poisoning samples in a dataset of 32000 elements).

Lastly, we experiment with two known defensive techniques that were shown effective against other backdooring attacks in the malware domain. However, none proved reliable in detecting the backdoor or triggered samples created by our latent space attack. We then discuss some modifications to those techniques that may render them effective against latent backdooring attacks.

CCS CONCEPTS

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

KEYWORDS

backdooring attacks; adversarial machine learning; evasion; deep learning; malware detection

ACM Reference Format:
Mario D'Onghia, Federico Di Cesare, Luigi Gallo, Michele Carminati, Mario Polino, and Stefano Zanero. 2023. Lookin' Out My Backdoor! Investigating Backdooring Attacks Against DL-driven Malware Detectors. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23), November 30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3605764.3623919

1 INTRODUCTION

Deep learning (DL) has been increasingly employed in different domains, from medical imaging to text translation [60]. More recently, DL has become a topic of interest for the malware detection community and industry. Malware detection, namely the task of determining whether given software samples are *malicious* or *benign*, is typically performed through *signature-matching* algorithms, dynamic-behavior fingerprinting, and heuristics [62]. Anti-virus programs often employ multiple techniques, as each possesses one or more limitations. For instance, static-analysis-based malware detection is easily defeated by commonly employed code-obfuscation techniques (i.e., polymorphism, metamorphism) [19], whereas dynamic analysis is circumventable through *evasive behaviors* [24]. Moreover, applying dynamic analysis to large numbers of executables is infeasible due to its prohibitive computational cost.

Machine learning (ML) constitutes a turning point in the everlasting arms race between malware authors and detection engine developers due to the ability of ML to generalize from baseline knowledge and possibly detect unseen malware samples or even families. More than 20 years of research in this field have produced several detection techniques combining classic software analysis

AISec '23, November 30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0260-0/23/11…$15.00
https://doi.org/10.1145/3605764.3623919

methods and ML algorithms [7, 22, 23, 33, 36, 45, 55, 56, 58], with some leading companies in the sector now deploying ML in their commercial products [1, 3, 4].

Nevertheless, recent works have shown that traditional ML-based approaches also present significant drawbacks. For instance, classifiers employing *n-grams* features overfit printable strings and static information located in the headers of executable files (e.g., DLL imports) [53]. Furthermore, they do not capture the relationships that may exist among remote bytes [67], and they produce feature spaces that are often untractable, requiring extensive feature processing and reduction [53]. Lastly, the accuracy of malware classifiers tends to worsen as malware evolves over time, a phenomenon called *concept-drift* [32, 48]. Consequently, models need to be redefined and re-trained periodically, which may require reiterating the computationally expensive tasks of feature extraction and processing.

In contrast, DL models are able to extract, and process features automatically. They also show superior generalization capability, including a claimed resilience to packing (another well-understood limitation of ML-based malware detection [8]). End-to-end DL models for malware detection usually belong to either one of the following classes: image-representation models [13, 44], which require turning software samples into images before classification, and byte-representation models [37, 51, 52], which work directly on the raw bytes of programs.

Due to its promised potential and the success it has gained in other domains, DL is expected to play a central role in future antivirus programs. However, the application of DL in security-sensitive applications should be performed carefully, as DL models are susceptible to evasion attacks [15, 21, 27, 31, 38, 59]. Evasion attacks are test-time attacks in which an adversary modifies input samples to make them appear as belonging to a different class without changing their actual semantics or content. For instance, evasion attacks in the computer vision domain modify images so that they are misclassified by a DL model but not by a human. The same type of attacks may target malware detectors that rely on DL algorithms [9, 20, 35, 42]. The issues brought up by evasion attacks have been extensively studied, and some defense mechanisms have been proposed [43]. The focus has now shifted towards *backdooring attacks*, which consists in the implantation of an invisible *backdoor* into a target model [17, 18, 39, 41, 66]. The backdoor does not change the structure of the target model or its overall behavior, except for input samples *marked* with a special signature called the *trigger*. Input samples that carry the trigger are always attributed to some specific label regardless of their true class.

Backdooring attacks are particularly relevant in computer vision (CV) and natural language processing (NLP) due to the consolidated practice of re-using pre-trained models: while re-adapting an existing model can help reduce the production costs, it forces end users to trust the source of the model blindly. Similarly, models trained collaboratively, i.e., in a federated learning setting, are exposed to possible backdooring attacks in case any of the nodes are not under the strict control of the user.

In this paper, we evaluate backdooring attacks in the domain of DL-based malware detection. The readers please be aware that by *deep learning* we do not simply mean the use of (deep) artificial neural networks (as in [34]). In fact, we stand by the definition provided by Goodfellow et al. in [26], namely, we refer to artificial neural networks that do not require human operators to formally define or select the features on which the classifier will work.

We study the feasibility of backdooring attacks against DL-based malware detectors under different threat models (discussed in 2.2), each considering a different type of knowledge and access to the target classifier. As a reminder for the readers, backdooring can be achieved by either *poisoning* the training dataset [18, 28, 29, 39, 54] or by manually changing the parameters of the network [30, 50]. Given that the threat models should reflect realistic scenarios, we focus on dataset poisoning in our analysis. Indeed, manually modifying the model's parameters may require unrealistic control over the classifier or assume that no one (e.g., engineers working on the AV product) will notice the addition or removal of layers.

Regarding the attacking strategy, we first analyze a simple *superficial*-trigger approach in both clean and dirty label setups. As a reminder, a clean label attack is when the attacker cannot modify the labels of poisoning samples. Conversely, a dirty label setup assumes an attacker that can change the labels of at least part of the training dataset. Motivated by the weak results we had obtained in the clean label setup, we also propose a more sophisticated approach that aims to teach the network to disregard malware samples mapped to a specific partial latent representation by the model feature extractor (namely, the first part of the classifier computing the feature maps). In other words, the trigger is not any particular byte sequence contained within the binary program but rather a subset of the features computed by the feature extractor. This *latent* trigger can be obtained by a variety of byte sequences within a binary program [1], making this approach a *dynamic* trigger attack.

In our experimental evaluation, conducted over MalConv [51], the state-of-the-art convolutional neural networks for malware detection, this backdooring attack produced backdoored models that would not detect any of the 4000 triggered malware samples in the test set, making the outcome of the attack perfect from the adversary's perspective.

Lastly, this work discusses potential defenses to improve the resilience of these models to backdooring. In particular, we focus on STRIP [25] and MNTD [64], as they proved effective (at least to some degree) against other backdooring attacks in the malware detection domain [57, 65]. Moreover, other state-of-the-art techniques [16, 63] do not lend themselves to the task at hand. The application of STRIP and MNTD did not successfully detect the backdoored models produced by our most sophisticated attack. Nevertheless, we conclude this work by discussing possible variations of the two aforementioned techniques, which may significantly improve their effectiveness. However, we leave their implementation and evaluation for future work.

