DRSM: De-Randomized Smoothing on Malware Classifier Providing Certified Robustness

Shoumik Saha, Wenxiao Wang, Yigitcan Kaya, Soheil Feizi & Tudor Dumitras
{smksaha, wwx, cankaya, sfeizi, tudor}@umd.edu
Department of Computer Science
University of Maryland - College Park

Abstract

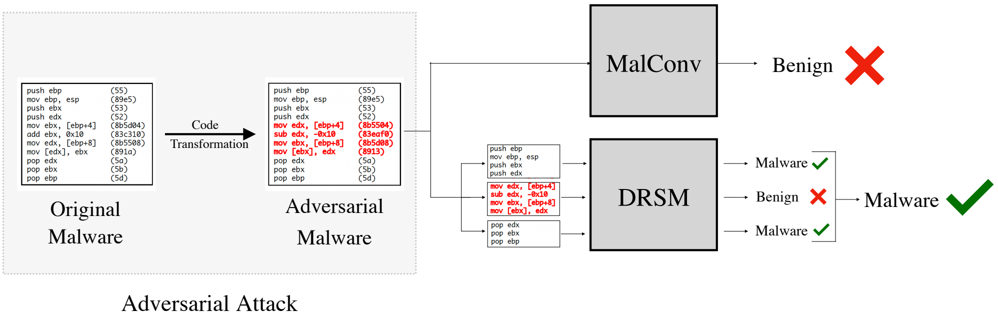
Machine Learning (ML) models have been utilized for malware detection for over two decades. Consequently, this ignited an ongoing arms race between malware authors and antivirus systems, compelling researchers to propose defenses for malware-detection models against evasion attacks. However, most if not all existing defenses against evasion attacks suffer from sizable performance degradation and/or can defend against only specific attacks, which makes them less practical in real-world settings. In this work, we develop a certified defense, DRSM (De-Randomized Smoothed MalConv), by redesigning the *de-randomized smoothing* technique for the domain of malware detection. Specifically, we propose a *window ablation* scheme to provably limit the impact of adversarial bytes while maximally preserving local structures of the executables. After showing how DRSM is theoretically robust against attacks with contiguous adversarial bytes, we verify its performance and certified robustness experimentally, where we observe only marginal accuracy drops as the cost of robustness. To our knowledge, we are the first to offer certified robustness in the realm of static detection of malware executables. More surprisingly, through evaluating DRSM against 9 empirical attacks of different types, we observe that the proposed defense is empirically robust to some extent against a diverse set of attacks, some of which even fall out of the scope of its original threat model. In addition, we collected $15.5K$ recent benign raw executables from diverse sources, which will be made public as a dataset called PACE (Publicly Accessible Collection(s) of Executables) to alleviate the scarcity of publicly available benign datasets for studying malware detection and provide future research with more representative data of the time. Our code and dataset are available at - https://github.com/ShoumikSaha/DRSM

1 Introduction

Machine learning (ML) has started to see more and more adoption in static malware detection, as it also has in many other mission-critical applications. Traditionally, ML models that use static features (Anderson & Roth, 2018) require a feature engineering step due to the large size and complex nature of programs. More recently, however, researchers have proposed models like MalConv (Raff et al., 2018) that can consume whole program simply as raw binary executable to eliminate this step. As expected, there has been a rise in studies showing the adversarial vulnerability of these models in the last few years (Kreuk et al., 2018; Lucas et al., 2021), resulting in an ongoing arms race.

Currently, existing defenses, such as non-negative or monotonic classifier (Fleshman et al., 2018; Íncer Romeo et al., 2018) and adversarial training (Lucas et al., 2023), not only introduce sizable drops in standard accuracy but also provide robustness only to specific attacks while still being vulnerable to the rest.

While certified robustness has been studied by many (Cohen et al., 2019; Lecuyer et al., 2019; Salman et al., 2019; Levine & Feizi, 2020a;b), it remains under-explored in the context of malware detection. To fill this gap, we redesign the *de-randomized smoothing* scheme, a certified defense originally developed for images (Levine & Feizi, 2020a), to detect malware from raw bytes. With MalConv (Raff et al., 2018) as the base classifier, we use DRSM (De-Randomized Smoothed Mal-



```
push ebp            (55)
mov ebp, esp        (89e5)
push ebx            (53)
push edx            (52)
mov ebx, [ebp+4]    (8b5d04)
add ebx, 0x10       (83c310)
mov edx, [ebp+8]    (8b5508)
mov [edx], ebx      (891a)
pop edx             (5a)
pop ebx             (5b)
pop ebp             (5d)
```

```
push ebp            (55)
mov ebp, esp        (89e5)
push ebx            (53)
push edx            (52)
mov edx, [ebp+4]    (8b5504)
sub edx, -0x10      (83eaf0)
mov ebx, [ebp+8]    (8b5d08)
mov [ebx], edx      (8913)
pop edx             (5a)
pop ebx             (5b)
pop ebp             (5d)
```

```
push ebp
mov ebp, esp
push ebx
push edx
```

```
mov edx, [ebp+4]
sub edx, -0x10
mov ebx, [ebp+8]
mov [ebx], edx
```

```
pop edx
pop ebx
pop ebp
```

Figure 1: Overview of a prototypical adversarial attack on MalConv and DRSM model. MalConv misclassifies the adversarial malware file as 'benign'. Our DSRM creates ablated sequences of the file and makes predictions on each, among which, the majority (*winning*) class is still 'malware'.

Conv) to denote the resulting defense. To our knowledge, DRSM is the first defense offering *certified robustness* for malware executable detection.

It is challenging for malware domain to utilize de-randomized smoothing scheme due to the inherent difference between image and raw byte file structure. As a solution, we propose a *window ablation* scheme that generates a set of ablated sequences by dividing the input sequence into non-overlapping windows. For each of these ablated sequences, we train a base classifier keeping the ground truth from original input. At inference, DRSM take the majority of predictions from these base classifiers as its final prediction. Figure 1 shows a simplified toy example: An adversarial attack may successfully evaded MalConv model with the presented small changes to the raw executables, but it would still be detected by DRSM if the perturbation could not manipulate sufficient votes.

We find that our DRSM (98.18%) can achieve comparable standard accuracy to MalConv (98.61%), and outperforms a prior defense MalConv(NonNeg) (88.36%) by a large margin. Besides our theoretical formulation for DRSM's certified robustness, we show that it can provide up to 53.97% certified accuracy depending on the attacker's capability. We discuss the performance-robustness trade-offs, and its adaptability upon demand. Moreover, we evaluate the empirical robustness of our DRSM model against 9 different attacks in both white and black box settings, including attacks outside of the intended threat model of De-Randomized Smoothing. Depending on the attack, even the least robust DRSM model can provide $87.9\%{\sim}26.5\%$ better robustness than MalConv.

A practical difficulty in malware research is collecting benign raw executables, due to copyrights and legal restrictions Anderson & Roth (2018). Throughout this work, we collect $15.5K$ fairly recent and diverse benign executables from different sources, which can be a better representative of the real world. These will be made public as a new dataset, namely PACE (Publicly Accessible Collection(s) of Executables), to alleviate the accessibility issue of benign executables and facilitate future research.

Our major contributions include: (1) A new defense, DRSM (De-Randomized Smoothed MalConv), that pioneers certified robustness in the executable malware domain (Section 5).
(2) A thorough evaluation of DRSM regarding its performance and certified robustness, which suggests DRSM offers certified robustness with only mild performance degradation (Section 6).
(3) A thorough evaluation of DRSM regarding its empirical robustness against 9 empirical attacks covering different settings and types, which suggests DRSM is empirically robust to some extent against diverse attacks. (Section 7).
(4) A collection of $15.5K$ benign binary executables from different sources, which will be made public as a part of our new dataset PACE. (Section 4).

2 Related Work

ML in Static Malware Detection. There have been several studies of how malware executables can be classified using ML. As early as 2001 Schultz et al. (2001) proposed a data mining technique for malware detection using three different types of static features. Pioneered by Nataraj et al. (2011), CNN-based techniques for malware detection became popular among security researchers, e.g., Kalash et al. (2018); Yan et al. (2018). Eventually, Raff et al. (2018) proposed a static classifier, named MalConv, taking raw byte sequences to detect malware using a convolutional neural network. We will use it as the base classifier in this work. It is still considered a state-of-the-art for detection from raw byte inputs, and its popularity led to a follow up model MalConv 2 (Raff et al., 2021).

