

# Certifiably robust malware detectors by design

**Abstract**—Malware analysis involves analyzing suspicious software to detect malicious payload. Static malware analysis, which does not require software execution, relies increasingly on machine learning techniques to achieve scalability. Although such techniques obtain very high detection accuracy, they can be easily evaded with adversarial examples where a few modifications of the sample can dupe the detector without modifying the behavior of the software. Unlike other domains, such as computer vision, creating an adversarial example for malware without altering its functionality requires specific transformations. This article proposes a taxonomy of the transformations an attacker can use depending on the threat models that modelize their capability. We show the effectiveness of this taxonomy by proposing a new set of features and model architecture that can lead to certifiably robust malware detection by design. In addition, we show that every robust detector can be decomposed into a specific structure, which can be applied to learn empirically robust malware detectors, even on fragile features. Our framework ERDALT is based on this structure. We compare and validate these approaches with various machine-learning-based malware detection methods, allowing for robust detection with limited detection performance reduction.

## 1. Introduction

Malware infection is a major cybersecurity threat that evolves quickly. In 2022, a new ransomware incident occurred every 2 seconds and, according to Cybersecurity Ventures<sup>1</sup>, ransomware’s damage cost will reach 265 billion USD by 2031. This threat is fought with many malware analysis techniques developed by the industry and academia.

These techniques can be broadly categorized into two kinds: static analysis and dynamic analysis. Static analysis does not run the program and relies on descriptive features such as section description, control-flow graph, and opcodes n-grams. This analysis is widely used because it is fast and inexpensive but can be evaded by code obfuscation or packing (code compression or encryption). On the other hand, dynamic analysis executes the code in a controlled sandbox. This technique requires more equipment and time, but it is difficult for the malware to hide its malicious payload when activated. However, malware can evade dynamic analysis by verifying whether they are used in a virtual machine or any monitoring environment. Furthermore, dynamic analysis typically requires “fresh” malware because samples over a few months old generally cannot reach the command-and-control servers that have been shut down. For these reasons, our article focuses on

the static analysis of malware targeting Windows, the most popular and targeted desktop operating system.

Many traditional detection techniques used by anti-virus are based on pattern matching, relying on indicators of compromise from threat intelligence (file hashes, IP addresses of command-and-control servers, suspicious domains, etc.) or on YARA rules (consisting of a set of specific strings in the binary). However, these techniques are difficult to scale up and cannot reliably detect new variants or malware families. Machine learning techniques have been very useful in addressing these two issues [32]. These methods can achieve very high accuracy [31]. However, they are vulnerable to attacks called *adversarial examples*. Adversarial examples are samples with intentional perturbations that are optimized to result in the model giving an incorrect prediction or reduced calibration. While first identified in the field of computer vision [39], where a subtle and imperceptible noise added to pixels can completely change the class prediction, such adversarial examples have been found in many domains, including malware analysis [38].

Several methods have been proposed to improve the robustness of machine learning models against adversarial attacks. These defenses can provide empirical robustness (e.g., adversarial training [16]) or guaranteed robustness against “small” perturbation (e.g., Lipschitz-constrained networks [40] and randomized smoothing [9]) but with a significant loss of utility [8]. While these methods can be used for static malware analysis, they do not take advantage of the specificity of the domain. The key constraint in designing adversarial samples for evading malware detection comes from the discrete nature of malware and the highly constrained set of transformations the attacker can use to evade malware detectors. Besides, most of the methods with guaranteed robustness assume that the perturbation is small: while this hypothesis is relevant in computer vision, malware attackers only need the perturbation to conserve the functionality and the the magnitude of perturbation itself is not of significance.

Based on this assessment, we rely on a framework where the attacker’s capability is not limited by the magnitude of perturbation that can be applied on a malware sample but by the ability to use certain transformations. To this end, we propose a new taxonomy of adversarial transformations and deduce features the attacker cannot arbitrarily modify, no matter how many transformations they use, leading to a robust-by-design malware detector. We extend this proposition by characterizing certifiably robust malware detectors, leading us to a second defense against adversarial examples: the framework ERDALT. This defense does not require expert knowledge about the threat model and the attacker’s capability but extracts this knowledge from examples of adversarial attacks, leading to an empirically robust by design malware detector.

1. <https://shorturl.at/lyHVY>

Experiments show the trade-off between robustness and detection performances for various models with several defenses.

The contributions of this paper can be summarized as follow:

- a taxonomy of transformations used for crafting adversarial malware sample;
- a feature sets allowing for robust-by-design detectors;
- a characterization of certifiably robust detectors;
- ERDALT, a framework to learn empirically robust detectors from any feature set.

The paper is structured as follow. Section 2 positions our proposal concerning related work. We propose of taxonomy of threats in Section 3. Section 4 present our analysis of certifiable robust malware detector by design. Application to empirically robust malware with the ERDALT framework is presented in Section 5. Section 6 details our experiments. Section 7 contains an ablation study of the ERDALT framework and Section 8 concludes the article.

## 2. Related work

While most commercial anti-viruses rely on a database of known signatures to recognize malware, the surge of machine learning in recent decades allows detectors to identify unseen malware that would not correspond to any signature. Machine learning techniques require data to perform their analysis. While using the whole binary as input to a detector has been successfully tested with the MalConv detector [31], most methods require a feature extraction step [32]. Feature extractions for malware analysis are generally split into static and dynamic categories. One the one hand, static features are extracted from the binary without executing it. It is fast and harmless, but obfuscating techniques (such as packing) can hide part of the content of the malware. The static features proposed in EMBER [3] are widely used with machine learning models. On the other hand, dynamic features are extracted from an execution of the binary. The binary has fewer opportunities to hide its malicious payload, but the analysis requires a secured environment (for example, a virtual machine) and takes more time than static analysis. Besides, there is no guarantee to observe the malicious payload. Many machine learning models have been used with these features, such as decision trees, SVM, and neural networks [32].

Deep learning models has led to their adoption in safety-critical and high risk applications like malware detection [42]. With their increasing popularity, attack methods such as evasion attacks has also gained attention among both researchers and practitioners [39]. Some of the seminal initial efforts showing the adversarial vulnerability of deep learning methods are evasion attacks using adversarial examples [15], [39]. Consequently, adversarial robustness has emerged as an area of interest for deep learning research and the robustness to adversarial attacks has become an important factor in evaluating systems.

Several methods focus on provable guarantees that the network is robust to bounded adversarial perturbations. However, exactly solving the certification problem is NP

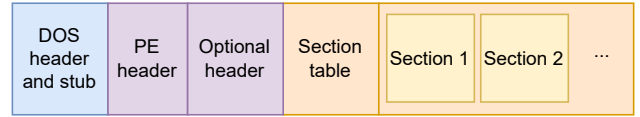


Figure 1: Format of a PE file

complete [20]. We argue that domain specific intuitions can help mitigate adversarial risks by shifting the focus from the magnitude of perturbation to other measures more meaningful in each domain.

As stated earlier, one security issue with neural networks is the existence of adversarial attacks that allow the malware to evade detectors. Two kinds of techniques for crafting adversarial examples have been proposed: black-box attacks that have only access to the output of a detector (sometimes only the majority class, sometimes the weight of each class) and white-box attacks that know the model’s architecture and weight to evade. White-box attacks are generally much easier to perform because efficient algorithms such as the Fast Gradient Sign Method (FGSM) [15] or the Carlini/Wagner attack [6] have been proposed. Black-box attacks are more difficult due to the lack of information about the model’s weights. They can be performed by first learning a surrogate model and then computing an adversarial example with a white-box attack [7]. This attack succeeds because experiments show that adversarial examples can generally be transferred from one model to another [27], [24].

Various research works have successfully performed adversarial attacks for evading malware detectors and show how vulnerable some classifiers are [25]. Two families of evasion attacks have been encountered: attacks on problem space and attacks on feature space [30]. Attacks on problem space seek to directly create new binaries that evade classifiers, while attacks on feature space seek to create features that evade classifiers. This distinction is not relevant, e.g., for computer vision where the feature extraction is reversible, so it’s trivial to transpose an attack on an image file to an attack on image features, and vice versa. However, static and dynamic feature extractions in malware analysis are generally neither reversible nor differentiable. For this reason, even if the attacker crafts an adversarial features vector, it is not trivial to construct a malware sample with the corresponding features. Attack on problem space (i.e., on binaries) is almost always black-box attacks due to the non-invertible and non-differentiable feature mapping that make classical white-box evasion attacks like FGSM impossible. Most black-box attacks rely on transformation of the structure and content of PE files, whose format is sketched in Fig. 1.