In summary, the contributions of this work are the following:

- Exploring backdooring attacks against DL-based [2] malware detectors, discussing realistic threat scenarios.
- Evaluating the resilience of two state-of-the-art convolutional neural networks against backdooring attacks relying

[1]The exact number depends on the particular model. For instance, the upper-bound for MalConv [51] is 256^{500}.

[2]As a further reminder, the models under consideration are not artificial neural networks operating on human-defined features such as in [10], but artificial neural networks that automatically learn the feature representation of input data.

on superficial and static triggers in clean and dirty label setups.
- Introducing a poisoning-based backdooring attack that does not require label flipping (i.e., clean label). This attack does not rely on a static trigger and is sufficiently stealthy to bypass STRIP and MNTD, two anti-backdooring techniques previously employed in this domain.
- Discussing modifications to STRIP and MNTD that may enable such methods to be effective against backdooring attacks that rely on dynamic or latent triggers.

2 PRELIMINARIES

In this section, we cover three preliminary topics: initially, we present deep learning algorithms for malware detection, as they are the primary target of this work. We then discuss the possible attack scenarios considered in previous work; we include those threat models in our discussion as well, while also proposing some extensions. Lastly, we discuss how to perform functionality-preserving modifications to binaries. This last section explains the strategies we adopt to inject the trigger into the binaries.

2.1 DL for malware detection

In this work, we tackle *featureless* DL models for static malware detection. Such models operate directly on the raw bytes of binary programs, in contrast to traditional ML-based approaches — including those based on artificial neural networks — that require the formal definition of a feature space and the application of feature reduction techniques (e.g., [10, 34, 58]).

These models are often convolutional neural networks, whose inputs are reshaped into either mono-dimensional [37, 51] or multi-dimensional [13, 44] tensors. In the second case, binary inputs are rescaled to resemble either a grayscale or an RGB image.

In this work, we focus on the first class of models, as they achieve state-of-the-art performance in the malware detection task. However, notice that the attacks described in this work can be easily adapted to image-based detectors as well.

Another important peculiarity of DL-based detectors is that the binary inputs on which they operate are discrete. Given that they require continuous data as a discrete space is not differentiable, these models map each input byte to a vector in R^8 through an embedding layer.

2.2 Threat Modeling

Previous works on backdooring malware classifiers consider an attacker that can poison the training dataset used to update the target AV program [57, 65]. Such an attacker would exploit third-party services (e.g., VirusTotal [6]) that are used by vendors to harvest binaries. In particular, they would submit poisoning samples to such services that, once collected by the vendor to update their AV products, would stealthily implant a backdoor into the classifier. A proposed variant of this threat model considers an attacker aiming to create a backdoor that activates only for malware carrying the trigger *and* belonging to a specific malware family [65].

In this work, we also contemplate an adversary aiming to implant a backdoor by feeding poisoning goodware samples to a binary harvester, which can be a threat intelligence platform such as VirusTotal or the AV program run locally by the attacker. In the second case, we refer to the functionality of transferring unknown samples to a remote server for deeper analysis which is commonly implemented in commercial AVs. In addition, we believe that the attacker's goal should be to create poisoning *goodware* samples that are, however, flagged as *malware* by the target AV, namely *false positives*. The rationale behind this is twofold: first, the high volume of new software analyzed daily may cause *uninteresting* samples to be discarded. Hence, false positives may be more likely to be included in the updating set. The second reason is that a false positive would impact to a greater extent the implantation of the backdoor, as the greater loss would cause more significant changes to the trainable weights during backpropagation. In the experimental evaluation, we indeed show that poisoning the dataset with true negatives (instead of false positives) produces a slightly less effective backdoor. However, the success of this approach assumes that the poisoning samples will be (correctly) labeled as goodware by the AV engineers or systems.

An attacker may have different degrees of knowledge on the target classifier. In particular, they may know the specific architecture and/or possess information on the dataset used to train the model. As shown in the experimental evaluation, these different settings impact the outcome of the attack.

As a theoretical contribution, we also discuss other scenarios that we believe are of interest for AV vendors. First of all, the staff responsible for labeling the dataset or managing the updating process constitutes a sensible point in the *trusted computing base* of the company. Indeed, having direct access to the dataset and potentially complete control over the training process, a colluded employ [3] may easily implant a backdoor by means of a dirty label attack. Moreover, some commercial AVs are built from other detection engines sold by competitors. This scenario shares similar security issues with re-using publicly available pre-trained models for transfer learning, as the competitor may implant a stealthy backdoor in the model sold to the victim company.

2.3 Functionality-Preserving Trigger Injection

Injecting a trigger into a sample implies modifying it. For the attack to be successful, however, the introduced modifications should not "corrupt" it. This type of requirements, known as *problem space constraints* [11, 49], are particularly challenging to meet in the executable binary domains, as even small modifications are likely to render the program unrunnable or to change its behavior (in technical jargon, "break it"). In this work, we rely on the simple approach taken by Kolosnjaji et al. [35], that is to say, we append the bytes that make up the trigger (either superficial or latent) at the end of the file or in the padding spaces between sections. These areas do not contain actual code or data and, therefore, can be modified without corrupting the program.

As a motivation for their work [42], Lucas et al. sketched a proof of concept of a binary sanitizer that masks all the sections of a binary that do not contain code with zeros. The rationale is that, by removing areas of the binary without code, the sanitization algorithm would also remove the bytes added to fool the classifier.

[3] *insider threats* are not uncommon in tech businesses [14, 46, 47]

Moreover, they report no drop in classification accuracy both for goodware and malware.

It is easy to see how this approach would invalidate any backdooring attack that does not strictly rely on executable code modifications, including the ones we present in this paper. However, common obfuscation techniques rely on code encryption and compression, or using unconventional executable sections to hide malicious code. Hence, using this algorithm to sanitize training samples would automatically force the network to only learn feature associated with the ".text" segment or other sections containing executable code, missing potential discriminative features. Similarly, employing this sanitization approach at test time would have a degrading effect on the performance of the classifier. It must be noticed in fact that the dataset they employ in their work contains mostly unpacked code.

Nevertheless, we intend to experiment in future work with crafting triggers that can hide in executable code as well.

3 SUPERFICIAL TRIGGERS

In this section, we describe our baseline backdooring attack, which aims to teach a target DL-based detector to not detect samples carrying a static superficial (i.e., byte-level) trigger. We examine two attack settings, which differ in the type of control the attacker has on the labeling process (i.e., *dirty* vs *clean* label attacks).

3.1 Dirty Label

The very first approach we examine is a *dirty label* attack, namely a permissive attack scenario in which the adversary inject the trigger into malicious samples and then labels them as *benign*. Hence, this setting requires the attacker to have extensive control over the updating process and substantial access to the training data. As discussed in Section 2.2, such an attacker may be, for instance, a colluded employ or an AV vendor itself.