Adversarial Attacks and Defenses in Malware Detection. Along with the detection research, there has been plenty of research on adversarial attacks on these models. These attacks fall into different categories. For example, attacks proposed by Kolosnjaji et al. (2018); Kreuk et al. (2018); Suciu et al. (2019) appended and/or injected adversarial bytes in the malware computed by gradient. Demetrio et al. (2019; 2021b); Nisi et al. (2021) motivated attacks that modify or extend DOS and Header fields. Demetrio et al. (2021a) extracted payloads from benign files to be appended and injected into malware files. Recent work by Lucas et al. (2021) used two types of code transformation to generate adversarial samples. For defenses, Fleshman et al. (2018) proposed a defense, MalConv (NonNeg), by constraining weights in the last layer of MalConv to be non-negative. However, this model achieves low accuracy of 88.36%, and has been shown to be as vulnerable as MalConv in some cases (Wang et al., 2023; 2022; Ceschin et al., 2019). Another defense strategy, adversarial training cannot guarantee defense against attacks other than the one used during training, which limits its usage: Lucas et al. (2023) showed training it on Kreuk-0.01 degraded the true positive rates to $84.4\% \sim 90.1\%$. Notably, where variants of randomized smoothing schemes have been proposed for vision domains (Cohen et al., 2019; Lecuyer et al., 2019; Salman et al., 2019; Levine & Feizi, 2020a;b) (more details in the Appendix A.2), they remain under-explored in the context of malware detection. Although there is a concurrent work (Huang et al., 2023) that proposes certified robustness in malware domain, they differ from us in terms of the employed smoothing scheme and threat model.

Limited Accessibility to Benign Executables. Though there has been a large amount of work on malware detection, most of the work was done using private or enterprise dataset with restrictive access. Prior works (Anderson & Roth, 2018; Yang et al., 2021a; Downing et al., 2021) explain the copyright issue and only published the feature vector of benign files (see Table 5). This impose many constraints to the advancement of malware detection techniques, especially to have a complete model that requires raw executables as inputs.

3 Background and Notations

We denote the set of all bytes of a file as $X \in \{0, 1, 2, ..., N-1\}$, where $N = 256$. A binary file is a sequence of k bytes $x = (x_1, x_2, x_3, ...x_k)$, where $x_i \in X$ for all $1 \leq i \leq k$. Note that the length k varies for different files, thus k is not fixed. However, the input vector fed into the network has to be of a fixed dimension. So, the common approach is to – pad zeros at the end of x if $k < D$, or extract the first D bytes from x, to fix the input length to D.

3.1 Base Classifier

In this work, we will be using the state-of-the-art static malware detection model to this date, named MalConv (Raff et al., 2018), as our base classifier. While there are other models like Ember, GBDT (Anderson & Roth, 2018) for malware detection, note that – these models work on a specified feature format that needs an extra feature extraction step, whereas our model can directly take the raw binary executables. Let us represent the MalConv model (see Figure 7) as $F_\theta : X \to [0, 1]$ with a set of parameters θ that it learns through training. If the output of $F_\theta(X)$ is greater than 0.5 then the prediction is considered as 1 or malicious, and vice versa. We set the input length as 2MB following the original paper.

MalConv takes in each byte x_i from file X and then passes it to an embedding layer with an embedding matrix $Z \in \mathbb{R}^{D \times 8}$, which generates an embedding vector $z_i = \phi(x_i)$ of 8 elements. This vector is then passed through two convolution layers, using ReLU and sigmoid activation functions. These activation outputs are combined through a gating which performs an element-wise multiplication to mitigate the vanishing gradient problem. The output is then fed into a temporal max pooling layer, followed by a fully connected layer. Finally, a softmax layer calculates the probability.

3.2 Threat Model

We assume that the attacker has the full knowledge of the base-classifier, including architectures and model parameters. This is typically referred to as the white-box setting. The white-box setting considers potentially strong attackers, which is desired when assessing defenses.

In the primary threat model that we consider when developing our defense, the attacker can modify any existing bytes or add (append or insert) extra bytes bounded in a contiguous portion of the input sample in test time to evade the model. So, the goal of the attacker is to generate an aforementioned perturbation δ that can be applied on malware x to generate an adversarial malware $x^{'}$, for which $F_\theta(x^{'}) < 0.5$, i.e., the classifier predicts it as a benign file. Here, the attacker knows the classifier model F and its parameters θ, and can modify the original malware file x. However, finding the perturbation δ in a binary file is more challenging than vision due to its inherited file structures. For any arbitrary change in a malware file, the file can lose its semantics, i.e. malicious functionality, in the worst case, the file can get corrupted.

Even after such challenges in binary modification, prior attacks have been successful by adding contiguous adversarial bytes at the end (Kreuk et al., 2018) or other locations (Suciu et al., 2019; Demetrio et al., 2021a;b), or modifying bytes at specific locations(Demetrio et al., 2019; Nisi et al., 2021), to evade a model. Though the attacks that are bounded in one contiguous portion falls within our primary threat model, for empirical robustness evaluation, we include attacks that can have impacts at multiple different parts in the file. In addition, we also consider recent, more sophisticated attacks (Lucas et al., 2021; 2023) where the attacker has the power to disassemble malware and apply different code transformations at any place in the file. For coherence, we defer the details about these attacks to Section 7, where we evaluate the empirical robustness of our defenses against them.

4 A New Publicly Available Dataset—PACE

Like other domains, malware detection suffers from concept drift too. Previously, Yang et al. (2021b); Jordaney et al. (2017); Barbero et al. (2022) demonstrated how concept drift can have a disastrous impact on ML-based malware detection. Therefore, we used 3 datasets from different times in this work (Table 1). However, in the malware domain, having a large dataset to train a machine learning (ML) model may not be enough as maintaining diversity and recency is also crucial (Cao et al., 2020; Downing et al., 2021). We found that models trained without diverse benign samples can have a very high false positive rate (see details in Appendix A.1.3).

Despite the importance of diverse benign samples, unfortunately, most prior works (Anderson & Roth (2018); Downing et al. (2021)) could not publish raw executables of benign files due to copyright and legal restrictions. For this work, we crawled popular free websites, e.g., SourceForge, CNET, Net, Softonic, etc., to collect a diverse benign dataset of size 15.5K (Table 2), naming **PACE** (Publicly Accessible Collection(s) of Executables). We collected the malware from VirusShare at the same time (August 2022) as benign files. Following the common practice and guidelines, we are publishing the URLs along with the MD5 hash for each raw benign file in our dataset (see Appendix A.1 for more details). We hope this will help researchers to recreate the dataset easily and experiment with a better representative of real-world settings in the future.[1]

Table 1: Datasets used in this work with collection time, size, and public availability of raw executables

Dataset Name	Collection Time	Number of Binaries			Public Availability
		Malware	Benign	Total	
Ember	2017	400K	400K	800K	✗
VTFeed	2020	139K	139K	278K	✗
PACE (Our)	2022	15.5K	15.5K	31K	✓
Total		554.5K	554.5K	1.1M	

Table 2: PACE (Benign) Dataset

Source	Number of Binaries
SourceForge	7,865
CNET	3,661
Net	2,534
Softonic	1,152
DikeDataset	1,082
Netwindows	185
Manually Obtained from Windows OS	89
Total	15,568

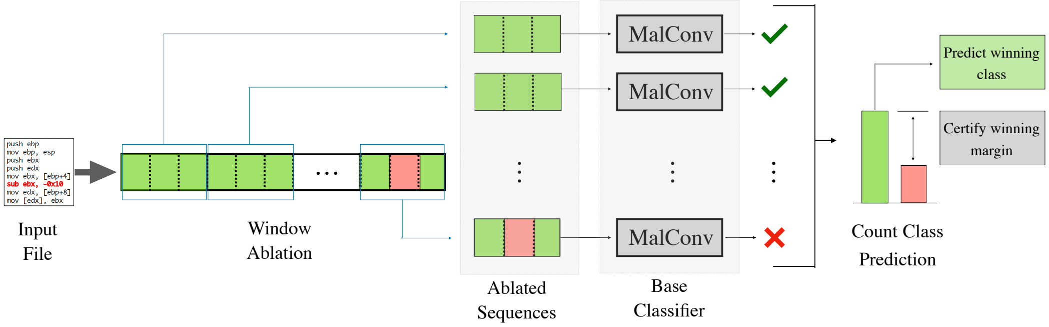
We used a MalConv model pre-trained on Ember (Anderson & Roth, 2018) dataset provided by the Endgame Inc. Then we used this model to re-train the MalConv, MalConv (NonNeg), and our DRSM models on both VTFeed and PACE (our) dataset.[2] We split our dataset into 70:15:15 ratios for train, validation, and test sets, respectively. During evaluation, we made sure that test samples came from the latest dataset (PACE) only. For model implementation details, see Appendix A.3.

[1] PACE malware samples will also be provided upon request.

[2] The authors of (Lucas et al., 2021) assisted in training models on VTFeed, which we could not have done by ourselves since VTFeed is not publicly accessible

5 DRSM: De-Randomized Smoothing on Malware Classifier

Since the malware detection problem cannot be directly mapped to typical vision problems, we had to redesign the 'de-randomized smoothing' scheme to make it compatible. Unlike images, our input samples are one-dimensional sequences of bytes, which makes the common vision-oriented ablation techniques, e.g., adding noise, masking pixels, block ablations, etc., infeasible. Additionally, even a random byte change in a file may cause a behavior change or prevent the sample from executing.



```
push ebp
mov ebp, esp
push ebx
push edx
mov ebx, [ebp+4]
sub ebx, -0x10
mov edx, [ebp+8]
mov [edx], ebx
```

Figure 2: DRSM (De-Randomized Smoothed MalConv) model framework. Here, the red small block in 'Window Ablation' represents the perturbation by attacker, and hence, the base classifier gives a wrong prediction for that (shown with red cross).

