

Certified Adversarial Robustness of End-to-End Malware Detectors via (De)Randomized Smoothing

DANIEL GIBERT, Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain

LUCA DEMETRIO, DIBRIS, Università degli Studi di Genova, Genoa, Italy

GIULIO ZIZZO, IBM Research Europe - Ireland, Dublin, Ireland

QUAN LE, CeADAR, University College Dublin, Dublin, Ireland

JORDI PLANES, Computer Science, University of Lleida Polytechnic School, Lleida, Spain

BATTISTA BIGGIO, University of Cagliari, Cagliari, Italy

End-to-end machine learning malware detectors are vulnerable to adversarial EXamples, carefully-crafted malicious programs that evade detection through minimal perturbations. Such attacks typically operate by either replacing unused content (patch attacks), or injecting new patterns (content-injection attacks). To counter these attacks, recent research has focused on certification methods for end-to-end models that aim to prove robustness guarantees within bounded perturbation sizes. However, existing approaches (i) are not robust against content-injection manipulations and (ii) provide only probabilistic guarantees for perturbations that are negligible relative to the overall program size. Hence, in this article we address these limitations through a novel deterministic certification schema based on (de)randomized smoothing. Our defense splits each executable into non-overlapping chunks and classifies them independently. The final decision is obtained via majority voting across all chunks, ensuring that localized modifications, such as injected or patched code, influence only a limited subset of chunks and have minimal impact in the overall classification. This design guarantees that each chunk either contains or does not contain an adversarial perturbation, enabling us to

D. Gibert was supported by grant RYC2023-043607-I funded by MICIU/AEI/10.13039/501100011033 and FSE+. This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under the Marie Skłodowska-Curie Grant Agreement (as per Article 29.4 of the grant agreement). We would like to thank Cormac Doherty and UCD's Centre for Cybersecurity and Cybercrime Investigation for their support. This project has also been partially support by Fondazione di Sardegna under the project "TrustML: Towards ML that Humans Can Trust", CUP: F73C22001320007; and European Union's Horizon Europe research and innovation program under the project ELSA, grant agreement No 101070617; and European Union's Horizon 2020 research and innovation programme under project TESTABLE, grant agreement No 101019206; and SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, and by MCIN/AEI/10.13039/501100011033/FEDER, UE under the project PID2022-139835NB-C22. This research project was made possible through the access granted by the Galician Supercomputing Center (CESGA) to its supercomputing infrastructure. The supercomputer FinisTerae III and its permanent data storage system have been funded by the NextGeneration EU 2021 Recovery, Transformation and Resilience Plan, ICT2021-006904, and also from the Pluri-regional Operational Programme of Spain 2014-2020 of the European Regional Development Fund (ERDF), ICTS-2019-02-CESGA-3, and from the State Programme for the Promotion of Scientific and Technical Research of Excellence of the State Plan for Scientific and Technical Research and Innovation 2013-2016 State subprogramme for scientific and technical infrastructures and equipment of ERDF, CEG15-DE-3114.

Authors' Contact Information: Daniel Gibert, Artificial Intelligence Research Institute (IIIA-CSIC), Bellaterra, Spain; e-mail: daniel.gibert@iiia.csic.es; Luca Demetrio, DIBRIS, Università degli Studi di Genova, Genoa, Italy; e-mail: luca.demetrio@unige.it; Giulio Zizzo, IBM Research Europe - Ireland, Dublin, Leinster, Ireland; e-mail: giulio.zizzo2@ibm.com; Quan Le, CeADAR, University College Dublin, Dublin, Leinster, Ireland; e-mail: quan.le@ucd.ie; Jordi Planes, Computer Science, University of Lleida Polytechnic School, Lleida, Lleida, Spain; e-mail: jordi.planes@udl.cat; Battista Biggio (corresponding author), University of Cagliari, Cagliari, Sardinia, Italy; e-mail: battista.biggio@unica.it.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2471-2566/2026/08-ART36