A few recent surveys [5], [29], [25] present adversarial attacks for malware analysis. They can be divided into roughly four categories:

- feature-space and white-box attacks, that attacks deep-learning models such as MalConv or convolutional networks that work on grayscale images extracted built from binaries [41], [44];
- problem-space and white-box attacks: using gradient-descent-based attacks, they target only MalConv as it is the main deep learning model

that works directly on the binary without explicit feature extraction;

- feature-space and black-box attacks: they use white-box attacks to evade a surrogate model learned to copy a black-box model;
- problem-space and black-box attacks: they modify iteratively the malware with a set of transformations guided by reinforcement learning [2], [37], evolutionary algorithms [11], [13], [23] or explanations [33]. Binary generation with generative adversarial networks (GAN) has also been proposed.

Even if most attacks are based on gradient descent, the realism of such attacks is disputed [4] because simpler and cheaper attacks are generally available and effective. In fact, the authors of [4] advocate for cost-driven analysis of attackers' capabilities to ensure that attacks are realistic and defenses tackle actual threats.

General defense methods against adversarial attacks have been proposed [36], such as adversarial training, regularization approach, gradient masking, and adversarial example detection. However, these methods are generally limited to perturbations with a small magnitude, but this assumption is not realistic in malware adversarial examples. For this reason, several protections specific to malware analysis have been proposed. For Android malware analysis, [14] proposes to use Lipschitz-bounded linear classifier to improve the robustness. For Windows malware analysis, [18] uses randomized smoothing to improve the robustness, but it significantly lowers the detector's accuracy.

[19] proposes to rely on monotonic models to be robust against some attacks. However, the loss of detection performance is significant: after applying the defense, the recall of the method dropped from 75% recall to 62%, making the detector much less useful. Besides, the only monotonic model they experiment with is XGBoost, which has not the same scalability as deep neural networks.

Our work expands on the idea proposed by [19] to rely on monotonicity to provide robustness against adversarial examples. First, we extend their work on the feature categories that can be modified or not by attacker with a cost analysis: depending on the capability of the attackers, the set of transformations they can use changes, and therefore the set of fragile features changes as well. Second, we characterize robust classifiers as the composition of a feature mapping and a monotonic classifier, showing that monotonicity can be seen as a universal tool for constructing robust classifiers. Finally, we show that the performance drop can be mitigated with manual or automatic feature mapping engineering.

### 3. Taxonomy of threats



In domains like computer vision, the typical assumption about the adversarial attack is that the perturbation is small, i.e., within a  $\epsilon$ -ball [1]. This assumption is depicted in Figure 2 (left part), where any picture  $P'$  within a  $\epsilon$ -ball centered around  $P$  is an adversarial example candidate for  $P$ . This hypothesis is motivated by the fact that adversarial examples in computer vision aim to perturb as little as possible the content of the picture so humans cannot not

detect the attack. For example, a spam that poses as an official bank message could use an adversarial attack to evade a spam detector checking for logos. In that case, the logo would still look genuine to the user. However, this hypothesis does not hold in malware analysis, where no end-user "looks" at the binary: the utility of the attack does not decrease with the size of the perturbation.

However, even though attackers are not subjected to a budget on the amount of perturbation they can apply to malware, they still require it to function properly (i.e., stealing credentials, encrypting disks, mining cryptocurrency, etc.). For this reason, as presented in the related work, adversarial attacks against malware analysis are generally performed in the problem space (i.e., the space of executable binaries) instead of the feature space, as it is the case for computer vision. Attacks generally rely on atomic transformations such as section addition, static import removal, or DOS stub modification to alter the malware in the problem space. This is depicted in the right part of Figure 2, where the set of adversarial example candidates (in white) is the set of binaries with the same behavior. In this figure, a program  $P$  is transformed into a functionally equivalent program  $P'$  with the transformation  $T$ .

Plenty modifications are available to the attacker, and they do not all have the same cost. For example, off-the-shelf tools can automatically add sections to a binary, making such adversarial transformations easy to automate [11], but modifications like static import removal requires access to the malware source code, which may not be the case in the context of the malware-as-a-service business [10].

To take into account the various cost of these malware transformations, we propose several threat models based on two parameters:

- Does the attacker has access to the source code? This ability is denoted with .
- Does the attacker has the skill and time to modify the malware? This ability is denoted with .

Classical transformations used for adversarial attacks against malware analysis and their required capability are presented in Table 1. These categories represent our current knowledge. However, future attacks and findings may make some transformations easier and change their category. Remark that many transformations are easy to perform in one direction (e.g., file access addition) but requires more capabilities to perform in the other direction (e.g., file access deletion). Besides, we do not consider packing in this table: although this technique can alter many static features, the packers themselves can still be identified.

This taxonomy of threats is independent of the actual features used for classification. It contrasts with several works that classify features by their ability to be modified by transformations [19], [37]. Indeed, in these articles, the authors rely on a given features mapping and verify which features is fragile and which one is not. Our methodology is different: we point out that, for given capabilities, the attacker can perform several transformations, so we can engineer features that can be used for robust malware detection.

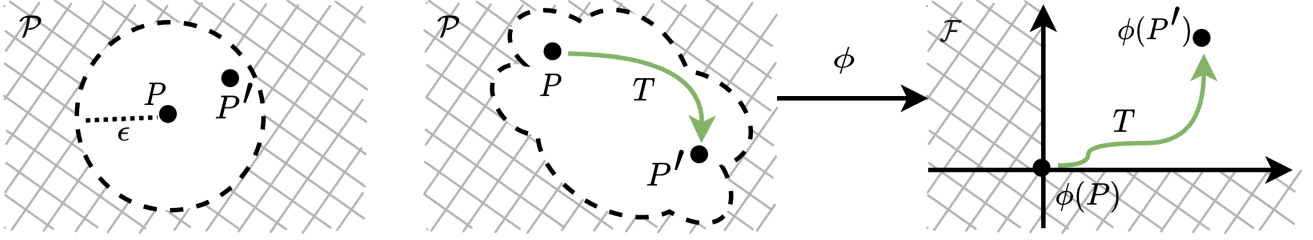


Figure 2: (a) Adversarial perturbation in the problem space of images.

(b) Adversarial perturbation in the problem space, and its correspondence in feature space. The crosshatched part of the space cannot be reached by perturbation.

TABLE 1: Transformation used for adversarial attacks, whether they impact static (S) or dynamic (D) analysis, and the required capability to perform them

Transformation	S	D	Required capability
DOS header modification	✓		none [12], [11]
Optional header modification	✓		none [11]
Padding addition	✓		none [21]
Content shifting	✓		none [11]
Semantical nope insertion	✓	✓	none [28], [37]
Remove signature	✓		none
Add trustworthy signature	✓		✎
Readable strings addition	✓		none
Readable strings removal	✓		none
Static import addition	✓		none [11]
Static import removal	✓		📄
Embedded resources addition	✓		none
Embedded resources removal	✓		📄+✎
Bytes n-grams modification	✓	✓	none [43]
Opcodes n-grams modification	✓	✓	none [43]
Byte/section entropy	✓		none
Section addition or extension	✓		none [11]
Section deletion	✓		📄+✎
File access addition		✓	none
File access removal		✓	📄+✎
Registry access addition		✓	none
Registry access removal		✓	📄+✎
System/API call addition		✓	none [22]
System/API call removal		✓	📄+✎
System/API call n-grams modification	✓	✓	none [22]
CPU/Memory/IO usage modification		✓	📄
Control-flow graph modification	✓	✓	none
Grayscale image modification	✓		none
Using undocumented Windows API	✓	✓	📄+✎

TABLE 2: Manually selected features from static analysis

Feature	Type
Has the DOS stub been modified?	Boolean
Does the PE has resources?	Boolean
Does the PE have no signature?	Boolean
Number of section	Count
Size of .text section	Integer
Total entropy of .text section	Integer
Size of .data section	Integer
Total entropy of .data section	Integer
Size of .rsrc section	Integer
Total entropy of .rsrc section	Integer
Size of .rdata section	Integer
Total entropy of .rdata section	Integer
#imported functions from kernel32.dll	Count
#imported functions from shell32.dll	Count
#imported functions from user32.dll	Count
#imported functions from comctl32.dll	Count
#imported functions from gdi32.dll	Count
#imported functions from urlmon.dll	Count
#imported functions from ntdll.dll	Count
#imported functions from winmm.dll	Count
#imported functions from advapi32.dll	Count
#imported functions from wininet.dll	Count
#imported functions from ole32.dll	Count
#imported functions from mscoree.dll	Count
#imported functions from gdiplus.dll	Count
#imported functions from oleaut32.dll	Count
#imported functions from msvbvm50.dll	Count
Number of imported functions from other dll	Count
Total number of imported functions	Count
#imported function name containing "Process"	Count
#imported function name containing "Thread"	Count
#imported function name containing "Get"	Count
#imported function name containing "Set"	Count
#imported function name containing "File"	Count
#imported function name containing "Write"	Count
#imported function name containing "System"	Count
#imported function name containing "Console"	Count
#imported function name containing "Delete"	Count

Each feature of a feature set can be classified depending on the minimal capability required to modify it. A feature that can be modified with off-the-shelf tools is more fragile than a feature that can only be modified with manual modification of the malware. From Table 1, we propose a new set of features manually selected to be difficult to change for an attacker with no particular capability. These features are described in Table 2. They contain indicators of adversarial transformation, like DOS stub modification and signature removal, as well as features that are difficult to reduce automatically, like the number of sections, the total entropy of a section, the number of statically imported functions, and the number of keywords used in imported functions. There are only 40 features, a very limited compared to classical features mapping like EMBER, which contains several thousands of features. In the following, we will use this set of features as a baseline for how robust and accurate classifiers are when

learned with a feature mapping designed with adversarial examples in mind. The source code for extracting these manually selected features is available online<sup>2</sup>.