So, we introduce the *'window ablation'* strategy which involves segmenting the input sample into multiple contiguous sequences of equal size. If the input length of the base classifier is L, and the size of the ablated window is w, then there will be $\lceil \frac{L}{w} \rceil$ ablated sequences of length w resulting in the ablated sequence set $S(x)$. So, even if an attacker generates a byte perturbation of size p, it can modify at most $\Delta = \lceil \frac{p}{w} \rceil + 1$ ablated sequences (+1 when a perturbation overlaps 2 windows). Since a perturbation can only influence a limited number of ablated sequences, it cannot directly change the decision of the smoothed-classifier model – which was our prior motivation to integrate this technique. A visual representation of our strategy is provided in Figure 2.

The goal of the defender is to – using F_θ as the base classifier, find a *de-randomized smoothed model* G_θ that can detect any adversarial malware $x^{'}$ generated using a perturbation δ. G_θ takes in each sequence s from the ablated sequence set $S(x)$, and returns the most frequent predicted class. Specifically, for an input file x, ablated sequence set $S(x)$, and base classifier F_θ, the *de-randomized smoothed model* G_θ can be defined as:

$$G_\theta(x) = \arg\max_c n_c(x)$$

where,

$$n_c(x) = \sum_{x^{'} \in S(x)} I\{F_\theta(x^{'}) = c\}$$

denotes the number of ablated sequences that were predicted as class c. The percentage of files that are correctly classified by the *de-randomized smoothed model* G_θ is the **'standard accuracy'**.

We say the classifier G_θ *certifiably robust* on an ablated sequence set if the number of predictions for the correct class exceeds the incorrect one by a 'large margin' (dictated by byte size of perturbation). This 'large margin' puts a lower bound on attacker's success in altering predictions of the classifier G_θ since a perturbation δ of size p can, at most, impact $\Delta = \lceil \frac{p}{w} \rceil + 1$ ablated sequences.

Mathematically, the *de-randomized smoothed model* G_θ is 'certifiably robust' on input x for predicting class c if:

$$n_c(x) > max_{c \neq c'} n_{c'}(x) + 2\Delta$$

Since our problem is a binary classification problem, this can be rewritten as:

$$\begin{aligned} n_m(x) &> n_b(x) + 2\Delta \quad &; \text{if } true{-}label(x) = malware \\ n_b(x) &> n_m(x) + 2\Delta \quad &; \text{if } true{-}label(x) = benign \end{aligned} \tag{1}$$

where, $n_m(x)$ and $n_b(x)$ are the number of ablated sequences predicted as malware and benign by the *de-randomized smoothed model* G_θ, respectively. The percentage of file that holds the inequality 1 for G_θ is the **'certified accuracy'**.

For simplicity, we will use DRSM-n to denote DRSM with the number of ablated sequences $|S(x)| =$ n, e.g. DRSM-4 means 4 ablated sequences on input x will be generated for DRSM.

6 Certified Robustness Evaluation

6.1 Standard Accuracy

For evaluation, we compare our DRSM models with MalConv(Raff et al., 2018) which is still one of the state-of-the-art models for static malware detection. Moreover, we consider the non-negative weight constraint variant of MalConv which was proposed as a defense against adversarial attack in prior work (Fleshman et al., 2018). We train and evaluate these models on the same train and test set (Section 4).

Table 3: Standard and Certified Accuracy of Models. MalConv and MalConv(NonNeg) cannot provide certified accuracy

Model	Standard Accuracy (in %) ↑			Certified Accuracy [$\Delta = 2$] (in %) ↑		
	Train-set	Validation-set	Test-set	Train-set	Validation-set	Test-set
MalConv	99.73	98.87	98.61	–	–	–
MalConv(NonNeg)	88.56	87.56	88.36	–	–	–
DRSM-4	99.49	98.12	98.18	14.74	7.84	12.2
DRSM-8	99.67	97.88	97.79	52.74	43.9	40.85
DRSM-12	96.07	95.58	95.88	45.77	44.43	46.21
DRSM-16	94.29	93.00	93.3	59.1	50.52	49.17
DRSM-20	91.17	91.05	91.15	51.64	51.92	52.68
DRSM-24	90.22	89.80	90.24	54.19	54.88	53.97

For DRSM-n, we choose $n = \{4, 8, 12, 16, 20, 24\}$ for our experiments and show the standard accuracy on the left side of the Table 3. Recall that – for DRSM-n, a file is correctly classified if the winning class from majority voting matches the true label for that file (Section 5). For ties, we consider 'malware' as the winning class. From the Table 3, we can see that – DRSM-4 (98.18%) and DRSM-8 (97.79%) can achieve comparable accuracy to the MalConv model (98.61%). However, increasing the n has a negative impact on the standard accuracy. For example, DSRM-20 and DSRM-24 achieve 91.15% and 90.24% standard accuracy, respectively. We investigate and find that – with more ablations (smaller window), the probability of one window containing enough malicious features to make a stable prediction becomes less. On the other hand, the MalConv (NonNeg) model has a lower accuracy, which is consistent with the results by Fleshman et al. (2018).

6.2 Certified Accuracy

Besides standard accuracy, we also evaluate the certified accuracy for DRSM-n models. Recall that – 'certified accuracy' is the percentage of files for which the inequality 1 holds true for DRSM-n models. In short, it denotes the lower bound of model performance even when the attacker can perturb bytes in Δ number of ablated windows and alter predictions for all of them. So, we run experiments on DRSM-n models by varying the Δ in equation 1, i.e., perturbation budget for the attacker. To maintain consistency between standard and certified accuracy, we take 'malware' as the winning class for ties by tweaking the first inequality in 1 to $n_m(x) \geq n_b(x) + 2\Delta$.

Notably, $\Delta \in \{2, 3, ..., \frac{n}{2}\}$. The range starts from 2, because any perturbation smaller than the window size can overlap with at most 2 ablated sequences, and goes up to $\frac{n}{2}$, because the inequality 1 will not hold beyond this point. The right side of Table 3 shows the certified accuracy of DRSM-n models for $\Delta = 2$. In Figure 3, we show the result of certified accuracy on the test set for each perturbation budget for the attacker (x-axis) (Figure 10 shows in terms of Δ). See Tables 7 and 6 in A.4 for more details. We emphasize that even with small $\Delta = 2(= \lceil \frac{255K}{256K} \rceil + 1)$, an attacker can perturb up to $255K$ bytes for DRSM-8, and yet the model maintains 40.85% certified accuracy.

By analyzing Table 3, we can see that n has a positive and negative correlation with certified and standard accuracy, respectively. While DRSM-24 provides the highest certified accuracy (53.97%),

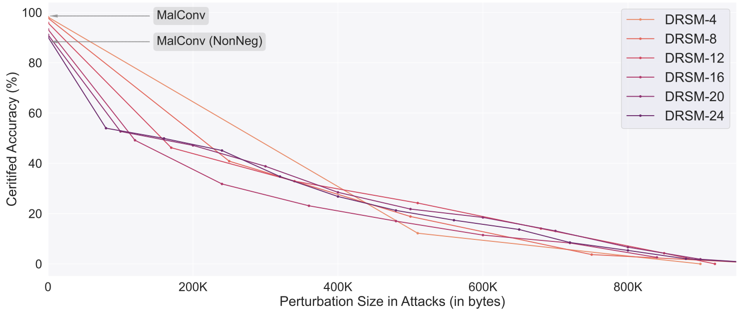


Figure 3: Certified Accuracy (%) of DRSM-n models for different perturbation budgets (Test-set). While MalConv and MalConv (NonNeg) are not certifiably robust, their standard accuracy is highlighted for references.

it has the lowest standard accuracy (90.24%) among all DRSM-n models. In contrast, DRSM-4 provides the highest standard accuracy (98.18%) with 12.2% certified accuracy. This observation may suggest a performance trade-off. It is worth highlighting that models like DRSM-8 and DRSM-16 strike a balance, delivering robust certified performance alongside commendable standard accuracy, while prior defense MalConv (NonNeg) achieves lower standard accuracy 88.36%. We also want to emphasize that – perturbing 200KB in a 2MB file (= 10%) is considered as a sizeable modification to a malware file, and yet our DRSM-n models can provide 37%∼64% certified accuracy for such perturbation (from Figure 3). Remember that – this accuracy reports the theoretical lower bound and in practice, our DRSM-n models provide even higher robustness (shown in Section 7).

7 Empirical Robustness Evaluation

Beyond theoretical robustness, we also evaluate the empirical robustness of our DRSM-n models. Recall from Section 3.2 that – in our threat model (any de-randomized smoothing scheme), attackers can add, or modify bytes in a contiguous portion of a malware file, to get it misclassified as a benign one. However, in real-life settings, attackers can be more capable and can deploy complex attacks where they can find multiple contiguous blocks to perturb.