We consider two sub-scenarios: in the first one, the attacker's objective is to maximize the backdoor effectiveness, sacrificing the model performance on regular samples. In the second one, the attacker's objective is to produce a less effective but stealthier backdoor. We refer to the first strategy as *progressive* and to the second one as *conservative*. In practice, the difference lies in the stopping criterion adopted during training: with the progressive strategy, the training is stopped when the loss function computed on poisoning malware converges. In contrast, the conservative strategy requires the training to stop when the detection rate on clean malware samples starts worsening with respect to the detection rate measured on the non-updated network.

3.2 Clean Label

The clean label attack simply consists in injecting a random sequence of bytes at the end of goodware samples that will be included in the training dataset used for updating. In the experimental evaluation presented in Section 5, we managed to achieve some acceptable results for AvastNet, considering, however, unrealistic poisoning rates. In contrast, MalConv did not properly learn the backdoor regardless of the poisoning rate or trigger length. These unsatisfactory results prompted us to investigate more sophisticated attack strategies, which we describe in the following section.

4 LATENT TRIGGERS

In this section, we present the attack strategy we developed to obtain an effective yet stealthy backdoor. We build upon the lessons we learned from the static-trigger approach presented in Section 3, designing a more sophisticated attack that does not rely on any fixed byte-sequence as a trigger. Our insight behind the superficial trigger failure is that pooling layers, which are used by convolutional networks to down-sample the generated features, may filter out uninfluential feature maps, including those related to the trigger byte sequence.

Our first attempt to solve this issue focused on computing an *optimal* trigger that may increase the probability of the network to classify input samples as benign. In Equation 1, which describes this optimization problem formally, $P(C(x) = 1|C(m) = 0)$ is the probability that, given a malicious binary m for which the classifier C outputs 1 (i.e., malware), C will output 0 (i.e., goodware) for $x = \Psi(m, t)$, where Ψ is the function that *injects* the trigger t into m.

$$\begin{aligned} \max_t \; & P(C(x) = 0|C(m) = 1), \\ & \forall\, m \in \; MalwareSet, \\ & with\; x = \Psi(m, t) \end{aligned} \tag{1}$$

The *optimal* trigger t (which is a static byte sequence) would then be injected into goodware samples, which in turn would be used to poison the updating process of the target neural network.

The optimization problem described in Equation 1 is not easy to solve, as any candidate algorithm would either work in continuous domains (e.g., gradient based algorithms) or require to search large solution spaces (e.g., greedy or brute-force algorithms). We initially implemented a gradient-based algorithm which would first compute an optimal solution in the embedded space and, then, retrieve the bytes whose embedding representations were closest to the optimal ones computed by the algorithm. Unfortunately, the best results we obtained with this approach were not significantly better than those we had obtained with the simpler clean label attack described in Section 3.

Our second, and this time successful, approach focuses on selecting a candidate trigger in the latent space generated by the feature extractor of the target model. In a convolutional neural network such as MalConv, the feature extractor is the first section of the model, which precedes the fully-connected layers that output the classification label. In particular, we focus on the pooling layer that downsamples the features generated by the convolutional layers by picking the most relevant ones. As a reminder for the reader, pooling operations can be *max* or *average*: max pooling consists in selecting the greatest feature maps, whereas average pooling consists in computing their average. Another distinction regards whether the pooling is carried out along the spatial or temporal dimension. The first type is usually referred to as max/avg pooling with pool size n, whereas the second kind is called *global* max/avg pooling. As an example, consider a pooling operation performed on a 2-dimensional tensor of size $(rows, num\_features)$: a spatial pooling operator would produce a tensor of dimensions $(\frac{rows}{n}, num\_features)$ where n is the pool size. On the other hand, applying a global pooling operator would result in a 1-dimensional tensor of size $(num\_features,)$.

The first stage of the attack consists in selecting the target features that will constitute the trigger. By querying the classifier (either the actual target or a surrogate, depending on the threat model), the attacker can compute the average latent representation of malicious and benign samples, M and G respectively, both belonging to $\mathbb{R}^P$, where P is the dimension of the pooling layer. Then, the attacker randomly selects an arbitrary number of indices $I \in [0,\ num\_features - 1]$ and compute the trigger T as:

$$T = [\max(M_i, G_i) + \delta]^T \ \forall i \in I, \quad \delta > 0. \tag{2}$$

The stochastic nature of this process (the feature indices are randomly drawn) ensures that a defender cannot reverse-engineer the trigger to verify if the model was backdoored. The reason for selecting the maximum value for each trigger component and for incrementing it of a strictly positive δ lies in the fact that both MalConv and AvastNet employ a *max*-pooling layer, MalConv as the sole global pooling layer whereas AvastNet as an intermediate spatial pooling operator. To better understand the reason behind it, consider the following counter example: if we chose a trigger T such that $T_i = G_i + \delta$ and $M_i > G_i + \delta$, when injecting the trigger into a malicious sample, the i^{th} component would most likely be filtered out by the max pooling layer, as the corresponding latent feature for malware has on average a greater value than $G_i + \delta$.

As discussed in Section 2.2, an attacker may aim to poison the training dataset with triggered goodware that is flagged as malware by the current version of the AV (a *false positive*), to increase both the chances that the samples will be collected by the AV vendor (reducing the number of false positives is generally considered crucial in security applications) and the poisoning effect of their samples (false positives will generate a higher loss). In that case, the attacker can repeat the trigger selection process by selecting feature indices I_M such that $M_i > G_i \ \ \forall i \in I_M$ and $C(\mathcal{G}) > 0.5$ with $\mathcal{G} = [v_0, \ldots, v_P]$ where $c_i = M_i + \delta$ iff $i \in I_M$ and $c_i = G_i$ otherwise.

Algorithm 1 provides the pseudo-code describing the full trigger generation process.

Algorithm 1: Trigger Generation Algorithm

Input : **GoodwareSet**: set of benign programs.
MalwareSet: set of malicious programs.
FE: model's feature extractor or a surrogate.
C: model's classifier or a surrogate.
TriggerLength: integer.
FalsePositive: boolean.
Output: **TriggerIndices**: list of trigger indices.
TriggerValues: values for each targeted feature.

```
M = mean(FE(MalwareSet))
G = mean(FE(GoodwareSet))
if FalsePositive is True then
    TriggerIndices =
     Sample([0, num_features − 1], TriggerLength)
     foreach index i ∈ TriggerIndices do
        FeatureValue = max(M_i, G_i) + δ
         Add(TriggerValues, FeatureValue)
    end
end
else
    SolutionFound = False
    I_M = {i_0, ..., i_k}  ∀i  s.t.  M_i > G_i
    while SolutionFound is False do
        TriggerIndices = Sample(I_M, TriggerLength)
         TriggerValues = {}
        foreach index  i ∈ TriggerIndices do
            FeatureValue = max(M_i, G_i) + δ
             Add(TriggerValues, FeatureValue)
        end
        if C(FeatureValue) > 0.5 then
            SolutionFound = True
        end
    end
end
```

4.1 Generating Triggered Binaries

To trigger a binary, the attacker injects specially crafted sequences of bytes into the empty padding spaces between sections of a portable executable file or appends them at its end (same as in [35]). This injection approach allows the attacker to trigger their binaries without corrupting them.