#### 4. Certifiable robust detector by design

In this section, we prove that monotonic models can lead to robust by design detectors. We also prove that the converse is true: any robust detector can be decomposed as a monotonic detector on a latent feature space.

We introduce  $M$ , the transformations available in a threat model. From this set, we define the preorder  $\preceq_M$  in the space of programs such that  $P \preceq_M P'$  if the program

2. removed for anonymous submission

$P$  can be transformed into  $P'$  with transformations available in  $M$ . We can remark that this preorder is reflexive and transitive, but it is generally not complete (we may have  $P_1 \not\preceq_M P_2$  and  $P_2 \not\preceq_M P_1$ ) nor anti-symmetric (we may have  $P_1 \preceq_M P_2$  and  $P_2 \preceq_M P_1$ ). Remark that this preorder is not computable. We can formally define a robust detector for such a preorder as a detector whose decision cannot be modified from "malicious" to "benign" by applying transformations from  $M$  on the original program.

**Definition 1.** Let  $M$  be a set of transformations,  $C : \mathcal{P} \rightarrow \mathbb{R}$  a classifier and  $\tau$  its decision threshold.  $C$  is said to be robust against adversarial attacks if, for any program  $P$  and  $P'$  such that  $P \preceq_M P'$ ,  $C(P) \geq \tau \implies C(P') \geq \tau$ .

As mentioned by [30], the notions of problem space and feature space are prevalent in malware analysis. The problem space, denoted  $\mathcal{P}$ , is the set of programs. By far, the most common approach in machine learning for malware analysis is to extract features from programs. Let  $\phi$  be a mapping from the problem space  $\mathcal{P}$  to the feature space  $\mathcal{F}_\phi$ . In malware analysis,  $\phi$  is generally not invertible nor differentiable. The definition above can be adapted to reflect the use of a features mapping:

**Definition 2.** Let  $M$  be a set of transformations,  $\phi : \mathcal{P} \rightarrow \mathcal{F}$  a feature mapping,  $C : \mathcal{F} \rightarrow \mathbb{R}$  a classifier and  $\tau$  its decision threshold.  $C$  is said to be robust against adversarial attacks if, for any program  $P$  and  $P'$  such that  $P \preceq_M P'$ ,  $C(\phi(P)) \geq \tau \implies C(\phi(P')) \geq \tau$ .

This definition allows us to reason separately about the feature mappings  $\phi$ , generally standardized within the domain, and the classifiers  $C$ , which can be based on many machine learning techniques. In particular, the classifiers are typically general-purpose algorithms and models not dedicated to malware analysis. Thus, they cannot exploit the specific structure of the problem space of binaries. This is the role of the feature mapping, written by malware experts, to extract relevant data such as the ones described in the previous section. For this reason, we argue the adversarial examples issue should not be solved directly by the classifiers but by the features mapping itself.

Following this guideline, we identify two ways for obtaining robust classifiers: 1) only extract features that are hard for the attacker to modify and use any classifier, and 2) only extract monotonic features, i.e., that the attacker can only increase, and use a monotonic classifier. As shown in the previous subsection, almost every feature is easy to modify, at least partially, by the attacker, so the first strategy is moot for malware analysis. The second strategy is based on a monotonic feature mapping that ensures that, if  $P \preceq_M P'$ , then  $\phi(P) \leq \phi(P')$ , where  $\leq$  is the product order on  $\mathcal{F}$ .

This second strategy is illustrated in Fig. 2: with a monotonic feature mapping, any transformation  $T$  in the problem space is mapped to a transformation that cannot decrease the value of any feature. In other words, given some binary  $P$ , the attacker can only produce an adversarial example in the first quadrant relative to  $P$ . It is straightforward that the monotonicity of  $\phi$  and  $C$  are indeed sufficient for  $C$  to be robust.

**Proposition 1.** Let  $M$  be a threat model,  $\phi$  a feature map-

ping and  $C$  a classifier. If  $\phi$  is monotonically increasing w.r.t.  $\preceq_M$  and if  $C$  is monotonically increasing w.r.t.  $\leq$ , then  $C$  is robust against adversarial attacks for the threat model  $M$ .

*Proof.* Let  $P$  be a malware correctly detected by  $C$  and  $P'$  an adversarial example of  $P$  in the threat model  $M$ . By definition of  $\preceq_M$ ,  $P \preceq_M P'$ . Due to the monotonicity of  $\phi$ ,  $\phi(P) \leq \phi(P')$ . Due to the monotonicity of  $C$ ,  $C(\phi(P)) \leq C(\phi(P'))$ . Since  $P$  is correctly detected by  $C$ ,  $C(\phi(P)) \geq \tau$ . So  $C(\phi(P')) \geq C(\phi(P)) \geq \tau$  and  $P'$  does not evade  $C$ . Therefore,  $C$  is robust.  $\square$

The feature mapping presented in Table 2 is monotonic when the attacker has no particular capability. Therefore, robustness is guaranteed when such a feature set is used with a monotonic classifier (this claim is experimentally assessed in Section 6). Remark that some common feature engineering strategies, such as histograms, are typically not monotonic. For example, in the classical EMBER features set [3], there are bytes histograms of readable strings. However, even though it is difficult for the attacker to remove readable strings (cf. Table 1), they can easily reduce the proportion of one byte by adding new strings with other bytes. For these reasons, such features should be avoided because they cannot be used reliably with monotonic classifiers.

In the following, we show that such monotonic classifiers are not just yet another tool to obtain robustness against adversarial examples. In fact, with the right feature post-processing, every robust classifier can be interpreted as a monotonic classifier, as shown by the following proposition:

**Proposition 2.** Let  $M$  be a threat model,  $\phi$  a feature mapping and  $C$  a classifier such that  $C$  is robust against adversarial attacks for the threat model  $M$ . There exist  $f$  and  $g$  such that  $C \circ \phi = C \circ g \circ f \circ \phi$  and  $C \circ g$  is monotonically increasing.

*Proof.* Let  $\mathcal{P}$  the set of executable binaries and  $\mathcal{D}_\phi$  the codomain of  $\phi$ . Consider the following complete pre-order  $\preceq_\phi$  defined on  $\mathcal{D}_\phi \times \mathcal{D}_\phi$ :  $\phi(P) \preceq_\phi \phi(P')$  if and only if  $C(\phi(P)) \leq C(\phi(P'))$ . Since  $C \circ \phi$  is robust by assumption, this pre-order satisfies the property:  $P \preceq_M P' \implies \phi(P) \preceq_\phi \phi(P')$ .

Denote  $k$  the dimension of the feature space and  $f$  any strictly monotonically increasing function from  $(\mathcal{D}_\phi, \preceq_\phi)$  into  $(\mathcal{D}_f, \leq)$ , where  $\mathcal{D}_f = f(\mathcal{D}_\phi) \subseteq \mathbb{R}^k$ . So, if  $v_1 \preceq_\phi v_2$ , then  $f(v_1) \leq f(v_2)$ . Remark that  $f$  is surjective (by definition of its codomain) but generally not injective: indeed, two programs  $P_1$  and  $P_2$  such that  $\phi(P_1) \preceq_\phi \phi(P_2)$  and  $\phi(P_2) \preceq_\phi \phi(P_1)$  will be mapped to the same vector due to the anti-symmetry of  $\preceq$ . Let  $g$  be a function defined on  $\mathcal{D}_f$  such that  $g(v) \in f^{-1}(v)$  (the latter is a set because  $f$  is generally not injective). Let us show that  $C \circ \phi = C \circ g \circ f \circ \phi$ . Let  $P \in \mathcal{P}$ , let  $v_1 = \phi(P)$  and  $v_2 = g(f(v_1))$ . Due to the definition of  $f$  and  $g$ ,  $f(v_1) = f(v_2)$ . Since  $\preceq_\phi$  is a complete pre-order, there are only three possibilities:  $v_1 \preceq_\phi v_2$  (but in that case,  $f(v_1) < f(v_2)$ ),  $v_2 \preceq_\phi v_1$  (but in that case,  $f(v_2) < f(v_1)$ ), or  $v_1 \sim_\phi v_2$ . Only the last case is possible. Due to the definition of  $\preceq_\phi$ , we can conclude that  $C(v_1) = C(v_2)$ , i.e.,  $C(\phi(P)) = C(f(g(\phi(P))))$ . Therefore,  $C \circ \phi = C \circ g \circ f \circ \phi$ .