In this work, we consider 9 different attacks in both white and black box settings and categorize them into 3 types based on their alignment with our threat model. **Fully Aligned:** if an attack perturbs bytes in one contiguous block; **Partially Aligned:** if an attack perturbs bytes in multiple different contiguous blocks; **Not Aligned:** if an attack applies code transformation and changes bytes all over the file (not limited to any contiguous block). Table 4 shows the list of attacks that have been considered in this work along with their type, settings, and short description. For more details about individual attacks and their implementation, see Appendix A.5.

To evaluate the attacks against MalConv, MalConv (NonNeg) and DRSM-n models, we randomly sampled 200 malware from the test-set of our dataset that are correctly classified by the model before attack. Let us call this subset of malware the 'attack set'. We call an attack 'successful' if the attack can generate a functional adversarial malware that can change the model's prediction from 'malware' to 'benign'. Even though the majority voting in DRSM-n is not differentiable, it can still be attacked by targeting its base classifiers. Correspondingly, whenever necessary, we generate adversarial malware from the 'attack set' by differentiating through the base classifier. Attack settings (white/black-box) are determined based on the attacker's knowledge about the base classifier.

Figure 4 shows the attack success rate (ASR) for different attacks in the white-box setting. We find that – most attacks have less ASR on DRSM-n models than MalConv by a large margin. For example, FGSM append attack has 82.50% ASR on MalConv whereas 10.0% and 7.0% on DRSM-4 and DRSM-8, respectively. Moreover, for $n \geq 16$ in DRSM-n models, the ASR for all white-box attacks is (1%∼5%). We got the highest ASRs on MalConv model for DOS Extension (98.00%) and Disp (89.50%) attack, while the ASRs on DSRM-n models were in range of (1%∼72%) and (1%∼42%), respectively.

Table 4: Attacks evaluated. ○ - Fully Aligned, ◐ - Partially Aligned and ● - Not Aligned describe the alignment of the attacks to our primary threat model (see Section 3.2).

Attack	Threat Model	Settings		Short Description
		White-box	Black-box	
FGSM Append (Kreuk et al., 2018)	○	✔		Appends random bytes at the end of the file generated by FGSM
Slack Append (Suciu et al., 2019)	◐	✔		Injects non-functional bytes in slack regions generated by FGSM
DOS Extension (Demetrio et al., 2021b)	◐	✔		Extends the DOS header and injects adversarial noise
DOS Modification (Partial) (Demetrio et al., 2019)	○	✔	✔	Puts adversarial noise in between of `MZ` and offset `0x3c` in the DOS header
DOS Modification (Full) (Demetrio et al., 2021b)	○	✔	✔	Modifies every byte in the DOS header without corrupting the file
Header Field Modification (Nisi et al., 2021)	◐	✔	✔	Modifies fields in PE header
Disp (Lucas et al., 2021)	●	✔		Displaces code instructions using `jmp` and semantic `nop`
IPR (Lucas et al., 2021)	●	✔		Replaces instructions in multiple ways (equiv. replace, register reassign, reorder, etc.) without altering functionalities
GAMMA (Demetrio et al., 2021a)	◐		✔	Extracts payloads from benign programs and injects them in malware

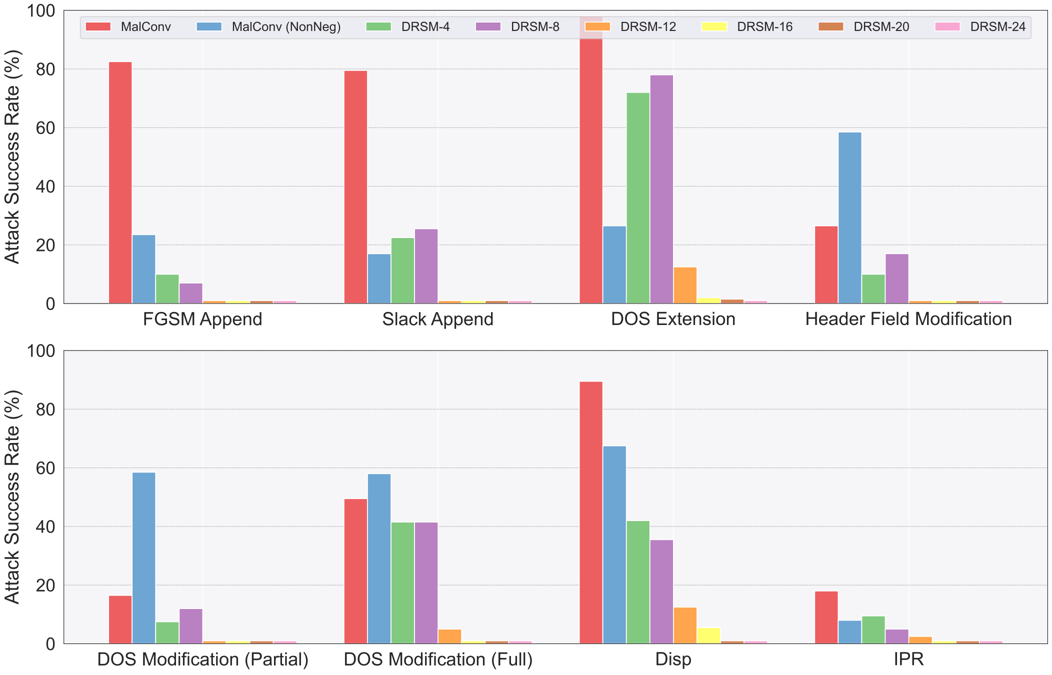


Figure 4: Attack Succes Rate (ASR) % for white-box attacks on all models

Though Disp and IPR attacks fall outside of our threat model, surprisingly, DRSM-n can still provide good robustness against them (Figure 4). Here is a potential explanation: transformed bytes by Disp and IPR at different places get divided into multiple ablated sequences and thus, they become less impactful in altering multiple predictions compared to one prediction. An interesting observation is that the attacks that modify the header fields have marginally higher ASR on DRSM-8 than on

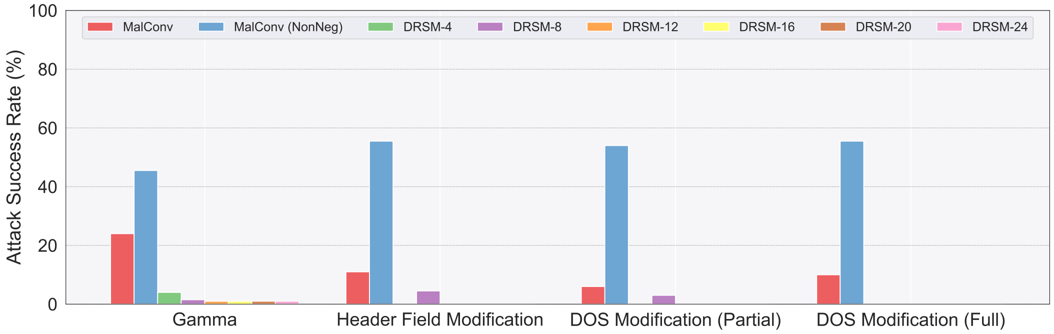


Figure 5: Attack Succes Rate (ASR) % for black-box attacks on different models

DRSM-4: Potentially, this is because for DRSM-8 the perturbed positions in header fields happen to cover more windows than other cases. Higher ASR of DOS extension attack is discussed in Appnedix A.4.1.

We also evaluated the models against black-box attacks using genetic optimizers. For example, GAMMA attack extracts payload from benign programs and injects them into malware by querying the model. From Figure 5, GAMMA has 24% ASR on MalConv whereas $(4{\sim}1)\%$ on DRSM-n models. While it is true that – these black-box attacks have less ASR on MalConv compared to the white-box ones, still DRSM-n models outperform. Interestingly, we found that MalConv(NonNeg) suffers in query-based black-box attacks, which is consistent with some recent works, e.g., Dropper attack by Wang et al. (2022), MPass, GAMMA attack by Wang et al. (2023), Goodware string append by Ceschin et al. (2019).

8 Limitations

Though the DRSM framework can strike a good balance between standard accuracy and robustness, it has some limitations too. Because of the majority voting, its final classification is naive by nature. In a malware file, some fields or sections might have higher importance than other sections. For example, the header fields in general have higher importance in classification than the data section (Demetrio et al., 2019). But due to the majority voting, both of them might get same importance in DRSM. Another limitation is – the 'window ablation' scheme solely depends on the size of the file; no section information is considered. But to solve this, one will have to disassemble the file first, which would add up some computational cost. Moreover, since the padding at the end of the file does not contain any useful information, and the model randomly classify such paddings in the most cases, DRSM framework does not take them into consideration.

In this work, we did not take an 'adaptive' attacker into the consideration, who can try to perturb every window in the file to evade DRSM. However, the attacker needs to know the size of ablations, and has to find perturb-able bytes in each window, which might be challenging but not infeasible. Since the 'de-randomized smoothing' is directly non-differentiable, and no state-of-the-art gradient based attack has been defined for it so far, we had to attack it through its base classifier in this work.