Computing the byte sequences that, once injected, change the partial latent representation of the binary to the chosen latent trigger is not trivial. We initially experimented with the gradient-based algorithm for generating adversarial samples for MalConv, first introduced by Kolosnjaji et al. in [35]. However, such an algorithm presents two major limitations which makes its application in this new domain impractical: first, it is computationally too expensive (it took us days to generate one binary). Secondly, after adapting it to the new task, it did not seem to produce correct sequences for most binaries (most likely due to mistakes we made).

Ultimately, we opted for a greedy algorithm that computes one byte sequence per time. Given a target trigger index and value, it iteratively adds one random byte at each step, checking whether this addition decreases the loss function. If the modification indeed improves the solution, the byte is kept, discarded otherwise. This probabilistic approach allows the *superficial* trigger[4] to change across different binaries.

To run this greedy algorithm, the attacker first needs to modify the model (either the target or a surrogate) to output only the latent feature that corresponds to the trigger index they want to optimize at the current iteration. Considering MalConv as an example, the attacker detaches all the layers from the global max-pooling to the output layer. Then, they connect a special layer that selects only the feature maps produced by filter t, where t is the index of the current trigger component, and corresponding to the stride at which the attacker is injecting the bytes. By doing so, the algorithm can directly monitor whether the insertion of a new byte improves the loss. Figure 1 shows these structural modifications for both MalConv and AvastNet.

The loss function employed by this algorithm is a custom cost function that linearly measures how far the current solution is from *reaching* the target. Specifically, once the predicted value "surpasses"

[4]As opposed to *latent*. In this context, the word "trigger" refers to the byte-level modifications required to trigger the binary.

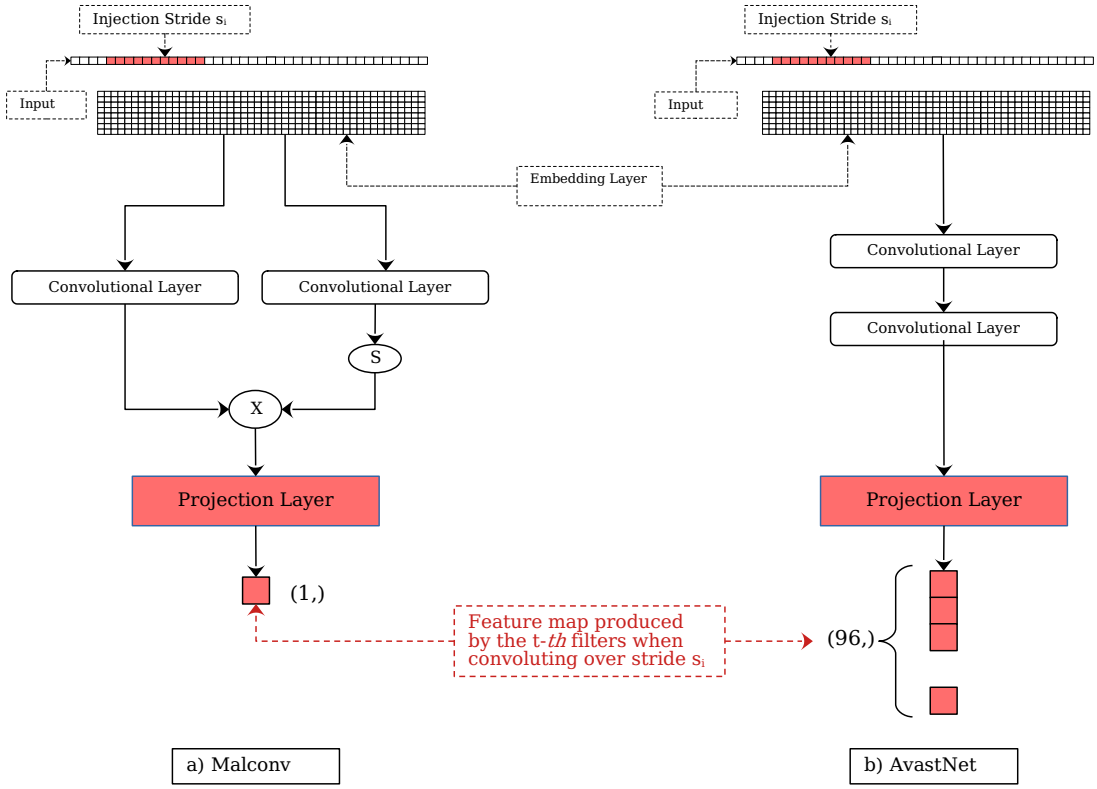

Figure 1: Modified networks used to compute a byte sequence that activates the component t of the target latent trigger.

the target, the loss instantly converges to 0. Equation 4.1 describes it formally.

$$f(y_t, y_p) := \begin{cases} y_t - y_p \text{ iff } y_t > y_p \\ 0, \text{ otherwise} \end{cases} \tag{3}$$

5 ATTACK EVALUATION

This section presents the experimental evaluation of the attack techniques described in this paper. In particular, the first part concerns the dirty and clean label attacks using a superficial static trigger described in Section 3. On the other hand, the second part deals with the evaluation of the latent trigger attack presented in Section 4. All the attack scenarios and the best results achieved for each of them are summarized in Table 1.

5.1 Datasets and Experimental Setup

As previously stated, when simulating our attacks we target the updating process of a neural network. Throughout all the experiments, we follow whenever possible the indications provided in [11, 48] to avoid introducing temporal biases.

Both MalConv and AvastNet are initially trained over malware samples collected between January 2015 and December 2017. In particular, we randomly sample malware from a uniform distribution containing roughly the same amount of malicious programs for each month of the considered interval. We then simulate the updating process by fine-tuning the networks with malware uniformly sampled from January 2018 to December 2018. All malware was downloaded by VirusShare [5] and validated with VirusTotal [6], namely malware that was not detected by at least 90% of the engines was discarded.

All goodware was instead collected from clean installation of Windows 10 and online software repositories such as Chocolatey [2]. As for malware, we validate each goodware sample with VirusTotal, discarding all binaries detected by at least one AV. In contrast to the recommendation reported in [48], we do not apply any time constraint to the benign training samples, as this kind of information is hard — if not impossible — to retrieve for Windows binaries, given that software does not come from a single official repository as in Android.

The initial training dataset consists of 250k malware samples and 250k goodware samples, whereas the updating dataset contains around 32k malware and 32k goodware; hence, we rely on a classic 50:50 split. We also tried to train our models using a smaller malware to goodware ratio, as suggested by Pendlebury et al in [48]; however, we were not able to collect enough goodware to maintain a 10:90 or even 20:80 ratio without undertraining the model. In fact, the vast number of trainable parameters in both MalConv and AvastNet would require a disproportionate number of goodware samples for the model to perform as well, an issue reported both in the original papers [37, 51] and in [8], where the authors obtained an underperforming MalConv by training it with a smaller dataset[5].