<https://doi.org/10.1145/3827613>

(i) handle manipulations occurring at arbitrary locations within the program, and (ii) compute deterministic estimates of the perturbation magnitude required to evade detection. We demonstrate the effectiveness of our certification schema through extensive experimental analysis, comparing our defense against a range of state-of-the-art attacks and defenses. The results show that our approach achieves unmatched robustness across all tested attack scenarios, substantially outperforming competing defenses.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**; • **Computing methodologies** → *Machine learning*; • **Applied computing** → *Computer forensics*;

Additional Key Words and Phrases: Malware detection, adversarial machine learning, adversarial defenses, adversarial malware, evasion attacks

ACM Reference Format:

Daniel Gibert, Luca Demetrio, Giulio Zizzo, Quan Le, Jordi Planes, and Battista Biggio. 2026. Certified Adversarial Robustness of End-to-End Malware Detectors via (De)Randomized Smoothing. *ACM Trans. Priv. Sec.* 29, 3, Article 36 (August 2026), 30 pages. <https://doi.org/10.1145/3827613>

1 Introduction

During the past decade, we have witnessed a significant rise in cyberattacks targeting Windows machines through the use of malware, malicious programs designed to compromise or damage end-user devices. According to the **European Union Agency for Cybersecurity (ENISA)**,¹ companies are continuously targeted by cyber attacks, with ransomware incidents surging in the first half of 2023. To keep up with malware, and to overcome the prior pitfalls of traditional antivirus engines in detecting zero-day malware, researchers and cybersecurity companies have explored **machine learning (ML)** techniques to complement their solutions, which are well-suited for processing large volumes of data [9]. While feature-based detectors [1, 2] are still prominent and are the most common type of malware detectors, in recent years there has been a growing shift toward end-to-end detectors [10, 16, 23, 24], which infer maliciousness directly from raw input programs without any preprocessing. This approach eliminates the need for manual feature extraction, as end-to-end models are trained directly on the raw bytes composing each program, while simultaneously benefiting from the hardware acceleration provided by GPUs.

While promising, deep learning models have been shown to be vulnerable to adversarial EXEmples, malicious software specifically crafted to deceive such models. These samples are generated through optimization algorithms [5, 6, 14, 28, 30, 31], introducing minimal perturbations to test-time samples while preserving the original functionality of the program. These perturbations typically take the form of (i) *patch* attacks, which alter unused portions of existing code, and (ii) *content-injection*, which insert new content by exploiting ambiguities in executable files format.

Countermeasures against these attacks are beginning to emerge within the research community, often drawing inspiration from less constrained domains such as image classification. In particular, attention has been drawn by *certification methods*, which train models on multiple versions of input samples, using majority voting on all predictions on each of these to compute the final decision [3, 19]. Depending on how these versions are computed, certification methods can bring guarantees on the nonexistence of attacks in proximity of input samples, determining in advance how much perturbation is needed by an attacker to flip such decision. Specifically, *randomized smoothing* techniques [12, 13], which produce variants of input samples by perturbing them with Gaussian noise, have been ported to the malware domain to counter adversarial EXEmples. These techniques, however, still encounter issues when dealing with evasion attacks. In particular,

¹<https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023>

certifiable malware detectors based on randomized smoothing fail to certify robustness against perturbations larger than 1% of the file's size, a negligible amount of content w.r.t. the size of samples, thus resulting weak against most content-injection techniques [13]. By looking at the developed techniques in the state for the art, most content-injection attacks [5] leverage several kilobytes of insertion, already exceeding the bounds imposed by randomized smoothing defenses.