9 Conclusion

In this work, we tried to find a solution for the 'accuracy vs. robustness' double-edged sword in the malware field. We showed that certified defense is also possible in the executable malware domain, hoping that it will open up a new paradigm of research. Besides theory, we equally emphasized the empirical robustness of our proposed DRSM. We would like to conclude by highlighting some areas and future directions our work identifies. Firstly, there is room for improving the standard accuracy of DRSM by introducing an additional classification layer, albeit at the expense of challenging the fundamental non-differentiable nature of the smoothing scheme. Secondly, recent defenses from vision, besides de-randomized smoothing, hold promise for future exploration. Malware detection is inherently an arms race and we hope our work can facilitate future research in developing more practical defenses with our defense and dataset.

Acknowledgments

We are immensely grateful to Keane Lucas for providing the VTFeed dataset, and the private implementation of Disp, IPR attack (guided) to evaluate in this paper.

This project was supported in part by a grant from an NSF CAREER AWARD 1942230, ONR YIP award N00014-22-1-2271, ARO's Early Career Program Award 310902-00001, HR00112090132 (DARPA/RED), HR001119S0026 (DARPA/GARD), Army Grant No. W911NF2120076, the NSF award CCF2212458, NSF Award No. 2229885 (NSF Institute for Trustworthy AI in Law and Society, TRAILS), an Amazon Research Award, and an award from Capital One.

This research was also partially supported by an Amazon Research Award and by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112190093. Approved for public release; distribution is unlimited.

References

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

Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 805–823. IEEE, 2022.

Michael Cao, Sahar Badihi, Khaled Ahmed, Peiyu Xiong, and Julia Rubin. On benign features in malware detection. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1234–1238, 2020.

Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Luiz S Oliveira, and André Grégio. Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors. In *Proceedings of the 3rd Reversing and Offensive-oriented Trends Symposium*, pp. 1–9, 2019.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pp. 1310–1320. PMLR, 2019.

Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Explaining vulnerabilities of deep learning to adversarial malware binaries. *arXiv preprint arXiv:1901.03583*, 2019.

Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security*, 16:3469–3478, 2021a.

Luca Demetrio, Scott E Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security (TOPS)*, 24 (4):1–31, 2021b.

Evan Downing, Yisroel Mirsky, Kyuhong Park, and Wenke Lee. {DeepReflect}: Discovering malicious functionality through binary reconstruction. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 3469–3486, 2021.

William Fleshman, Edward Raff, Jared Sylvester, Steven Forsyth, and Mark McLean. Non-negative networks against adversarial attacks. *arXiv preprint arXiv:1806.06108*, 2018.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6572`.

Zhuoqun Huang, Neil G. Marchant, Keane Lucas, Lujo Bauer, Olga Ohrimenko, and Benjamin I. P. Rubinstein. Rs-del: Edit distance robustness certificates for sequence classifiers via randomized deletion, 2023.

Íñigo Íncer Romeo, Michael Theodorides, Sadia Afroz, and David Wagner. Adversarially robust malware detection using monotonic classification. In *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, IWSPA '18, pp. 54–63, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356343. doi: 10.1145/3180445.3180449. URL https://doi.org/10.1145/3180445.3180449.

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

Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil D. B. Bruce, Yang Wang, and Farkhund Iqbal. Malware classification with deep convolutional neural networks. In *9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018, Paris, France, February 26-28, 2018*, pp. 1–5. IEEE, 2018. doi: 10.1109/NTMS.2018.8328749. URL https://doi.org/10.1109/NTMS.2018.8328749.

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 *26th European Signal Processing Conference, EUSIPCO 2018, Roma, Italy, September 3-7, 2018*, pp. 533–537. IEEE, 2018. doi: 10.23919/EUSIPCO.2018.8553214. URL https://doi.org/10.23919/EUSIPCO.2018.8553214.

Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Deceiving end-to-end deep learning malware detectors using adversarial examples. *CoRR*, abs/1802.04528, 2018. URL http://arxiv.org/abs/1802.04528.

Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE symposium on security and privacy (SP)*, pp. 656–672. IEEE, 2019.

Alexander Levine and Soheil Feizi. (de)randomized smoothing for certifiable defense against patch attacks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020a. URL https://proceedings.neurips.cc/paper/2020/hash/47ce0875420b2dbacfc5535f94e68433-Abstract.html.

Alexander Levine and Soheil Feizi. Robustness certificates for sparse adversarial attacks by randomized ablation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 4585–4593. AAAI Press, 2020b. URL https://ojs.aaai.org/index.php/AAAI/article/view/5888.

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

Keane Lucas, Samruddhi Pai, Weiran Lin, Lujo Bauer, Michael K Reiter, and Mahmood Sharif. Adversarial training for {Raw-Binary} malware classifiers. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 1163–1180, 2023.

L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306799. doi: 10.1145/2016904.2016908. URL https://doi.org/10.1145/2016904.2016908.

Dario Nisi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Lost in the loader: The many faces of the windows pe file format. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 177–192, 2021.

Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. Malware detection by eating a whole EXE. In *The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, volume WS-18 of *AAAI Technical Report*, pp. 268–276. AAAI Press, 2018. URL https://aaai.org/ocs/index.php/WS/AAAIW18/paper/view/16422.

Edward Raff, William Fleshman, Richard Zak, Hyrum S. Anderson, Bobby Filar, and Mark McLean. Classifying sequences of extreme length with constant memory applied to malware detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(11):9386–9394, May 2021. doi: 10.1609/aaai.v35i11.17131. URL https://ojs.aaai.org/index.php/AAAI/article/view/17131.

Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. *Advances in Neural Information Processing Systems*, 32, 2019.

Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001*, pp. 38–49. IEEE Computer Society, 2001. doi: 10.1109/SECPRI.2001.924286. URL https://doi.org/10.1109/SECPRI.2001.924286.

Octavian Suciu, Scott E. Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops, SP Workshops 2019, San Francisco, CA, USA, May 19-23, 2019*, pp. 8–14. IEEE, 2019. doi: 10.1109/SPW.2019.00015. URL https://doi.org/10.1109/SPW.2019.00015.

Jialai Wang, Wenjie Qu, Yi Rong, Han Qiu, Qi Li, Zongpeng Li, and Chao Zhang. Mpass: Bypassing learning-based static malware detectors. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2023.

Shaohua Wang, Yong Fang, Yijia Xu, and Yaxian Wang. Black-box adversarial windows malware generation via united puppet-based dropper and genetic algorithm. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 653–662. IEEE, 2022.

Jinpei Yan, Yong Qi, and Qifan Rao. Detecting malware with an ensemble method based on deep neural network. *Secur. Commun. Networks*, 2018:7247095:1–7247095:16, 2018. doi: 10.1155/2018/7247095. URL https://doi.org/10.1155/2018/7247095.

Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. Bodmas: An open dataset for learning based temporal analysis of pe malware. In *2021 IEEE Security and Privacy Workshops (SPW)*, pp. 78–84. IEEE, 2021a.

Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2327–2344, 2021b.

A Appendix

A.1 Our Published Dataset: PACE

A.1.1 Dataset Details

Our diverse benign dataset contains benign raw executables from 7 different sources (Table 2). Figure 6 shows the cumulative distribution function (CDF) of the file sizes of our benign files.

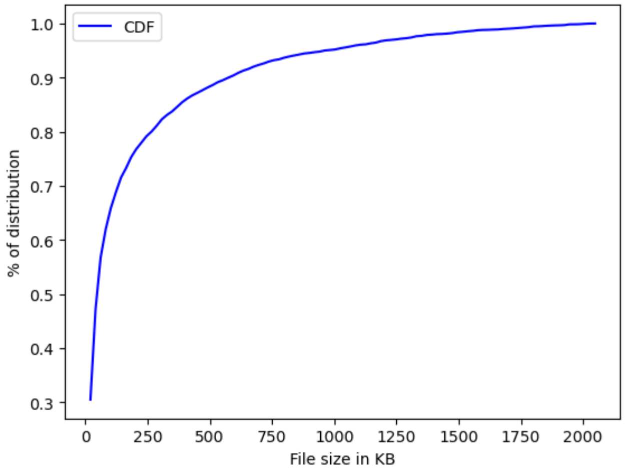


Figure 6: CDF plot of file sizes for our published benign dataset

Data Format. For each benign raw executables, we are going to publish the URL link to download it from with its MD5 hash for the response (See `'dataset'` folder in our supplementary material). For example, one line from our csv file is –

URL	MD5 hash
https://sourceforge.net/projects/pdfcreator/files/PDFCreator/PDFCreator%200.7/PDFCreator-Setup-0_7_0.exe/download	`afaf0caffeff781f6070f2a9aeb54bdf`

A.1.2 Other Datasets

Table 5: Other Benign Datasets and their public availability

Benign Datasets	Public Availability
PACE (Our)	Raw Binary Executable
Ember	only Feature Vector
VTFeed	✗
DeepReflect	only Feature Vector
BODMAS	only Feature Vector

A.1.3 Performance Degradation on Our PACE (Benign) Dataset

While there have been works about concept drift on malware (Yang et al., 2021b; Jordaney et al., 2017; Barbero et al., 2022) and they demonstrated how malwares evolved over the time, there have been very less work on concept drift of benign files. The probable reason can be the common belief that – benign files do not evolve or change, i.e., the distribution remains same for them. However, we evaluated a version of MalConv model on our PACE (benign) dataset that was trained on (Ember + VTFeed) dataset. Surprisingly, this MalConv version was misclassifying 12.22% benign files from PACE dataset while it was still having 98.91% test accuracy on VTFeed dataset. Recall that – our PACE dataset is the most recent one among these (2022). It is obvious that – these benign datasets have different distributions due to the variation in collection time. It might be the case that – with time, different companies release (or update) their softwares for newer version of Windows, and as a result, it causes a shift in benign file distribution too. So, we would suggest researchers to report their model performance on recent datasets in future, especially when it is about security-critical domain like malware detection.