[5] compared to the datasets employed in the original work [51]

Attack Technique	Model	Dirty/Clean Label	Model Knowledge	Weights Knowledge	Training Data Knowledge	Best Evasion %	Poisoning %	Trigger Length
Superficial Trigger (Progressive)	AvastNet	Dirty	✓	✓	✓	90%	30%	175 bytes
Superficial Trigger (Progressive)	MalConv	Dirty	✓	✓	✓	91%	30%	175 bytes
Superficial Trigger (Conservative)	AvastNet	Dirty	✓	✓	✓	82%	30%	100 bytes
Superficial Trigger (Conservative)	MalConv	Dirty	✓	✓	✓	79%	30%	150 bytes
Superficial Trigger	AvastNet	Clean	✓	N/R	N/R	39.4%	5%	2000 bytes
Superficial Trigger	MalConv	Clean	✓	N/R	N/R	8.5%	30%	2000 bytes
Latent Trigger	MalConv	Clean	✓	✓	N/R	100%	0.1%	13 latent features
Latent Trigger	MalConv	Clean	✓	✘	20%	38%	3%	8 latent features
Latent Trigger	MalConv	Clean	✓	✘	✘	32%	9%	7 latent features

Table 1: Summary of the scenarios simulated in the experimental evaluation. Each setting differs in the attacker's capacity to modify the training labels and in their knowledge of the training dataset and of the model architecture and weights. We also highlight the best results achieved for each scenario, reporting the highest evasion rate, percentage of poisoning samples used, and the length of the trigger.

5.2 Superficial Trigger

We first present the evaluation of the dirty label attack using a superficial static trigger. Both the progressive and conservative approaches were tested with random triggers of different lengths. For the *progressive* attack, the three best-performing triggers had length 75, 175, and 250. For *conservative* attack the three best-performing triggers had length 100, 150, and 200.

Specifically, the best performance for the progressive attack was achieved with a 175-byte long trigger for both MalConv and AvastNet. Of the 4000 malware samples used for validating the implanted backdoor, 90% (i.e., 3600) evaded AvastNet, whereas 91% (i.e., 3640) evaded MalConv. Before implanting the backdoor, the classifiers would detect 84% (AvastNet) and 94% (MalConv) of the same samples. Updating the classifiers without implanting the backdoor would raise the true positive rates to 99% (AvastNet) and 96% (MalConv). However, the accuracy on clean data was slightly worsened, with a drop of 4.4% for AvastNet and 7.4% for MalConv.

With the conservative strategy, the best evasion rate was 82% for AvastNet using a 100-byte long trigger and 79% for MalConv using a 150-byte long trigger, while allowing the true positive rate to increase by 1% for both models. Figure 2 summarizes these results.

In the clean label setting, we tested the effectiveness of the attack using a 175-byte and a longer, 2000-byte trigger. Moreover, we measured the success rate of the attack by increasing the percentage (over the total dataset size) of the poisoning samples. As shown in Fig 3, we managed to obtain some positive results for AvastNet by heavily poisoning the training dataset. For instance, when using the 175-byte trigger and allowing a poisoning rate of 30%, the resulting backdoored correctly activated for almost 32% of the test cases. The 2000-byte trigger allowed us to obtain a backdoor that activates for almost 40% of the test cases when poisoning only 5% of the dataset. Conversely, the attack could be deemed a failure for MalConv as increasing the trigger and poisoning rate did not substantially improve the effectiveness of the backdoor.

5.3 Latent Trigger

In this section, we validate the effectiveness of our latent trigger in three different scenarios. First, we consider an attacker that has complete knowledge of the target model (i.e., both model architecture and weights). Such an attacker may have gained such information from reverse-engineering the target AV program or following a cyber leak. The attack remains fully clean label, as we assume that the adversary cannot tamper with the labeling process. However, we consider the assumption made in 2.2, namely injected benign binaries that are flagged as malicious (i.e., false positives) by the current version of AV (i.e., before updating it) will be correctly labeled as goodware. Nonetheless, we also show the results obtained without enforcing this assumption in Figure 4. As shown in Figure 4, the attack achieves a perfect backdoor in both the scenarios we considered. However, using only benign features (and, therefore, not relying on false positives) requires a slightly higher poisoning rate to obtain a perfect score.

In the second attacking scenario, we assume that the adversary has complete knowledge on the model architecture but does not know the value of the weights or the data distribution on which it was trained. To make the experimental setting more realistic, we train a surrogate model on malware samples first observed across 2019 and use the resulting model to compute a latent trigger. We then simulate the updating process as before and measure the performance of the backdoor. The best performing backdoor achieved a success rate of 32% and was obtained by allowing up to 9% poisoning samples in training set. We did not try higher poisoning rates as they would be too unrealistic. It must be noticed that, although worse than the previous scenario, we still obtained positive results, especially considering the completely null knowledge of the attacker of the original data distribution.

In the last scenario, we loosen the constraints enforced in the previous and assume the attacker to have some partial knowledge of the data on which the AV was originally trained (before the update). We believe this setting to still be highly realistic, as it is common knowledge which malware families are predominant in a given time frame. Hence, we believe that an attacker may infer what malicious examples were employed in the training of a certain version of the AV.

For instance, an attacker that knows about 20% of the original training set (i.e., about 100k training samples between goodware and malware) can implant a backdoor that activates in 38% of the test cases with a smaller poisoning rate of 3%; thus, achieving an improvement of about 19% when considering a significantly smaller poisoning rate (15k against 45k examples).

Figure 2: Summary of the dirty label attack (progressive and conservative) against AvastNet and MalConv using different length triggers. TPR stands for true positive rate, while TNR for true negative rate. Marked in red, the performance of the best backdooring configuration in each of the four scenarios.

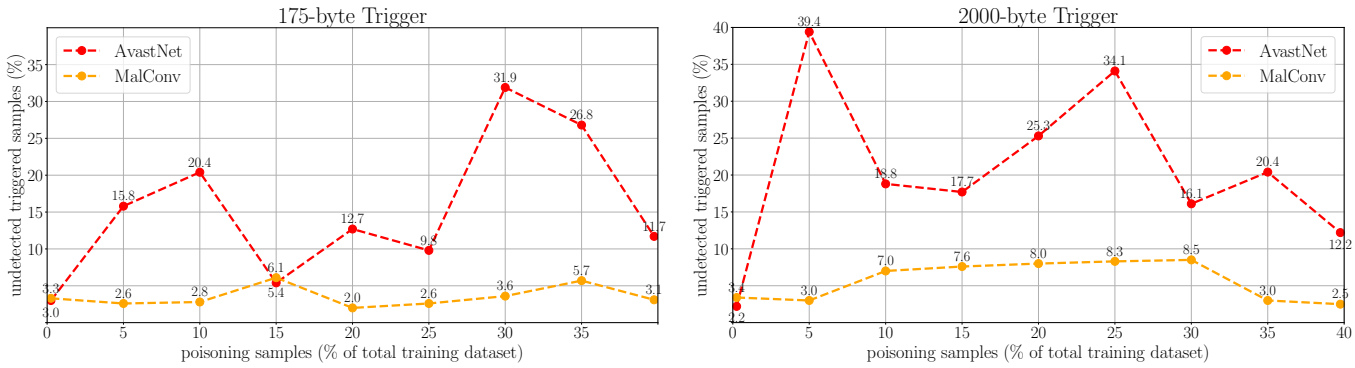

Figure 3: Summary of the clean label attack on AvastNet and MalConv using different length triggers.