Let us now show that  $C \circ g$  is monotonically increasing. Let  $v_1, v_2 \in \mathcal{D}_\phi$  such that  $v_1 \leq v_2$ . By definition of  $g$  and  $f$ ,  $g(v_1) \preceq_\phi g(v_2)$ . Due to the definition of  $\preceq_\phi$ , it implies that  $C(g(v_1)) \leq C(g(v_2))$ . Therefore,  $C \circ g$  is monotonically increasing.  $\square$

This result shows that if there exists a robust classifier  $C$  with good performances, then with the correct feature preprocessing  $f$ , one can learn a monotonic detector on the features  $f \circ \phi$  and obtain the same performances as  $C$  while keeping the certifiable robustness. Such monotonic classifiers have been proposed for malware analysis [19]. Our work shows that 1) this idea is not specific to malware analysis but stems from the asymmetrical cost of the transformations in the problem space, 2) such classifiers can be certified to be robust with respect to a threat model, and 3) that all robust classifier can be expressed as a monotonic classifier with some change to the feature mapping.

The next section shows how we leverage Proposition 2 to propose a new framework, ERDALT, that learns feature preprocessing alongside a monotonic classifier to obtain a more robust classifier.

## 5. ERDALT: an empirically robust by-design malware detector

We previously proposed a feature set (cf. Table 2) that can be used with a monotonic model. This feature set has been manually selected, and we certainly missed monotonic features that could help the classifier be more discriminating. This section explores another approach: since any robust classifier can be decomposed into a post-processing function  $f$  and a monotonic function  $C$ , we propose to learn these two functions  $f$  and  $C$  jointly to obtain a robust classifier. This way, one can rely on usual (and fragile) features and let the learning procedure craft monotonic features with the post-processing function. To perform this learning, we assume we have access to adversarial examples alongside original samples so the model can learn what the attacker can (and cannot) do. This assumption is akin to what requires other protection techniques such as adversarial training.

Many typical features from EMBER or from our own feature set are monotonic with respect to common transformations. Let's take an example of a typical transformation by substitution that can break this monotonicity. Suppose the attacker can replace API calls with equivalent ones, such as replacing `CreateFile` with `CreateFileEx`. In that case, the two features that count the number of `CreateFileEx` and `CreateFile` calls are not monotonically increasing, but the effect of the transformation on the features is linear. Intuitively, the post-processing  $f$  could build a feature that counts the occurrences of either `CreateFileEx` or `CreateFile`. This feature would not be affected by the transformation and could be used with a monotonic classifier.

In this section, we call *perturbation vector* the difference of features between the transformed software and the base software, any  $\phi(T(P)) - \phi(P)$  for any  $T$  and  $P$ . For example, if a feature is unchanged by  $T$ , then its value is 0 in the perturbation vector. In the previous example of the transformation by substitution, the perturbation vector

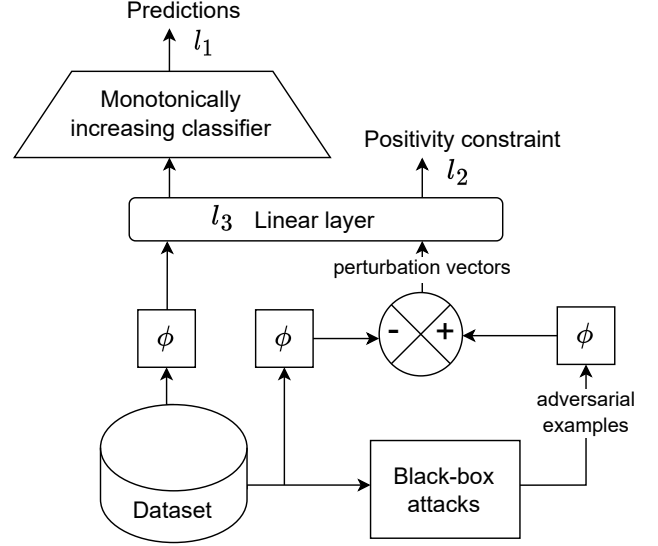


Figure 3: ERDALT: the proposed framework. The first dense layer is a preprocessing step applied to the feature, akin to the  $f$  function in Proposition 2. This network seeks to minimize the error of detection  $l_1$  while ensuring the positivity of the perturbations with the loss  $l_2$ . Finally, the loss  $l_3$  encourages the linear layer to be close to a diagonal matrix.

will be 1 for the count of `CreateFileEx`, -1 for the count of `CreateFile`, and 0 for the rest of the vector.

To simplify the learning and help the generalization, we propose to introduce an assumption that would still allow to tackle this kind of transformations by substitution: we assume perturbation vectors related to one transformation does not depend on the modified software itself. This is the case for the previous example: the perturbation vector does not depend on the original software. This hypothesis can be mathematically formalized as follow: for a threat model  $M$ , the feature mapping  $\phi$  is such that the set of perturbation vectors, i.e.  $\{\phi(T(P)) - \phi(P) \mid T \in M, P \in \mathcal{P}\}$ , is finite. We denote this set of perturbations vectors  $\Delta_M$ .

We propose the deep learning architecture presented in Figure 3 to learn an empirically monotonic feature mapping from examples of attacks. It is composed of a linear layer (corresponding to the post-processing function  $f$ ) and a monotonic classifier (corresponding to  $C \circ g$ ).

The linear layer must be a linear function, without any bias nor activation function. It is fitted such that, when the input is the perturbation vectors, its output is always positive. Such behavior is enforced with the  $l_2$  loss. This constraint guarantees the monotonicity of the feature post-processing.

The upper layers should be monotonic. Monotonicity can be achieved for neural networks with different techniques [26], [34], [35]. Its loss  $l_1$  is any typical loss function used for classification.

This framework is robust by design, as shown by the following proposition. For this reason, we name it “Empirically Robust by Design with Adversarial Linear Transformation” (ERDALT for short).

**Proposition 3.** Let  $M$  be a threat model and  $\phi$  a feature



mapping such that  $\Delta_M = \{\phi(T(P)) - \phi(P) \mid T \in M, P \in \mathcal{P}\}$  is finite. If  $\Delta_M$  is known during training, then ERDALT is robust.

*Proof.* Let us decompose the architecture in two parts: the first dense layer  $L$ , that have no bias nor activation, and the upper layers  $L'$ . The first layer is constrained such that, for all perturbation vector  $\delta \in \Delta_M$ ,  $L(\delta) \geq 0$ . The upper layers have a monotonicity constraint, so if  $v_1 \leq v_2$  then  $L'(v_1) \leq L'(v_2)$ . Let us show that  $L' \circ L$  is robust. Let  $P$  and  $P'$  be two programs such that  $P \preceq_M P'$ . It means there exists a succession of  $n$  transformations  $T_1, T_2, \dots, T_n$  such that  $P' = (T_n \circ \dots \circ T_2 \circ T_1)(P) = (\bigcirc_{k=1}^n T_k)(P)$ . So:

$$\begin{aligned} \phi(P') - \phi(P) &= \phi((\bigcirc_{k=1}^n T_k)(P)) + \sum_{j=1}^{n-1} \phi((\bigcirc_{k=j}^1 T_k)(P)) \\ &\quad - \sum_{j=1}^{n-1} \phi((\bigcirc_{k=j}^1 T_k)(P)) - \phi(P) \\ &= \sum_{j=1}^n \phi((\bigcirc_{k=j}^1 T_k)(P)) - \sum_{j=0}^{n-1} \phi((\bigcirc_{k=j}^1 T_k)(P)) \\ &= \sum_{j=1}^n \left( \phi((\bigcirc_{k=j}^1 T_k)(P)) - \phi((\bigcirc_{k=j-1}^1 T_k)(P)) \right) \\ &= \sum_{j=1}^n \left( \phi(T_j \circ (\bigcirc_{k=j-1}^1 T_k)(P)) - \phi((\bigcirc_{k=j-1}^1 T_k)(P)) \right) \end{aligned}$$

We can remark that this last sum only contains elements of  $\Delta_M$ . Let us denote them  $\delta_i$ . Due to the linearity of  $L$ :

$$L(\phi(P')) - L(\phi(P)) = L\left(\sum_i \delta_i\right) = \sum_i L(\delta_i) \geq 0$$

So,  $L(\phi(P)) \leq L(\phi(P'))$ . Since  $L'$  is monotonically increasing,  $L'(L(\phi(P))) \leq L'(L(\phi(P')))$ . This proves the robustness of  $L' \circ L$ .  $\square$

The linear layer of ERDALT can make linear combinations of its input, but will also drop the features that cannot be made monotonic with respect to the threat model. In this sense, it also serves as automatic feature selection.