A.2 De-Randomized Smoothing

De-Randomized Smoothing (Levine & Feizi, 2020a) is a certified defense against patch attacks in image classifications. It proposes two structured ablation methods to adapt Randomized Ablation (Levine & Feizi, 2020b), a certifiably robust image classficiation scheme against L_0 adversarial attacks, for the purpose of defending against patch attacks.

Given an input image, Randomized Ablation uses a base classifier to provide predictions on a large number of randomly ablated version of the input image, where each ablated version retains exactly k randomly selected pixels of the original input (k is a hyper-parameter of Randomized Ablation). This ensures that for any ρ pixels attacked, the probability that the attack affect a single ablated version will be

$$Pr[\text{at least one attacked pixel is retained}] = 1 - \frac{\binom{hw-\rho}{k}}{\binom{hw}{k}},$$

where h, w denotes the height and width of the input image. Thus if one aggregates the predictions made for ablated versions, only a bounded portion of them will be affected by the L_0 adversarial attacks and therefore certified robustness can be offered.

Building on Randomized Ablation, De-Randomized Smoothing proposes two structured ablation methods, block smoothing and band smoothing, to reduce the probability that at least one attacked pixel is retained in an ablated version for the context of patch attacks. For block smoothing, a single $s \times s$ square block of pixels will be retained per ablated version of the input image, thus for any adversarial patch of size $m \times m$, the probability that the attack affect a single ablated version will be

$$Pr[\text{at least one attacked pixel is retained}] = \frac{(m+s-1)^2}{hw}.$$

For band smoothing, a single band (column or row) of pixels of width s will be retained per ablated version of the input image, thus for any adversarial patch of size $m \times m$, the probability that the attack affect a single ablated version will be

$$Pr[\text{at least one attacked pixel is retained}] = \begin{cases} \frac{m+s-1}{w} & \text{for column smoothing} \\ \frac{m+s-1}{h} & \text{for row smoothing} \end{cases}.$$

Notably, for both Block Smoothing and Band Smoothing, the number of all possible ablated versions is much smaller (i.e. hw, w, h for block smoothing, column smoothing and row smoothing, respectively) compared to Randomized Ablation, making it possible to iterate through all of them at inference and make the defenses deterministic.

A.3 Model Implementation

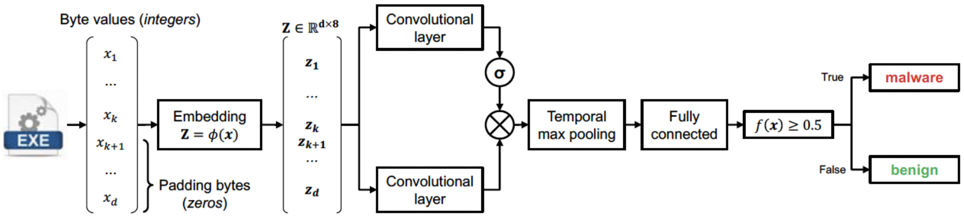


Figure 7: MalConv model architecture

For MalConv and MalConv (NonNeg) implementation, we used input size of $2MB$. For our optimizer, we used –

- Optimizer: `SGD`
- learning-rate: 0.01

- momentum: 0.9
- nesterov: `True`
- weight-decay: $1e-3$

We used the same setting for every model – MalConv, MalConv (NonNeg), and DRSM-n. For training on VTFeed and our dataset, the batch size was 16 and 32, respectively. All the models were re-trained for 10 epochs. We trained the models using multiple gpus at different times. But mostly used gpus were 4 NVIDIA RTX A4000 and 2 RTX A5000.

A.4 More on Results

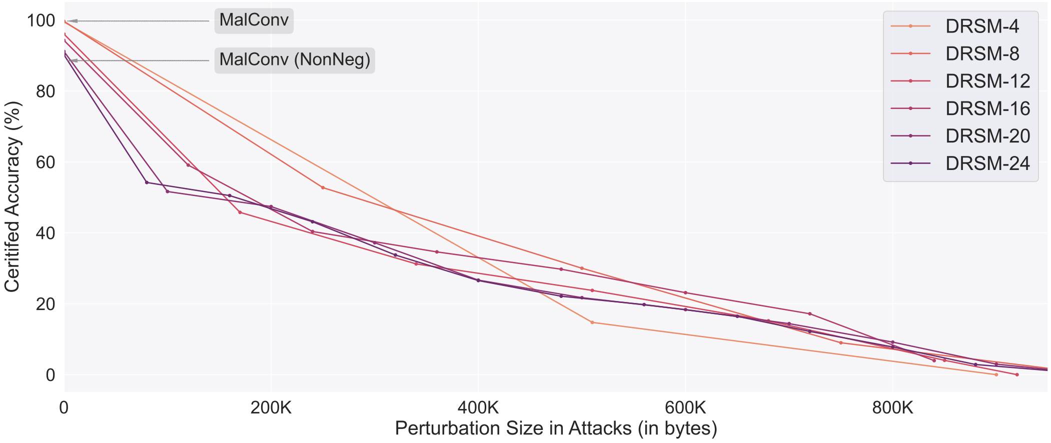


Figure 8: Certified Accuracy (%) of DRSM-n models for different perturbation budgets (Train-set)

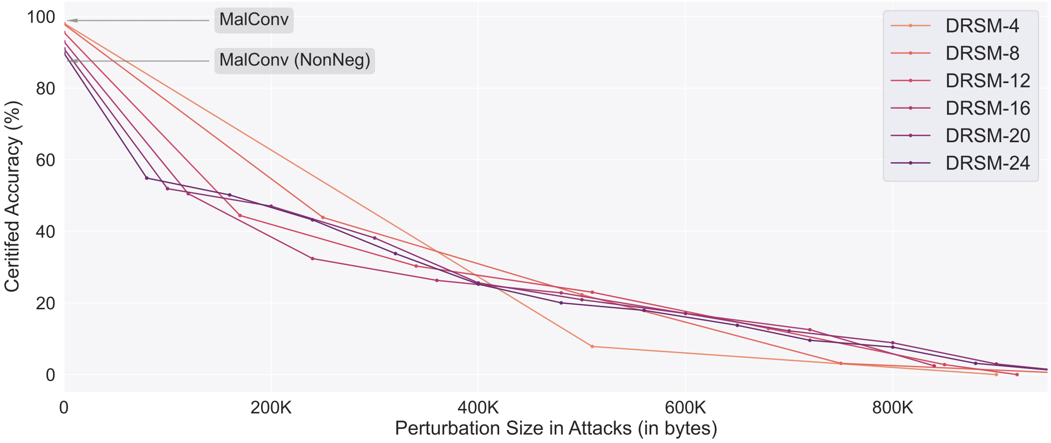
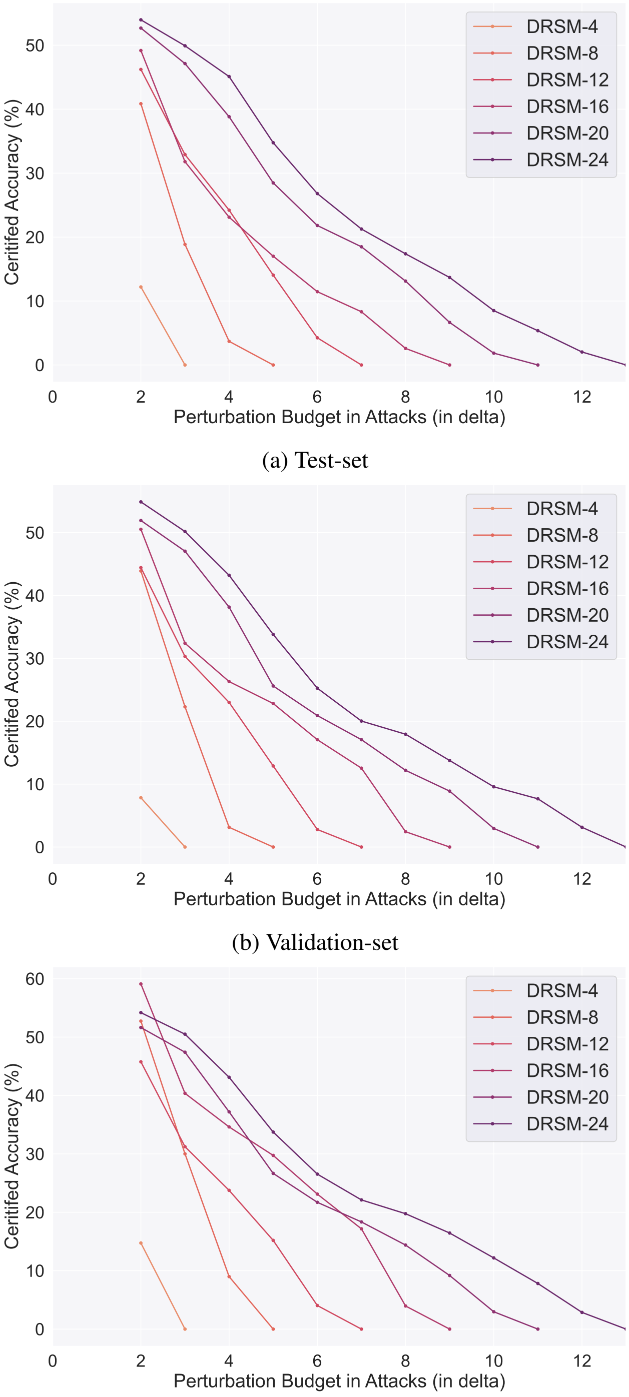