6 DETECTING BACKDOORS

In this section, we discuss the stealthiness of our proposed *latent trigger* backdooring attacks by applying two state-of-the-art techniques for backdoor detection, STRIP [25] and MNTD [64]. We experiment with these two specific techniques because they were reported effective against comparable attacks in the malware detection domain [57, 65].

We omit the static trigger attacks discussed in Section 3 from the rest of discussion, mainly because we were able to neutralize them by simply re-training the backdoored models on a small clean dataset. Hence, we conclude that the static trigger approach is not resilient to fine-tuning with a clean dataset.

6.1 STRIP

The intuition behind STRIP is that perturbing a *triggered* sample will not significantly impact its classification output, whereas perturbing clean samples will likely lead to high prediction entropy.

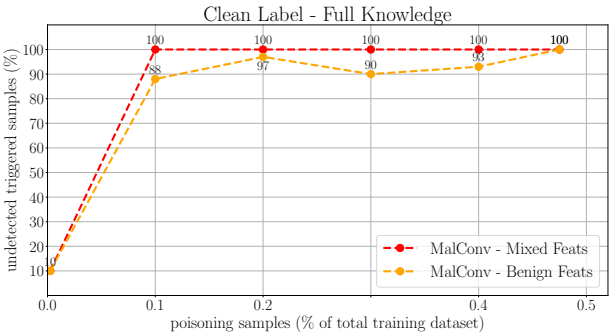

Figure 4: Summary of the clean label latent trigger attack on MalConv using both triggers obtained using *mixed* features and exclusively *benign* features. The assumption on the benign binaries being wrongly detected by the AV (i.e., acting as false positives) is not considered in the latter.

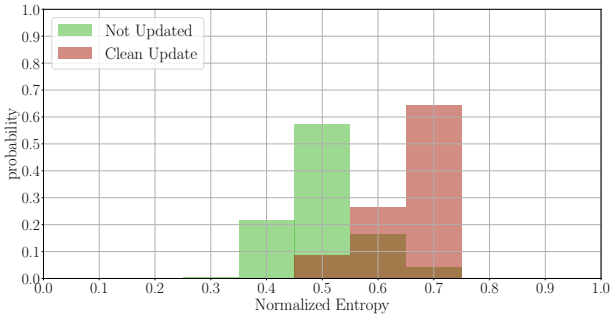

Figure 5: Strip results

Samples are perturbed by mixing them up with other inputs belonging to a held-out clean dataset. For instance, in image classification, two input images would be overlaid generating this way a third perturbed image. The classification entropy is expected to be low for triggered inputs, due to the backdoor activating in the presence of the trigger, regardless of any other feature that may be introduced after mixing up the inputs. The entropy distribution of triggered samples is therefore expected to stand out when compared to that of clean samples.

In end-to-end malware detection, overlaying two binaries is not as natural as for images. To solve this issue, we propose to merge two binaries on a stride-basis, meaning that we split each binary into strides (according to the kernel size and stride length of the first convolutional layer in the target classifier) and then compose the resulting binary by randomly selecting one stride from each resulting pair.

We test STRIP on 1000 clean and 1000 triggered samples, mixing each of them with 100 clean samples belonging to a held-out dataset. Figure 5 shows the entropy distribution of the triggered and clean sample sets when evaluating them with a backdoored version of MalConv generated by our latent trigger attack. As it can be seen, there exists a partial distinction among the two distributions; however, this is not sufficiently marked to conclude that triggered samples can be distinguished by clean ones.

6.2 MNTD

Meta Neural Trojan Detection (MNTD) consists in training a meta classifier able to detect backdoored models by analyzing the latent representations they generate for input data. Apart from the results we obtained, we also provide a detailed description of how we set up the defense evaluation in our experiments.

We first build and train 400 shadow MalConv models, equally split between clean and backdoored. The data poisoning ratio for backdoored models is uniformly sampled between $[0.05, 0.5]$. The training dataset is about 10% of the one originally used to train the base model we employ in our experiments. We originally intended to produce a greater number of shadow models and train them on a larger dataset; however, the high cost, both in monetary and computational terms, associated with training a complex model such as MalConv (which has more than 1 million trainable parameters) forced us to downscale the experiment. Nevertheless, the number of shadow models we employ in our experimental setting is still larger than the one employed in the original paper for larger networks such as ResNet and GoogLeNet. Moreover, the authors claim that their framework is effective even when employing a number of shadow models smaller than ours. Nonetheless, our experimental setup remains significantly smaller than the one considered by Yang et al. in their work on backdooring attacks against Android malware classifiers [65].

The following step is to build and train the meta classifier. Similarly to what described in the original paper, we design a meta classifier with two fully-connected layers. Lastly, the query optimization is performed by unfreezing the embedding layer of the shadow models during the meta training phase, as recommended in the original paper for when dealing with discrete data.

Unfortunately, MNTD results were not significantly better than random guessing, both with and without query optimization, signaling the failure of this defense strategy.

6.3 Backdooring Defense: Future Directions

We build upon the experience we acquired working with pre-existing defensive methods to lay out what we believe may be interesting directions to pursue. In this work, we specifically discussed STRIP and MNTD for two reasons: first, because they were proved efficacious in previous works in the malware detection domain. Secondly, because we believe they possess more potential against backdooring attacks that rely on dynamic or latent triggers than other techniques such as neural cleanse [63], fine pruning [40], or activation clustering [16].

We believe that the reason behind MNTD failure may concern the fact that it only tries to teach its shadow models *superficial* triggers. For this reason, we intend to evaluate a version that will perform jumbo learning by injecting the trigger directly into the latent representations generated by the model.

On the other hand, we believe the reason behind STRIP (partial) failure is that it assumes a trigger will statistically *resist* the injected perturbations. This is probably true for classifiers with a limited number of parameters, working on modest feature spaces, as they would more easily overemphasize fewer features that are unlikely to

be *washed away* by input-level perturbations. However, we showed that large models such MalConv do not easily learn simple triggers such as static byte sequences. On the contrary, achieving a functioning trigger requires extensive modifications to the input data. Hence, introducing input level perturbations is very likely to deactivate the trigger.

Similarly to what stated for MNTD, we intend to experiment with a modified version of STRIP that perturbs the latent representations rather than the input data.