For example, consider an attacker that can use an invertible transformation  $T$ . Since  $T$  is invertible, it means that, if  $\delta \in \Delta_M$  for some  $P$  and  $T$ , then  $-\delta \in \Delta_M$  for  $T(P)$  and  $T^{-1}$ . Since  $L$  is constrained by  $L(\delta) \geq 0$  and  $L(-\delta) = -L(\delta) \geq 0$ , we can conclude that  $L(\delta) = 0$ . So, these features affected by  $T$  are discarded by  $L$ , and this transformation won't affect the detector's output.

The advantage of this method is that it does not require expert knowledge like we use to create the features set in Table 2: as long as adversarial examples are available, possibly by using off-the-shelf tools or adversarial examples found in the wild, this method can be used. However, it is not robust by design but only empirically robust by design: indeed, if the adversarial training set is incomplete, the model could be evaded.

## 6. Experiments

In this section, we experimentally assess the contributions on robust classifiers, including the framework ERDALT. We seek to answer the following research questions:

- What is the impact of the features on the detection performances and the robustness performances?
- What protection method allows for the best trade-off between detection performances and robustness?
- How do the defense mechanisms perform w.r.t. the number of training adversarial examples?
- How do the PV feature selection and the linear layer of ERDALT compare?
- How transferable is the robustness of ERDALT against unseen attacks?
- How robust are the methods we propose against attacks with more budget?

### 6.1. Experimental protocol

**6.1.1. Dataset.** There are no standard PE executable datasets for malware analysis. Widespread Windows malware datasets, like EMBER2017, EMBER2018 [3] or SOREL-20M [17], do not contain actual executable samples but only features. We cannot use these datasets since we compare methods with various feature sets.

For these reasons, we rely on a new dataset<sup>3</sup> containing 670 families of malware, each one with 100 different samples, for a total of 67000 malware. The goodwill dataset contains 16611 samples, collected from Windows default installation and various online repositories like Chocolatey<sup>4</sup>. In the following, we create a balanced training dataset containing 120 families of malware (so 12000 samples) and 12000 goodwill. The testing set is composed of 4611 goodwill and 550 families of malware (so 55000 samples). The hashes of the train and test datasets are available in our repository.

**6.1.2. Evaluation metric.** We use the classical area under the ROC (or ROC AUC for short) metric to compare the detection performances. The ROC AUC is the area under the receiver operator characteristic curve, which plots the false positive rate against the true negative rate with respect to a decision threshold. In binary classification, like in malware detection, the ROC AUC typically ranges from 0.5 (random classifier) to 1 (perfect discrimination).

**6.1.3. Features mapping.** As discussed in Section 3, feature mappings can have a significant impact on both the detection accuracy and the robustness to adversarial attacks. To assess such effects in malware analysis, we experiment with two feature sets:

- a new set of manually selected features presented in Table 2;
- the state-of-the-art EMBER features proposed by [3] that are widely used for static Windows malware analysis.

3. removed for anonymous submission

4. <https://chocolatey.org/>

EMBER contains 2381 features: 256 features for bytes histograms, 256 features for bytes entropy, 104 features for string description, 10 features for general section description (number of sections, number of empty sections, number of sections with an empty name, number of readable and executable sections, number of writable sections), 62 features for header description, 255 features for section description (including name, size, entropy, virtual size and properties), 1280 features for imports description, 128 features for exports description and 30 features for data directories description.

**6.1.4. Models.** We evaluate two groups of models:

- classical models widely used in machine learning: neural networks,  $k$ -nn, decision trees, random forests, gradient-boosted trees and AdaBoost;
- monotonic models: monotonically increasing gradient-boosted trees and monotonic neural network;

The  $k$ -nn algorithm makes predictions for a vector by producing a majority vote of the closest vectors in the dataset. They are generally effective but can be easily evaded with fragile features. Decision trees are explainable machine learning models that predict according to comparisons between the feature values and some references. Multiple models leverage a set of decision trees. Random Forest is a bagging ensemble method based on decision trees, where each tree is trained with a subset of the training data, and a majority vote over these trees produces the prediction. AdaBoost and gradient-boosted trees are boosting ensemble methods, where trees are greedily added to improve the overall model. The neural networks we use are multi-layer perceptrons, i.e., they have several linear layers separated with nonlinear activation functions.

We use the scikit-learn implementation of the non-deep learning models with the default parameters. Here are the implementation details of the deep learning models.

**Baseline neural network:** we train a fully connected network with four hidden layers. The first hidden layer has the same number of nodes as the input, the subsequent layers have 800 nodes, and the output is a single node enabling binary classification. We use a dropout with a keep-probability of 50%, after the first three layers. The model is trained with binary cross-entropy loss ( $l_1$ ) between the classifier prediction and the binary label (malware or not).

**Monotonic classifier:** to make the output monotonically increasing with respect to the input, we constrain all weights to be positive and choose monotonically increasing activation functions. We follow the method in [35] to make the classifier network provably monotonic by construction: after each training step, the negative weight parameters are updated with  $w \leftarrow e^{w-\epsilon}$ , where  $\epsilon$  is a hyperparameter for curbing the blowing up of weight values. We choose  $\epsilon = 5$  across all the experiments. In short, the parameter update follows:

$$W \leftarrow W - \alpha \nabla l(W; X)$$

$$w_i \leftarrow e^{w_i - \epsilon}, \text{ if } w_i < 0 \quad \forall w_i \in W$$

We choose ELU as the activation since it allows the propagation of negative values in the network. With positively constrained weights, choosing ELU for the task

helps retain negative values flowing to subsequent layers in the network. This enables learning a richer representation, retaining fundamental properties like identity mapping, which are not achievable using ReLU in a monotonically increasing network. The implementation is available online<sup>5</sup>.

**6.1.5. Protection against adversarial attacks.** We experiment with four protection methods against adversarial attacks: adversarial training, feature selection, our framework ERDALT, and a hybrid between ERDALT and adversarial training. Remark that all these methods require adversarial examples so they can be fairly compared. Each method has access to 1033 adversarial examples produced during the experiment described in Table 3.

Adversarial training is a classical method widely used to protect models against evasion. It works by adding adversarial examples to the training data. This technique has been successfully used in malware analysis as well [24].

ERDALT is implemented as follows. The feature projection layer is trained with a loss that penalizes projecting the perturbation vectors outside the first quadrant. Let us denote  $\Delta$  the perturbations vectors available in the training set. Then, for all  $\delta \in \Delta$ , the output of the projection layer  $L$  with length  $m$  respects:  $L(\delta)_i \geq 0, \forall i \in m$ . The parameters in  $L$  are not constrained for monotonicity. The projection is achieved by optimizing a hinge loss,  $l_2 = \max(L(\Delta), 0)$ . The output of the projection layer is the input to the monotonic classifier, as indicated in Figure 3.

To avoid the first layer from learning spurious combinations of input features that can reduce the robustness of the learned features, we use a third loss  $l_3$ . This loss encourages the model to select input features and not learn combinations of features with small coefficients, so the weight matrix of the learned model is higher in the diagonal entries. More precisely,  $l_3$  is computed as the sum of squared amplitudes of non-diagonal elements. The layer weights are initialized with an identity matrix with random elements in the diagonal.

We train all three models (base model, monotonic models, and ERDALT) for up to 300 epochs with a batch size of 16 and Adam optimizer with an initial learning rate of 0.001. The three losses (binary cross-entropy  $l_1$ , projection loss  $l_2$ , and diagonalizing loss  $l_3$ ) are combined with a scheduler that follows a linear increase in coefficients:  $l_1$  from 0.5 to 1,  $l_2$  from 0.5 to 200,  $l_3$  from 0.5 to 1. The model selection is done using  $l_1$  loss computed over the validation split, given  $l_2$  loss over the perturbation vectors is zero.

We propose to compare ERDALT with another approach based on feature selection. Given a dataset of perturbation vectors  $\Delta$ , this procedure identifies all features with non-negative perturbation, i.e., that are monotonic with respect to this set of attacks. When used jointly with a monotonic model, such models should be robust, as shown by Proposition 1. We call this method ‘‘perturbation-vectors-based features selection’’, or PV feature selection for short.

5. removed for anonymous submission



Finally, we propose to use adversarial training with the framework ERDALT to verify whether both techniques can profit from each other.

**6.1.6. Adversarial attacks.** The adversarial attacks rely on the secml-malware package developed by [13]. These attacks are black-box functionality-preserving transformations on Windows malware aiming at evading static analysis. They are very effective at evading detectors, even with minimal modifications. We rely on eight attacks proposed by this package, namely:

- 1) header perturbation (section names, checksum, etc.)
- 2) file padding (injecting bytes at the end of the file)
- 3) section padding (injecting bytes at the end of sections)
- 4) DOS header modification and extension
- 5) section shifting with additional bytes
- 6) addition of API imports found in goodwill samples
- 7) addition of sections found in goodwill samples
- 8) extension of PE header

Two more attacks, one based on adding sections found in goodwill samples and one based on adding sections found in goodwill samples caused random segfault crashes so we excluded them from our study.