Figure 9: Certified Accuracy (%) of DRSM-n models for different perturbation budgets (Validation-set)

A.4.1 DOS Extension Attack on DRSM

Comparatively, the DOS Extension attack has higher ASRs than other attacks on DRSM-n models, and the reason might be – the extended bytes in the DOS header shift the bytes in later windows and have an implicit impact on them. A future mitigation can be – extracting sections from the file and training different base classifiers on each of them. Thus, attacks DOS extension will not be able to impact other (or later windows).



(a) Test-set

(b) Validation-set

(c) Train-set

Figure 10: Certified Accuracy (%) of DRSM-n models in terms of Δ for each set

Table 6: Certified Accuracy (in %) shown as a range for different Δ

Model	Certified Accuracy (in %) $\uparrow$		
	Train-set	Validation-set	Test-set
MalConv	–	–	–
MalConv(NonNeg)	–	–	–
DRSM-4	14.74	7.84	12.2
DRSM-8	$52.74 \sim 8.99$	$43.9 \sim 3.14$	$40.85 \sim 3.7$
DRSM-12	$45.77 \sim 4.04$	$44.43 \sim 2.79$	$46.21 \sim 4.25$
DRSM-16	$59.1 \sim 3.96$	$50.52 \sim 2.44$	$49.17 \sim 2.59$
DRSM-20	$51.64 \sim 2.97$	$51.92 \sim 2.96$	$52.68 \sim 1.85$
DRSM-24	$54.19 \sim 2.86$	$54.88 \sim 3.14$	$53.97 \sim 2.03$

Table 7: Certified Accuracy (in %) for different perturbation budget for all models

Model	Perturbation	Certified Accuracy (in %) $\uparrow$		
	Budget (in bytes)	Train-set	Validation-set	Test-set
MalConv	-	-	-	-
MalConv (NonNeg)	-	-	-	-
DRSM-4	200K	14.74	7.84	12.2
DRSM-8	250K	52.74	43.9	40.85
	500K	30.01	22.3	18.85
	750K	8.99	3.14	3.7
DRSM-12	170K	45.77	44.43	46.21
	340K	31.23	30.31	32.9
	510K	23.76	23.0	24.21
	680K	15.19	12.89	14.05
	850K	4.04	2.79	4.25
DRSM-16	120K	59.1	50.52	49.17
	240K	40.37	32.4	31.79
	360K	34.62	26.31	23.11
	480K	29.74	22.82	17.01
	600K	23.12	17.07	11.46
	720K	17.17	12.54	8.32
	840K	3.96	2.44	2.59
DRSM-20	100K	51.64	51.92	52.58
	200K	47.41	47.04	47.13
	300K	37.2	38.15	38.82
	400K	26.66	25.61	28.47
	500K	21.71	20.91	21.81
	600K	18.35	17.07	18.48
	700K	14.39	12.2	13.12
	800K	9.18	8.89	6.65
	900K	2.97	2.96	1.85
DRSM-24	80K	54.19	54.88	53.97
	160K	50.5	50.17	49.91
	240K	43.12	43.21	45.1
	320K	33.74	33.8	34.75
	400K	26.54	25.26	26.8
	480K	22.12	20.03	21.26
	560K	19.76	17.94	17.38
	650K	16.45	13.76	13.68
	720K	12.19	9.58	8.5
	800K	7.81	7.67	5.36
	880K	2.86	3.14	2.03

Table 8: Attack Success Rate (ASR) % of different evasion attacks in White-box setting

Attack (White-box)	Attack Success Rate (%) ↓							
	Models							
	MalConv	MalConv(NonNeg)	DRSM-4	DRSM-8	DRSM-12	DRSM-16	DRSM-20	DRSM-24
FGSM Append	82.50	23.5	10.00	7.00	1.00	1.00	1.00	1.00
Slack Append	79.50	17.0	22.50	25.50	1.00	1.00	1.00	1.00
DOS Extension	98.00	26.5	72.00	78.00	12.50	2.00	1.50	1.00
Header Field Modification	26.50	58.5	10.00	17.00	1.00	1.00	1.00	1.00
DOS Modification (Partial)	16.50	58.5	7.50	12.00	1.00	1.00	1.00	1.00
DOS Modification (Full)	49.50	58.00	41.50	41.50	5.00	1.00	1.00	1.00
Disp	89.50	67.5	42.00	35.50	12.50	5.50	1.00	1.00
IPR	18.00	8.00	9.50	5.00	2.50	1.00	1.00	1.00

Table 9: Attack Success Rate (ASR) % of different evasion attacks in Black-box setting

Attack (Black-box)	Attack Success Rate (%) ↓							
	Models							
	MalConv	MalConv(NonNeg)	DRSM-4	DRSM-8	DRSM-12	DRSM-16	DRSM-20	DRSM-24
GAMMA	24.00	45.50	4.00	1.50	1.00	1.00	1.00	1.00
Header Field Modification	11.00	55.50	0.00	4.50	0.00	0.00	0.00	0.00
DOS Modification (Partial)	6.00	54.00	0.00	3.00	0.00	0.00	0.00	0.00
DOS Modification (Full)	10.00	55.50	0.00	0.00	0.00	0.00	0.00	0.00

A.5 Attacks

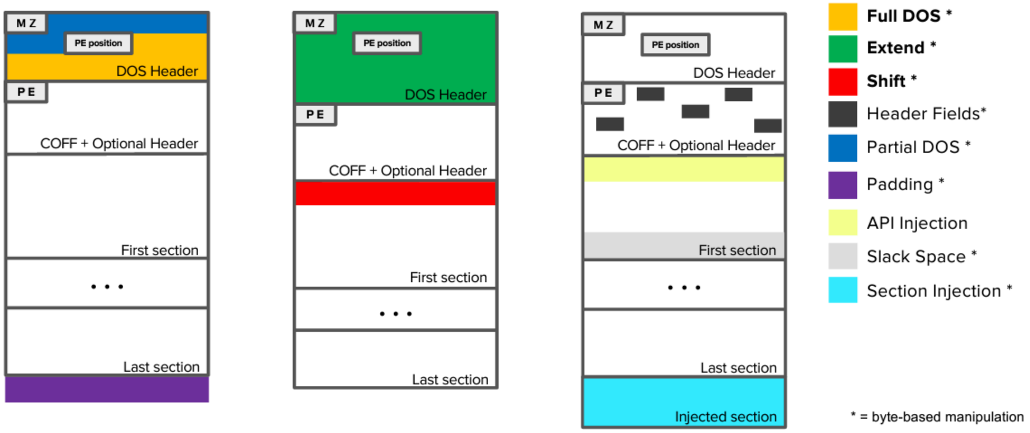


Figure 11: Graphical Representation of the locations perturbed by different attacks with adversarial payloads

A.5.1 Implementation Sources

For FGSM append, Slack append, DOS extension, DOS modification, and Header field modification attacks, we used the code implementaiont of `'secml-malware'` [3]python library which is public and easily reproducible. We collected the guided-varsion implementation of Disp and IPR attacks from the auhtor Lucas et al. (2021). Since their implementation was private, we cannot release the code for these two attacks, and they are not reproducible from our provided code.

A.5.2 FGSM Append Attack

Append attack in adversarial malware was first proposed by Kolosnjaji et al. (2018). In this attack, authors added some noise at the end of a malware file computes by gradient of the model. However, the first proposed method was computationally heavy. Later, it was improved by Kreuk et al. (2018) using Fast Gradient Signed Method (FGSM) motivated by Goodfellow et al. (2015). In Figure 11, the ‘Padding’ label (purple) depicts the FGSM Padding (or Append) attack.

In our experiment, we kept a padding budget of 10KB ($= 0.5\%$ of the input file size) and ran the attack for 10 iterations. We noticed that, for some malwares the model prediction was 1.0 for which the attack failed.

A.5.3 Slack Append Attack

This attack was an incremental work on Kreuk et al. (2018) by Suciu et al. (2019). Unlike the previous one, the attacker can inject the payload in between of sections. The find the gaps between consecutive sections (called ‘slack spaces’) in a binary by $RawSize - VirtualSize$, and use that gap to inject gradient-generated adversarial bytes. Since these slack spaces can be at multiple places, this attack is partially inside our threat model. In Figure 11, the ‘Slack Space’ label (grey) indicates this attack.