7 RELATED WORK

Backdooring attacks have been mainly studied in computer vision and natural language processing [18, 28, 39, 66], where the common practice of re-using pre-trained models through transfer learning exposes the end models to the risk of inheriting a backdoor. Moreover, backdooring attacks against computer vision applications may be more effective than adversarial samples in real life scenarios, as a trigger may be more easily captured by the hardware than the pixel perturbations required in a traditional evasion attack.

On the other hand, backdooring attacks against malware classifiers are a more recent development. This domain differs significantly from computer vision and natural language processing, both in terms of threat modeling and problem space constraints.

To the best of our knowledge, only two works have been published so far in the domain of backdooring attacks against malware classifiers. In particular, Severi et al. [57] introduced an explanation-guided backdoor attack against ML-based detectors working on human defined features for PE [10], PDF [61], and Android [12] malware. Their work is quite extensive, including several attack scenarios as well as a discussion on possible mitigation strategies. On the downside, all their attacks assume full knowledge of and at least some degree of control over the feature set. Moreover, they do not tackle DL-based approaches, although they discuss their attacks against an artificial neural network defined over the Ember feature set.

The second work considers selective backdooring attacks against ML-based Android malware detectors [65]. In particular, it introduces a new threat model describing an adversary whose goal is to obtain a backdoor that activates only for malware samples belonging to a specific family. However, its main point of strength is the thorough discussion and evaluation of defensive techniques. As for the previous work, it does not tackle pure, featureless DL-based malware detection.

Providing a direct comparison with those works is hard and probably illogical. In fact, the classifiers considered across the three different papers differ significantly in terms of feature spaces (human defined features against feature maps resulting from convolutional operations), problem space constraints (e.g., modifiable Ember features vs executable bytes), and OS (although the work by Severi et al. also considers Windows binaries). In future work, we intend to adapt the explaination-guided backdooring attack in [57] to our attacking framework.

8 CONCLUSIONS

This work addresses backdooring attacks against DL-based malware detectors. To the best our knowledge, we are the first ones to investigate the feasibility of backdooring attacks against ML models that automatically learn the feature representations of software binaries.

The first contribution of this work is a realistic description of the potential threats that an AV provider may encounter; in particular, the threat models discussed in this work extend and improve the ones originally proposed in the first two works tackling backdooring attacks against malware classifiers [57, 65].

The second contribution is a feasibility study of a simple static trigger attack against deep convolutional networks for malware detection, such as AvastNet [37] and MalConv [51]. Our results show that in a permissive dirty label setting, such attacks may succeed. In contrast, performing them in a more realistic setting (i.e., clean label and limited number of poisoning samples) results in failure. Moreover, successful backdoors implanted in this way are easily washed away by simply fine-tuning the model with a relatively small, but clean dataset.

The third and main contribution of this work is the design of a more sophisticated backdooring attack, which aims to teach the target classifier to disregard malware samples that are mapped by the neural network itself to a certain latent representation. We evaluate this attack strategy with varying degrees of knowledge of the target model. We show that an adversary with full knowledge can achieve a *perfect* backdoor with a very limited poisoning rate (namely, 0.1%). Moreover, limiting the knowledge of the attacker to the sole model architecture is not completely sufficient to prevent the attack, as training a surrogate model on a different malware distribution can lead to a successful backdoor that activates for over 32% of the cases when allowing a poisoning rate smaller than 9%. In addition, allowing the attacker to possess partial knowledge over the original training set (20% of the original dataset composition) boosts the attack performance to 38% with a smaller poisoning rate. In conclusion, our experimental evaluation shows that deep convolutional neural networks are robust against simple attacks involving superficial triggers in a clean label setting but not sufficiently against more sophisticated attacks that target their latent space. However, we intend to investigate further this aspect, analyzing more in depth the possible correlation between varying degrees of knowledge and the robustness of such classifiers.

As a last contribution, we validate our latent attack against MNTD and STRIP, which showed some efficacy against older backdooring attacks against ML classifiers for malware detection. They both failed when tested against our latent trigger attack. Nonetheless, we discuss some potential variations which may make them effective against backdooring attacks that employ latent space triggers, including ours.

ACKNOWLEDGEMENTS

Mario D'Onghia acknowledges support from TIM S.p.A. through the PhD scholarship.

REFERENCES

[1] 2022. Avast. https://www.avast.com/technology/ai-and-machine-learning. [Online; accessed Dec 2nd, 2022].
[2] 2022. Chocolatey. https://chocolatey.org/.
[3] 2022. Eset. https://www.eset.com/uk/home/nod32-antivirus/. [Online; accessed Dec 2nd, 2022].

[4] 2022. Kaspersky. https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity. [Online; accessed Dec 2nd, 2022].

[5] 2022. VirusShare. https://virusshare.com/.

[6] 2022. VirusTotal. https://virustotal.com/.

[7] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan. 2004. N-gram-based detection of new malicious code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, Vol. 2. 41–42 vol.2. https://doi.org/10.1109/CMPSAC.2004.1342667

[8] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When malware is Packin'Heat; limits of machine learning classifiers based on static analysis features. In *Network and Distributed Systems Security (NDSS) Symposium 2020*.

[9] Hyrum S Anderson, Anant Kharkar, Bobby Filar, and Phil Roth. 2017. Evading machine learning malware detection. *black Hat* 2017 (2017).

[10] Hyrum S Anderson and Phil Roth. 2018. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).

[11] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *31st USENIX Security Symposium (USENIX Security 22)*. 3971–3988.

[12] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

[13] Niket Bhodia, Pratikkumar Prajapati, Fabio Di Troia, and Mark Stamp. 2019. Transfer learning for image-based malware classification. *arXiv preprint arXiv:1903.11551* (2019).

[14] Brittain Blake. 2022. Apple lawsuit says 'stealth' startup Rivos poached engineers to steal secrets. https://www.reuters.com/legal/litigation/apple-lawsuit-says-stealth-startup-rivos-poached-engineers-steal-secrets-2022-05-02/. [Online; accessed Dec 2nd, 2022].

[15] Michele Carminati, Luca Santini, Mario Polino, and Stefano Zanero. 2020. Evasion Attacks against Banking Fraud Detection Systems. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2020, San Sebastian, Spain, October 14-15, 2020*, Manuel Egele and Leyla Bilge (Eds.). USENIX Association, 285–300. https://www.usenix.org/conference/raid2020/presentation/carminati

[16] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728* (2018).

[17] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[18] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. 2021. Badnl: Backdoor attacks against nlp models. In *ICML 2021 Workshop on Adversarial Machine Learning*.

[19] Mario D'Onghia, Matteo Salvadore, Benedetto Maria Nespoli, Michele Carminati, Mario Polino, and Stefano Zanero. 2022. Apícula: Static Detection of API Calls in Generic Streams of Bytes. *Computers & Security* (2022), 102775.

[20] Mohammadreza Ebrahimi, Ning Zhang, James Hu, Muhammad Taqi Raza, and Hsinchun Chen. 2020. Binary black-box evasion attacks against deep learning-based static malware detectors with adversarial byte-level language model. *arXiv preprint arXiv:2012.07994* (2020).