## 6.2. What is the impact of the features set on robustness and performances?

In this experiment, we evaluate the robustness and the detection performances of various models applied to both feature sets. For each experimental setup, we run the eight attacks on 200 malware samples correctly identified as malware. Each attack has a 60-second timeout for scalability's sake. Therefore, the attack budget is reduced, and the robustness results here should be considered an upper bound of the actual robustness against an attacker with more resources. The robustness is defined as the proportion of decisions evaded by at least one attack.

Table 3 presents the overall robustness and detection performances. Let's have a look at the EMBER feature set. We can immediately remark most models using EMBER have a good AUC (higher than 90%), which is typical in malware detection. However, these models are generally vulnerable to adversarial attacks.  $k$ -nn is the most vulnerable model (all attacks succeeded). The other models can resist some attacks. The baseline neural network is the most robust model on EMBER by a slight margin. It has a lower ROC AUC than non-deep models: this is a common observation in malware static analysis.

Our manually selected features yield more robust models: they all have at least 93% robustness, and some of them were never evaded. Remark the significant impact of feature mapping on the robustness:  $k$ -nn classifier is typically not robust (all malware have successfully evaded the detectors based on EMBER), but its robustness with manual robustness is 93.5%. Besides, as expected, using the manual features with a monotonic model allows for 100% robustness, although the classification performance is much lower. The best trade-off between ROC AUC and

robustness is, in our opinion, the random forest model with manual features, with 94.6% AUC and 98.5% robustness.

In this experiment, the results indicate that classical machine learning methods outperform neural networks in terms of performance metrics. However, it is crucial to direct our attention towards enhancing the robustness of neural networks, given their inevitable adoption in future applications. While classical machine learning methods may exhibit superior performance in certain datasets, the unparalleled potential of neural networks lies in their scalability, adaptability, and ability to handle complex and unstructured data. As data volumes continue to grow, neural networks are poised to play a pivotal role in leveraging, especially domains like malware detection. To harness their full potential, it is imperative to focus on improving the robustness and reliability of neural networks, ensuring their effectiveness across diverse and dynamic datasets.

This experiment shows the drastic effect of the features set on both the detection performances and the robustness of the models: even models that are easy to attack can become quite robust with manually selected features. However, detection performances and robustness are typically in a trade-off regarding features set: EMBER has many more features than our manual selection, helping both the detection by containing more information and the attacker that can rely on many fragile features to evade detectors. Finally, our manually selected features have required a risk analysis and a good knowledge of attacker capability, so this approach could not be transposed to another domain without the cooperation of an expert. For this reason, we evaluate in the next section several empirical approaches to make models more robust without needing an expert.

## 6.3. How do the defense mechanisms compare?

In this subsection, we compare the defense mechanisms presented in Section 6.1.5, namely PV feature selection, adversarial training, ERDALT, and ERDALT combined with adversarial training.

The results are shown in Table 4. Contrary to the previous subsection, we do not evaluate all machine models because they behave similarly and we focus on the most promising models. As we can see, the PV feature selection approach yields the most robust models but with some detection performance penalties. In fact, this method shows that with access to adversarial examples, the automatic extraction of non-fragile features can yield performances comparable, and even better, to the manual extraction by an expert. Note, however, that the PV feature selection does not have the same guarantees of robustness that a risk analysis can provide.

Adversarial training can be used to get models that are quite robust (about 95% of attacks failed) without performance penalty compared to the results from Table 3. So, compared to the PV feature selection, they are slightly less robust but more effective at discriminating malware from goodwill. Remark that both methods, PV feature selection and adversarial training, can be used with any model.

Our framework ERDALT provides similar results in terms of robustness as adversarial training. The difference in ROC AUC between ERDALT and adversarial training can probably be explained by the use of machine learning

Model	Manual features		EMBER	
	ROC AUC	Robustness	ROC AUC	Robustness
Baseline neural network	89.9%	<b>100%</b>	91.6%	<b>82.0%</b>
Monotonic neural network	69.0%	<b>100%</b>	87.4%	71.5%
Random Forest	<b>94.6%</b>	98.5%	96.2%	81.0%
AdaBoost	85.0%	98.0%	94.2%	75.5%
<i>k</i> -nn	83.7%	93.5%	88.6%	0%
Decision tree	84.1%	99.5%	96.2%	67.0%
Monotonic gradient-boosted trees	76.2%	<b>100%</b>	92.7%	73.5%
Gradient-boosted trees	92.3%	99.0%	<b>97.5%</b>	75.0%

TABLE 3: Comparing the performance of different models over the three features set without specific adversarial protection

Protection	Model	EMBER	
		ROC AUC	Robustness
PV feature selection	Random Forest	95.2%	<b>100%</b>
	Monotonic gradient-boosted trees	86.7%	<b>100%</b>
	Gradient-boosted trees	93.8%	<b>100%</b>
Adversarial training	Random Forest	<b>97.6%</b>	94.5%
	Monotonic gradient-boosted trees	92.7%	95.5%
	Gradient-boosted trees	<b>97.6%</b>	96.5%
ERDALT	Neural network	93.0%	96.0%
ERDALT and adversarial training	Neural network	85.5%	<b>100%</b>

TABLE 4: Comparing the performance of different models with PV feature selection, adversarial training, ERDALT, and ERDALT with adversarial training

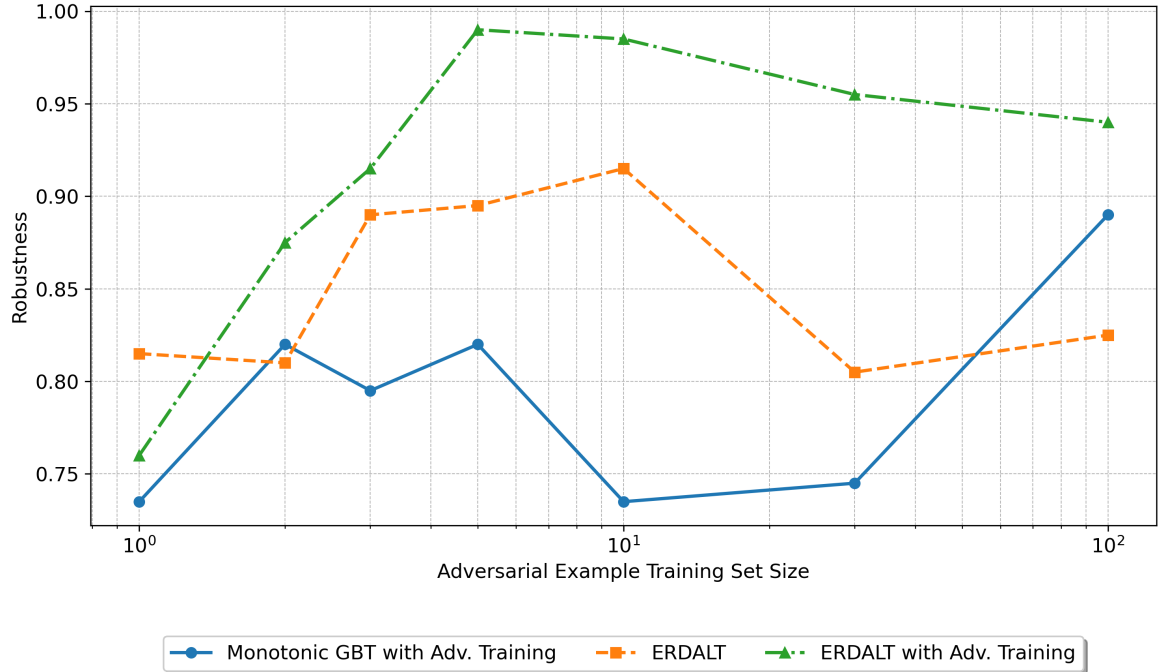


Figure 4: The robustness of ERDALT, adversarial training and ERDALT combined with adversarial training with respect to the number of adversarial examples available during training.

with adversarial training and the use of a neural network for ERDALT. However, these similar result should not be interpreted as ERDALT working in a similar fashion as adversarial training: as shown by the last line in Table 3, both method can be combined to obtain an even more robust model (in fact, no attack succeeded), at the cost of some performance penalty.

#### 6.4. How the defense mechanisms perform w.r.t. the number of training adversarial examples?

In the previous section, we showed the performances of various defense mechanisms with access to a set of about 1000 adversarial examples. This high number of adversarial examples can be seen as unrealistic in the context of malware analysis, where hundreds of variations of one malware family can be found but with most certainly a low amount of adversarial examples. In this section, we study the effect of the number of adversarial examples available at training on the robustness and the ROC AUC of the four defense mechanisms. The results are presented in Fig. 4 and 5, where ERDALT is represented with the blue line, adversarial training is represented with the green line and ERDALT combined with adversarial training is represented with the orange line.