In our experiment, we followed the same parameter as the previous one, and ran it for 10 iterations by keeping the padding budget 10KB. We want to mention that – though this attack seems more evasive than the previous one, for larger perturbation budget FGSM Append is more successful than this one. This was found in original paper, and our result is consistent with this finding too.

A.5.4 DOS Extension Attack

This attack creates a new space by extending the DOS header. Attacker increases the offset to PE header and modify the file structure accordingly. In these extended spaces, attacker can put

Figure 11 is taken from Demetrio et al. (2021b).

[3] https://github.com/pralab/secml_malware

adversarial bytes to evade a model (Demetrio et al., 2021b). Since the extension is on a contiguous portion (header) of the file and can have an impact on other windows in DRSM-n, it partially falls under our threat model. In Figure 11, the ‘Extend’ label (green) refers to this attack. We ran this attack on our ‘attack set’ for 10 iterations with 10^{-3} penalty regularizer.

A.5.5 DOS Modification Attack

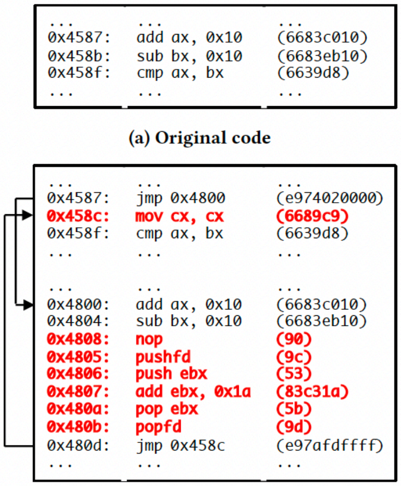
There are 2 versions of this attack – Partial(Demetrio et al., 2019), and Full(Demetrio et al., 2021b). In DOS header, two important bytes are – magic number `MZ` and real offset `0x3c`. The former attack modifies bytes in between of these two bytes while the latter one modified every bytes in the DOS header except those two. So, the ‘full’ modification version is more evasive than the ‘partial’ one. This attack is shown in blue and yellow color in Figure 11. We ran this attack on our models for 10 iterations.

A.5.6 Header Field Modification Attack

This attack was implemented getting the motivation from Nisi et al. (2021). They analyzed the discrepancies among tools and PE file formats. Thus, they found a set of bytes (or modifications) that can potentially evade a malware classifier. Since this attack modifies bytes at multiple different places but they are constrained only in the PE header, it is partially inside our threat model. In Figure 11, the ‘Header Fields’ label (black) shows how this attack changes header fields in PE header. We ran this attack for 20 iterations.

A.5.7 Disp (Code Displacement) Attack

In this attack, the attacker has to use to disassemble a malware first. Then the attacker displace consecutive instructions in a basic block. Such displacements are usually done `jmp` and `nop` instructions. Lucas et al. (2021) proposed this attack for the first time. Figure 12 shows an example of such attack.



```
...         ...            ...
0x4587:     add ax, 0x10   (6683c010)
0x458b:     sub bx, 0x10   (6683eb10)
0x458f:     cmp ax, bx     (6639d8)
...         ...            ...
```

(a) Original code

```
...         ...            ...
0x4587:     jmp 0x4800     (e974020000)
0x458c:     mov cx, cx     (6689c9)
0x458f:     cmp ax, bx     (6639d8)
...         ...            ...

...         ...            ...
0x4800:     add ax, 0x10   (6683c010)
0x4804:     sub bx, 0x10   (6683eb10)
0x4808:     nop            (90)
0x4805:     pushfd         (9c)
0x4806:     push ebx       (53)
0x4807:     add ebx, 0x1a  (83c31a)
0x480a:     pop ebx        (5b)
0x480b:     popfd          (9d)
0x480d:     jmp 0x458c     (e97afdffff)
...         ...            ...
```

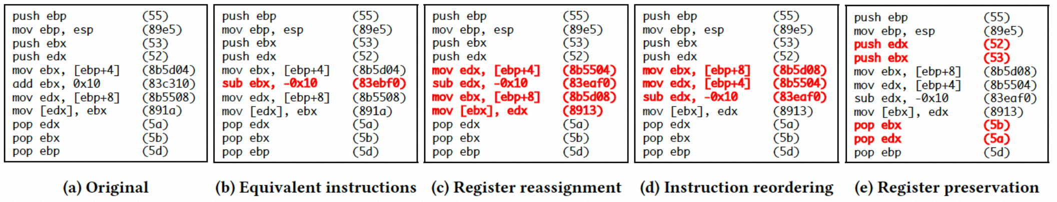
(b) After *Disp*

Figure 12: An example of Disp attack

We collected the private implementation of the guided version of this attack from the authors. We ran Disp-1 (the perturbation budget is 1% of the binary size) for 100 iterations.

A.5.8 IPR (In-Place Randomization) Attack

Like the previous attack, attacker has to disassemble the malware here. Then attacker can apply four types of transformations – i) replacing instructions with equivalent ones, ii) reassigning registers, iii) reordering instructions using dependency graph, and iv) altering register’s push and pop order. These transformations do not necessarily change the file size but it modifies the code at many different places. So, this attack falls outside of our threat model. Figure 13 shows the transformation types with an example. We collected the private implementation for this attack from authors of Lucas et al. (2021).



```
push ebp               (55)
mov ebp, esp           (89e5)
push ebx               (53)
push edx               (52)
mov ebx, [ebp+4]       (8b5d04)
add ebx, 0x10          (83c310)
mov edx, [ebp+8]       (8b5508)
mov [edx], ebx         (891a)
pop edx                (5a)
pop ebx                (5b)
pop ebp                (5d)
```

(a) Original

```
push ebp               (55)
mov ebp, esp           (89e5)
push ebx               (53)
push edx               (52)
mov ebx, [ebp+4]       (8b5d04)
sub ebx, -0x10         (83ebf0)
mov edx, [ebp+8]       (8b5508)
mov [edx], ebx         (891a)
pop edx                (5a)
pop ebx                (5b)
pop ebp                (5d)
```

(b) Equivalent instructions

```
push ebp               (55)
mov ebp, esp           (89e5)
push ebx               (53)
push edx               (52)
mov edx, [ebp+4]       (8b5504)
sub edx, -0x10         (83eaf0)
mov ebx, [ebp+8]       (8b5d08)
mov [ebx], edx         (8913)
pop edx                (5a)
pop ebx                (5b)
pop ebp                (5d)
```

(c) Register reassignment

```
push ebp               (55)
mov ebp, esp           (89e5)
push ebx               (53)
push edx               (52)
mov ebx, [ebp+8]       (8b5d08)
mov edx, [ebp+4]       (8b5504)
sub edx, -0x10         (83eaf0)
mov [ebx], edx         (8913)
pop edx                (5a)
pop ebx                (5b)
pop ebp                (5d)
```

(d) Instruction reordering

```
push ebp               (55)
mov ebp, esp           (89e5)
push edx               (52)
push ebx               (53)
mov ebx, [ebp+8]       (8b5d08)
mov edx, [ebp+4]       (8b5504)
sub edx, -0x10         (83eaf0)
mov [ebx], edx         (8913)
pop ebx                (5b)
pop edx                (5a)
pop ebp                (5d)
```

(e) Register preservation

Figure 13: An illustration of IPR attack

A.5.9 GAMMA Attack

This attack was first proposed by Demetrio et al. (2021a). Though it was a common belief that – goodware (or benign) payload (or string) can be added to a malware to evade a model, they are the first to propose a query-based black-box method for this. In this attack, the attacker generates payload from some benign programs, then inject them into a malware and return the best subset of generations by querying the model. Figure 14 shows the overview of the attack.

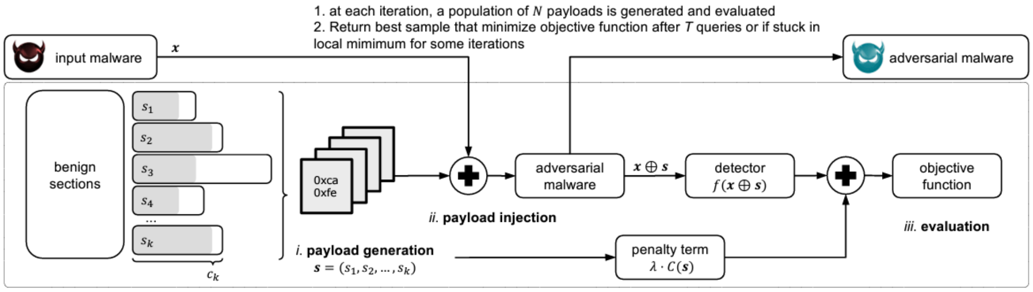


Figure 14: Overview of GAMMA attack

In our experiment, we ran the hard-label version of GAMMA attack with section injection. We set the population size as 200, and ran it for 20 iterations. For payload extraction, we used the `.data` section of benign files.

For Disp and IPR attacks, we used IDAPro disassembler.
Figure 12 and 13 are taken from Lucas et al. (2021).
Figure 14 is taken from Demetrio et al. (2021a).