Hence, in this article, we address these limitations by introducing a novel certification schema based on (de)randomized smoothing to certify the robustness of end-to-end malware detectors against adversarial EXEmples leveraging patch or content-injection manipulations, regardless of the size of the perturbations. Our approach splits each input program into contiguous, non-overlapping chunks of bytes, which are classified independently. The splitting process is guided by domain knowledge, leveraging information from the file headers to identify the bytes corresponding to headers and sections, thereby aligning them with the chunk boundaries. The final classification is obtained through majority voting, which mitigates the impact of injected adversarial content. This splitting mechanism is crucial to minimizing the influence of such manipulations, as each chunk is evaluated independently. Consequently, attacks that patch or inject content influence only a small subset of chunks, thereby limiting their ability to alter the classifier's overall decision. Thanks to this splitting mechanism, we can derive a deterministic robustness certificate for our classifier against different types of attacks, provided that the number of predictions for the correct class exceeds those for any other class by a sufficient margin. This certificate provides guarantees that the model's output will remain consistent within a defined range of perturbations, which is an essential property for maintaining reliable classification performance in the presence of adversarial EXEmples.

Owing to its generality, our technique is architecture-agnostic, meaning that it can be applied on top of any end-to-end malware detection architecture [10, 16, 23, 24], without requiring the generation of any adversarial EXEmples during training, thus significantly reducing computational costs. Even without seeing adversarial examples during training, attacks [6, 14, 28, 30, 31] that manipulate or inject a few hundreds or thousands of bytes cannot bypass detection as they will only be able to manipulate a small portion of the total number of chunks. This is in stark contrast to undefended [10, 16, 23, 24] and randomized smoothing-based models [12, 13], where small payloads of hundreds or thousands of bytes are enough to evade the models.

In this article, we build upon our previous work [11] by:

- Generalizing the (de)randomized smoothing defense for malware detection into a deterministic certification framework for any functionality-preserving adversarial manipulation, i.e., patch and injection, regardless of their location of application and size. We extended (de)randomized smoothing conceptually to handle a broader class of content-injection attacks, not just append attacks.
- Introducing a domain-informed chunking, where instead of splitting at arbitrary byte boundaries we align chunks with PE file headers and sections to minimize the influence of the attacks.
- Providing theoretical robustness certificates against multiple, arbitrarily located patch and content-injection manipulations.
- Rigorously evaluating our defense across four diverse neural network architectures, namely MalConv [23], MalConvGCT [24], AvastConv [16], and ShallowConv [10].
- Performing an extensive ablation study that compares our certified defense with randomized smoothing-based defenses [12, 13, 26] against a wide range of adversarial attacks [5, 6, 14, 28, 30], establishing a new benchmark in terms of adversarial and certified accuracy against adversarial EXEmples.

- Constructing an adaptive, multi-location attack that splits an adversarial payload across several PE regions: space between headers and the first section, slack space at the end of existing sections, newly created sections, and the file overlay.
- Showcasing how the proposed (de)randomized smoothing scheme can facilitate a finer-grained analysis of the file, by identifying specific chunks within a file that exhibit malicious traits.

The rest of the article is organized as follows. Section 2 provides a review of the literature on end-to-end malware detection, including attacks and defenses. Section 3 presents our certified deterministic defense based on (de)randomized smoothing, and Section 4 evaluates the proposed defense mechanism against state-of-the-art evasion attacks. Finally, Section 5 summarizes our concluding remarks and presents some future lines of research.

2 Background and Related Work

Before detailing our methodology, we first introduce the core concepts needed to better understand this article, along with a review of the literature on end-to-end malware detection, and the attacks and defenses specifically proposed to bypass and harden end-to-end malware detectors, respectively.

2.1 Windows PE File Format

The **Portable Executable (PE)** format is the standard file format used for executables, object code, and DLLs in Windows systems.² It is organized as a series of structure components, beginning with the MS-DOS header and PE signature, which ensure compatibility and identify the file type. These are followed by the COFF header and the Optional header, which contain critical metadata such as target architecture, entry point address, and memory allocation. The Section Table comes next, describing the layout and attributes of each section (e.g., .text, .data, and .rdata). Sections are the actual regions in the binary that hold code, data, and resources.