The results are noisy. This is due in part to the learning process and the attack process. In particular, the black-box attacks work with stochastic modifications. However, we can nonetheless make some conclusions from these Figures.

As expected, the more adversarial examples are available to the method, the more robust it is (cf. Fig. 4). We can see that the robustness induced by adversarial training is typically below the robustness obtained with ERDALT, and that combining ERDALT with adversarial training yields the most robust models. These results are consistent with the previous results. Remark that only a handful of examples are sufficient to get models that are 90% robust. This is probably due to the simple model used for robustness in ERDALT: only the linear layer actually depends on the adversarial examples, and it does not require many examples to be fit.

Figure 5 shows that the number of adversarial examples does not highly impact both ERDALT and adversarial training ROC. It is more difficult to conclude for ERDALT combined with adversarial training due to the noisy data. The fact that the most robust models also have the lowest ROC AUC has already been observed. Remark, however, that this loss of performance is reduced: adversarial training and ERDALT are only separated by a few percents.

#### 6.5. How does the feature selection methods compare?

Two protection methods rely on feature selection: the PV feature selection, which only keeps the features that cannot be decreased by attacks, and the first layer of ERDALT that, besides combination between features, will also drop the fragile features. For the linear layer of ERDALT, we considered that a feature is selected if the sum of the weight associated with it is greater than 3.

	PV feature selection	ERDALT selection	Intersection
Byte histogram	0%	86.7%	0%
Byte entropy	0%	83.2%	0%
Strings	1.9%	94.2%	1.9%
General	30.0%	60.0%	30.0%
Header	77.4%	83.9%	64.5%
Section	55.2%	76.5%	40.8%
Imports	44.5%	66.5%	22.2%
Exports	100%	49.2%	49.2%
Data directories	46.7%	90.0%	43.3%

TABLE 5: The proportions of features kept by feature selection (PV feature selection on the left, ERDALT’s linear layer in the middle, and the intersection of the selected features by both methods on the right)

As mentioned in Section 6.1.3, EMBER features can be regrouped into several categories. The amount of features kept for each category is detailed in Table 5.

First, remark that, except for the Exports features, ERDALT selects more features than the PV feature selection. In fact, by looking at the intersection between both feature sets, we can conclude that the features selected by ERDALT are approximately supersets of the features selected by the PV feature selection. This result shows that ERDALT retrieve similar features and can exploit more features by linear combinations. This is typically the case for byte histogram, byte entropy and strings: the PV feature selection considers these features to be fragile, but by combining them, ERDALT is able to create non-fragile features. Remarkably, some features are dropped, notably the Exports features. This is probably because these features are not useful for discriminating goodware from malware. Since ERDALT learns jointly the linear layer that selects features and the monotonic model that performs the detection, it can drop irrelevant features. This is not the case with PV feature selection that only verifies the sign of the perturbation but is not fitted to classify.

Remark, however, that the linear layer of ERDALT tends to avoid unnecessary linear combinations due to the  $l_3$  loss that encourages a diagonal matrix. The effect of this loss can be seen in Figure 6 that shows the structure of the linear layer of the base neural network and ERDALT. We use a measure,  $D(W)$ , to quantify the diagonal dominance of a weight matrix  $W$ . This measure assesses the degree to which the sum of non-diagonal elements surpasses the sum of diagonal elements, providing a quantitative evaluation of how closely a matrix resembles a diagonal matrix. The computation is as follows:

$$D(W) = \frac{\sum_{i=1}^n \sum_{j=1, j \neq i}^n |W_{ij}|}{\sum_{i=1}^n |W_{ii}|}$$

For the weight matrix of the first layer of ERDALT trained with and without  $l_3$  loss, the corresponding  $D$  values are 4.33 and 2410.87, respectively. This 99.82% reduction in the value of  $D$  with the use of  $l_3$  loss shows the transformation of the first layer from a linear layer to a near feature selection layer. This can help curb the possible spurious combinations of input features deriving in fragile features. The linear layer of ERDALT is not completely diagonal: most non-diagonal weights appear within a few

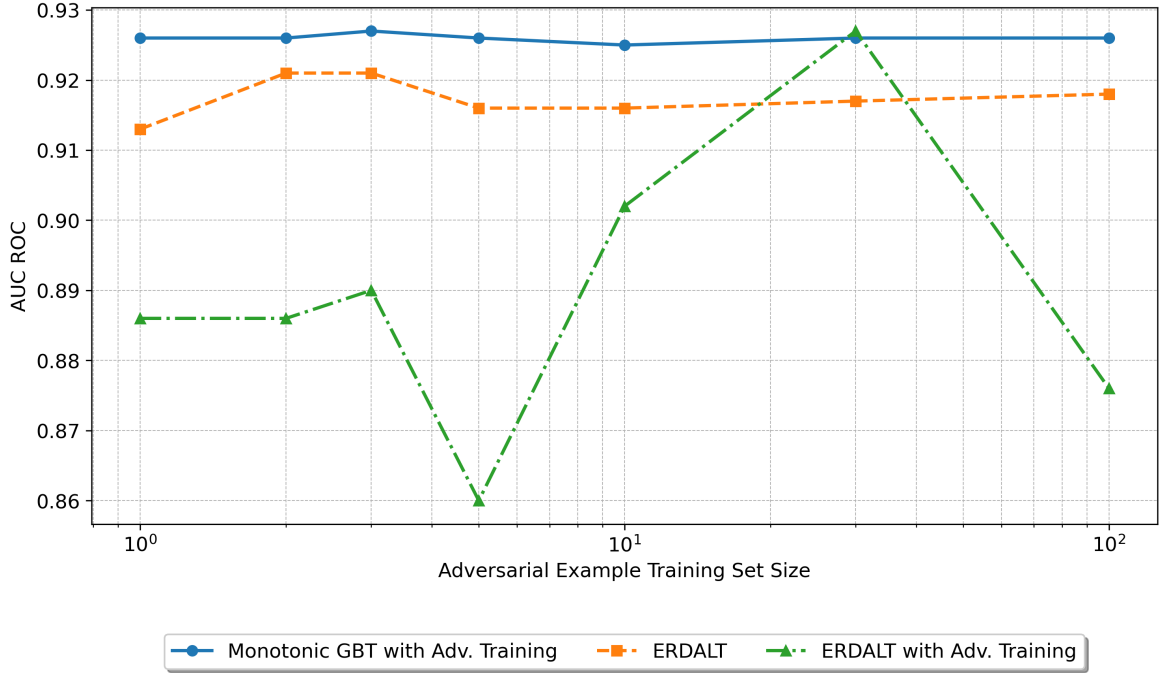


Figure 5: The ROC AUC of ERDALT, adversarial training and ERDALT combined with adversarial training with respect to the number of adversarial examples available during training.

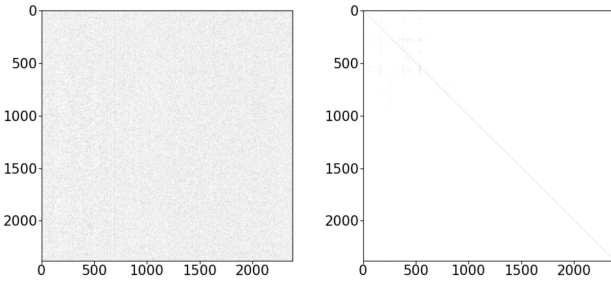


Figure 6: The weights of the linear layer without and with the loss  $l_3$  minimizing spurious combination of features

Attack	ERDALT	Baseline neural network
Header perturbation	<b>0.0%</b>	<b>0.0%</b>
File padding	<b>100%</b>	<b>100%</b>
Section padding	<b>100%</b>	98.7%
DOS header modification	<b>100%</b>	99.7%
Section shifting	<b>100%</b>	98.7%
API addition	<b>100%</b>	<b>100%</b>
Section addition	<b>98.3%</b>	60.3%
PE header extension	<b>96.3%</b>	93.3%

TABLE 6: Robustness of two models against unseen attacks

features corresponding to byte histogram, byte entropy and strings. This remark supports the previous discussion about how ERDALT is able to take advantage of these features thanks to linear combinations while PV feature selection cannot.

### 6.6. Is ERDALT robust against unseen attacks?

In the previous subsections, adversarial examples have been used to learn ERDALT. In the experiment, we evaluate the robustness of our model against unseen attacks.

The experimental protocol is modified to include a leave-one-out cross-validation on the different attacks: for each learning iteration, the model is fitted with all but one attack, and then the learned model is attacked with this left-out attack. We run this experiment on two models: ERDALT and a baseline neural network. Robustness metrics are presented in Table 6.

Except for the first attack (header perturbation), ERDALT is indeed robust against all other attacks, while the baseline is also vulnerable to the section addition attack.