[21] Alessandro Erba, Riccardo Taormina, Stefano Galelli, Marcello Pogliani, Michele Carminati, Stefano Zanero, and Nils Ole Tippenhauer. 2020. Constrained Concealment Attacks against Reconstruction-based Anomaly Detectors in Industrial Control Systems. In *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*. ACM, 480–495. https://doi.org/10.1145/3427228.3427660

[22] Mojtaba Eskandari and Sattar Hashemi. 2011. Metamorphic malware detection using control flow graph mining. *Int. J. Comput. Sci. Netw. Secur* 11, 12 (2011), 1–6.

[23] Zhang Fuyong and Zhao Tiezhu. 2017. Malware Detection and Classification Based on N-Grams Attribute Similarity. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Vol. 1. 793–796. https://doi.org/10.1109/CSE-EUC.2017.157

[24] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A Systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security* 113 (2022), 102550.

[25] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. 2019. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 113–125.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[27] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[28] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[29] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* 7 (2019), 47230–47244.

[30] Sanghyun Hong, Nicholas Carlini, and Alexey Kurakin. 2021. Handcrafted Backdoors in Deep Neural Networks. *CoRR* abs/2106.04690 (2021). arXiv:2106.04690 https://arxiv.org/abs/2106.04690

[31] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. 43–58.

[32] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*. 625–642.

[33] Kesav Kancherla and Srinivas Mukkamala. 2013. Image visualization based malware detection. In *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. 40–44. https://doi.org/10.1109/CICYBS.2013.6597204

[34] Jeffrey O. Kephart, Gregory B. Sorkin, William C. Arnold, David M. Chess, Gerald J. Tesauro, and Steve R. White. 1995. Biologically Inspired Defenses against Computer Viruses. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1* (Montreal, Quebec, Canada) *(IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 985–996.

[35] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*. IEEE, 533–537.

[36] Jeremy Z. Kolter and Marcus A. Maloof. 2004. Learning to Detect Malicious Executables in the Wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Seattle, WA, USA) *(KDD '04)*. Association for Computing Machinery, New York, NY, USA, 470–478. https://doi.org/10.1145/1014052.1014105

[37] Marek Krcál, Ondř Švec, Martin Bálek, and Otakar Jasek. 2018. Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only. In *ICLR*.

[38] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. 2018. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. Chapman and Hall/CRC, 99–112.

[39] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. 2020. Composite backdoor attack for deep neural network by mixing existing benign features. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 113–131.

[40] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 273–294.

[41] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2017. Trojaning attack on neural networks. (2017).

[42] Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K Reiter, and Saurabh Shintre. 2021. Malware Makeover: breaking ML-based static analysis by modifying executable bytes. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 744–758.

[43] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[44] Tajuddin Manhar Mohammed, Lakshmanan Nataraj, Satish Chikkagoudar, Shivkumar Chandrasekaran, and BS Manjunath. 2021. Malware detection using frequency domain-based image visualization and deep learning. *arXiv preprint arXiv:2101.10578* (2021).

[45] Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, and Yuval Elovici. 2008. Unknown malcode detection via text categorization and the imbalance problem. In *2008 IEEE International Conference on Intelligence and Security Informatics*. 156–161. https://doi.org/10.1109/ISI.2008.4565046

[46] United States Attorney's Office. 2020. Former Uber Executive Sentenced To 18 Months In Jail For Trade Secret Theft From Google. https://www.justice.gov/usao-ndca/pr/former-uber-executive-sentenced-18-months-jail-trade-secret-theft-google. [Online; accessed Dec 2nd, 2022].

[47] United States Attorney's Office. 2021. Fraudster Sentenced to Prison for Long Running Phone Unlocking Scheme that Defrauded AT&T. https://www.justice.gov/opa/pr/fraudster-sentenced-prison-long-running-phone-unlocking-scheme-defrauded-att. [Online; accessed Dec 2nd, 2022].

[48] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*. 729–746.

[49] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1332–1349. https://doi.org/10.1109/SP40000.2020.00073

[50] Xiangyu Qi, Jifeng Zhu, Chulin Xie, and Yong Yang. 2021. Subnet Replacement: Deployment-stage backdoor attack against deep neural networks in gray-box setting. *CoRR* abs/2107.07240 (2021). arXiv:2107.07240 https://arxiv.org/abs/2107.07240

[51] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

[52] Edward Raff, William Fleshman, Richard Zak, Hyrum S Anderson, Bobby Filar, and Mark McLean. 2021. Classifying sequences of extreme length with constant memory applied to malware detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 9386–9394.

[53] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. 2018. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques* 14, 1 (2018), 1–20.

[54] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. 2022. Dynamic backdoor attacks against machine learning models. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 703–718.

[55] Ashkan Sami, Babak Yadegari, Hossein Rahimi, Naser Peiravian, Sattar Hashemi, and Ali Hamze. 2010. Malware detection based on mining API calls. In *Proceedings of the 2010 ACM symposium on applied computing*. 1020–1025.

[56] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K Penya, Borja Sanz, Carlos Laorden, and Pablo G Bringas. 2010. Idea: Opcode-sequence-based malware detection. In *International Symposium on Engineering Secure Software and Systems*. Springer, 35–43.

[57] Giorgio Severi, Jim Meyer, Scott Coull, and Alina Oprea. 2021. {Explanation-Guided} Backdoor Poisoning Attacks Against Malware Classifiers. In *30th USENIX Security Symposium (USENIX Security 21)*. 1487–1504.

[58] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev, and Yuval Elovici. 2012. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics* 1, 1 (2012), 1–22.

[59] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. 2016. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*. https://doi.org/10.1145/2976749.2978392

[60] Pramila P Shinde and Seema Shah. 2018. A review of machine learning and deep learning applications. In *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 1–6.

[61] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF Detection Using Metadata and Structural Features. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, Florida, USA) *(ACSAC '12)*. Association for Computing Machinery, New York, NY, USA, 239–248. https://doi.org/10.1145/2420950.2420987

[62] P Vinod, R Jaipur, V Laxmi, and M Gaur. 2009. Survey on malware detection methods. In *Proceedings of the 3rd Hackers' Workshop on computer and internet security (IITKHACK'09)*. 74–79.

[63] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *2019 IEEE Symposium on Security and Privacy (SP)*. 707–723. https://doi.org/10.1109/SP.2019.00031

[64] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A Gunter, and Bo Li. 2021. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 103–120.

[65] Limin Yang, Zhi Chen, Jacopo Cortellazzi, Feargus Pendlebury, Kevin Tu, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. 2022. Jigsaw Puzzle: Selective Backdoor Attack to Subvert Malware Classifiers. *arXiv preprint arXiv:2202.05470* (2022).

[66] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. 2019. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2041–2055.

[67] Richard Zak, Edward Raff, and Charles Nicholas. 2017. What can N-grams learn for malware detection?. In *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 109–118.