### 6.7. What is the robustness against attackers with better budget?

The previous attacks have been run with a low budget: each attack had a 60-second timeout. In this section, we examine the robustness against much stronger attacks. Due to this higher cost, we only run three different attacks, the one with the highest success rate during the lost-budget attacks: header perturbation, API addition and PE header extension. We also limited this experiment to two models: the monotonic gradient-boosted tree on the manual features datasets (because it is robust by design) and ERDALT. The results are presented in Table 7.

We can first remark that different models are not vulnerable to the same attacks. The random forest is resistant to the PE header extension attack but is easily fooled by the API addition attack. ERDALT is globally robust, even if not very robust against the PE header extension attack. Finally, the monotonic classifier is entirely robust.

Attack	ERDALT on EMBER	Monotonic gradient-boosted trees on manual features
Header perturbation	<b>100%</b>	<b>100%</b>
API addition	<b>100%</b>	<b>100%</b>
PE header extension	71.8%	<b>100%</b>

TABLE 7: Robustness of three models against high-budget attackers

Linear layer	Monotonicity	ROC AUC	Robustness
×	×	91.6%	82.0%
✓	×	<b>94.3%</b>	91.0%
×	✓	87.4%	71.5%
✓	✓	93.0%	<b>96.0%</b>

TABLE 8: ROC AUC and robustness of models with some components of ERDALT removed

## 7. Ablation study of ERDALT

In this section, we assess the effect of the linear layer and the monotonicity constraints on the accuracy and robustness of ERDALT, to show empirically that the specific property of this model emerges from the conjunction of both components and cannot be explained by each component independently. The results are summarized in Table 8.

The baseline neural network has an AUC of 91.6% and a robustness of 82.0%, as seen in the previous section. With the linear layer that projects the  $\delta$  to the first quadrant alone, the robustness is increased to 91.0%. This is consistent with our previous conclusion: the linear layer acts as a feature selection and can ditch fragile features. The neural network with the monotonicity constraint has a far lower ROC AUC: 87.4%, and its robustness is also low: 71.5%. This is consistent with the result of other monotonic models when used on EMBER

The model with both linear layer and monotonic constraints, as proposed in Figure 3, has a much higher robustness (96.0%), and its ROC AUC is close to the baseline model. This experiment shows how it is the joined effect of the linear layer and the monotonic constraints that enable the detector to be both accurate and robust.

## 8. Conclusion and future work

This article focuses on the robustness of machine-learning models against black-box adversarial attacks in the context of malware analysis. In this domain, adversarial attacks are not required to rely on imperceptible perturbations (like in signal processing) but need to preserve the semantics of malware. Such attacks typically rely on semantics-preserving transformations, like API call addition or n-grams modification. We showed that different transformations require different capabilities and, following a risk analysis based on a threat model, deduced a set of manually selected features that are difficult for the attacker to alter with the available semantics-preserving transformation. These manually selected features can be used with a monotonic model to obtain a *robust by design* classifier. We also characterize robust classifiers and deduce a framework named ERDALT that can use

adversarial examples to train a linear layer that selects non-fragile features. This model is *empirically robust by design*, meaning that with enough adversarial examples, it can be guaranteed to be robust. Besides, this framework can be combined with classical adversarial training for an even more robust model. Finally, we show that adversarial examples can be used to perform feature selection and obtain robust models. However, this feature selection is quite aggressive and drops fragile features that could have been combined into non-fragile features by the linear layer of ERDALT. An experimental study and an ablation study back these claims.

ERDALT is not specific to static malware analysis but could be applied to other domains where adversarial attacks are based on local perturbation rather than being bounded by a maximum perturbation magnitude. In future work, we will adapt this approach to other security-related data, such as robust classifiers of network packets, robust malware dynamic analysis or robust automatic code summarization.

## References

- [1] Naveed Akhtar, Ajmal Mian, Navid Kardan, and Mubarak Shah. Advances in adversarial attacks and defenses in computer vision: A survey. *IEEE Access*, 9:155161–155196, 2021.
- [2] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.
- [3] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.
- [4] Giovanni Apruzzese, Hyrum S Anderson, Savino Dambra, David Freeman, Fabio Pierazzi, and Kevin A Roundy. ” real attackers don’t compute gradients”: Bridging the gap between adversarial ml research and practice. *arXiv preprint arXiv:2212.14315*, 2022.
- [5] Kshitiz Aryal, Maanank Gupta, and Mahmoud Abdelsalam. A survey on adversarial attacks for malware analysis. *arXiv preprint arXiv:2111.08223*, 2021.
- [6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. Ieee, 2017.
- [7] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. *Advances in neural information processing systems*, 32, 2019.
- [8] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International conference on machine learning*, pages 854–863. PMLR, 2017.
- [9] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.
- [10] Ron Davidson. The fight against malware as a service. *Network Security*, 2021(8):7–11, 2021.
- [11] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. Adversarial examples: Functionality-preserving optimization of adversarial windows malware. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.
- [12] 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.
- [13] 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, 2021.

- [14] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 16(4):711–724, 2017.
- [15] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [16] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015*, 2015.
- [17] Richard Harang and Ethan M. Rudd. Sorel-20m: A large scale benchmark dataset for malicious pe detection, 2020.
- [18] Zhuoqun Huang, Neil G Marchant, Keane Lucas, Lujo Bauer, Olga Ohrimenko, and Benjamin IP Rubinstein. Certified robustness of learning-based static malware detectors. *arXiv preprint arXiv:2302.01757*, 2023.
- [19] Íñigo Íncir 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*, pages 54–63, 2018.
- [20] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 97–117. Springer, 2017.
- [21] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*, pages 533–537. IEEE, 2018.
- [22] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Automating mimicry attacks using static binary analysis. In *USENIX Security Symposium*, volume 14, pages 11–11, 2005.
- [23] Yunus Kucuk and Guanhua Yan. Deceiving portable executable malware classifiers into targeted misclassification with practical adversarial examples. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 341–352, 2020.
- [24] Raphael Labaca-Castro, Luis Muñoz-González, Feargus Pendlebury, Gabi Dreo Rodosek, Fabio Pierazzi, and Lorenzo Cavallaro. Realizable universal adversarial perturbations for malware. *arXiv preprint arXiv:2102.06747*, 2021.
- [25] Xiang Ling, Lingfei Wu, Jiangyu Zhang, Zhenqing Qu, Wei Deng, Xiang Chen, Chunming Wu, Shouling Ji, Tianyue Luo, Jingzheng Wu, et al. Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art. *arXiv preprint arXiv:2112.12310*, 2021.
- [26] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. Certified monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:15427–15438, 2020.
- [27] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.
- [28] Daniel Park, Haidar Khan, and Bülent Yener. Generation & evaluation of adversarial examples for malware obfuscation. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1283–1290. IEEE, 2019.
- [29] Daniel Park and Bülent Yener. A survey on practical adversarial examples for malware classifiers. In *Reversing and Offensive-oriented Trends Symposium*, pages 23–35, 2020.
- [30] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE, 2020.
- [31] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [32] Edward Raff and Charles Nicholas. A survey of machine learning methods and challenges for windows malware classification. *arXiv preprint arXiv:2006.09271*, 2020.
- [33] Ishai Rosenberg, Shai Meir, Jonathan Berrebi, Ilay Gordon, Guillaume Sicard, and Eli Omid David. Generating end-to-end adversarial examples for malware classifiers using explainability. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–10. IEEE, 2020.
- [34] Davor Runje and Sharath M Shankaranarayana. Constrained monotonic neural networks. *arXiv preprint arXiv:2205.11775*, 2022.
- [35] Joseph Sill. Monotonic networks. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10, NIPS '97*, page 661–667, Cambridge, MA, USA, 1998. MIT Press.
- [36] Samuel Henrique Silva and Peyman Najafirad. Opportunities and challenges in deep learning adversarial robustness: A survey. *arXiv preprint arXiv:2007.00753*, 2020.
- [37] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. Mab-malware: A reinforcement learning framework for attacking static malware classifiers. *arXiv preprint arXiv:2003.03100*, 2020.
- [38] Octavian Suciu, Scott E Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14. IEEE, 2019.
- [39] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.
- [40] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Advances in neural information processing systems*, 31, 2018.
- [41] Bao Ngoc Vi, Huu Noi Nguyen, Ngoc Tran Nguyen, and Cao Truong Tran. Adversarial examples against image-based malware classification systems. In *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, pages 1–5. IEEE, 2019.
- [42] R Vinayakumar, Mamoun Alazab, KP Soman, Prabakaran Poor-nachandran, and Sitalakshmi Venkatraman. Robust intelligent malware detection using deep learning. *IEEE access*, 7:46717–46738, 2019.
- [43] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. A close look on n-grams in intrusion detection: anomaly detection vs. classification. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 67–76, 2013.
- [44] Chun Yang, Jinghui Xu, Shuangshuang Liang, Yanna Wu, Yu Wen, Boyang Zhang, and Dan Meng. Deepmal: maliciousness-preserving adversarial instruction learning against static malware detection. *Cybersecurity*, 4(1):1–14, 2021.