2.2 End-to-End Malware Detection

Due to the unprecedented success of deep learning in domains such as computer vision and natural language processing, malware detection has increasingly moved away from feature-based methods toward end-to-end learning. Unlike feature-based detectors, which manually extract features from computer programs based on domain knowledge, end-to-end malware detectors [10, 16, 23, 24], take as input the raw bytes representing a computer program. Because PE files vary in size from a few kilobytes to several megabytes, end-to-end detectors are typically implemented using **Convolutional Neural Networks (CNNs)**, which effectively extract hierarchical features from spatial data while maintaining computational efficiency. PE files can be thought of as a one-dimensional spatial structure, where both the order and location of the bytes are important. CNNs excel at identifying local patterns, such as byte sequences, and, through multiple convolutional layers, can capture high-level patterns, including instruction sequences or structural patterns that distinguish between goodware and malware.

2.3 Adversarial EXEmPles

Windows malware detectors implemented with ML have been shown to be vulnerable to *adversarial EXEmPles*, carefully-manipulated programs that evade detection. These are constructed by bending the PE file format to the attackers' will, by applying manipulations to its structure that preserve the original functionalities and fool the target model. The attacks in the literature [5, 6, 14, 28, 30], illustrated in Figure 1, can be divided into (1) patch attacks and (2) content-injection attacks. Patch

²<https://learn.microsoft.com/windows/win32/debug/pe-format>

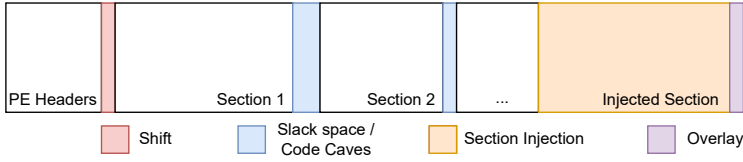


Fig. 1. Graphical representation of the locations perturbed by different attack strategies.

attacks [6, 28] refer to attacks that manipulate and replace existing bytes within a given executable file. These can be staged in unused locations inside programs like unused headers, or the unused bytes between sections known as slack space attack [6, 17, 28]. Content-injection attacks [5, 6, 14, 30] refer to attacks that insert new content somewhere within the executable file. These attacks include (i) appending content at the end of programs, known as padding or overlay attack [14, 28]; (ii) inserting new sections filled with content harvested from legitimate programs (known as GAMMA attack) [5]; (iii) insert content between the Optional Header and code (known as Shift attack) [6], and (iv) insert content between each section (known as code caves attack) [30].

2.4 Defenses against Adversarial EXEmPles

In the context of defenses, several approaches have been proposed to reduce or nullify the impact of adversarial EXEmPles at test time based on certification methods. These are designed to provide statistical guarantees on the robustness of the trained model. In this context, there are two competing methods based on randomized smoothing that can be leveraged to reach this result.

Randomized smoothing [3, 18, 20, 27] works by adding random noise to input data during both training and inference, thus making predictions stable and resilient to small perturbations. At training time, a base classifier f is trained on samples perturbed with noise $\tilde{x} = x + \epsilon$, where ϵ is gaussian noise and x is the original sample. At inference time, $\tilde{x}^{(1)}, \dots, \tilde{x}^{(n)}$ noisy versions of an input example x are generated and the final classification is determined as majority voting of all predictions. This process cannot be directly applied to malware detection since programs are not stored inside a continuous space, hence additive manipulations are not producing samples that are close in terms of behavior. Hence, many works investigated how to port randomized smoothing in this domain [12, 13], as illustrated in Table 1. There are two techniques that create variants of the same input program: (i) the *ablation-based* approach, proposed by Gibert et al. [12], which randomly replaces each byte with a special one encoded as 256; and (ii) the *deletion-based* approach, known as RS-Del, introduced by Huang et al. [13], which generates new samples by randomly deleting bytes from the original program.³

However, while effective against attacks that alter small amounts of bytes [3, 18, 20], both ablation-based and deletion-based techniques fail to certify robustness against perturbations larger than 1% of the file's size [13], which is a negligible percentage of content compared to the perturbation sizes caused by state-of-the-art attacks. Hence, randomized smoothing techniques have been shown

³Beyond Windows end-to-end malware detection, randomized smoothing-based certification has also been explored in other malware detection settings. Erdemir et al. [7] adapted randomized smoothing to feature-based malware detectors, deriving robustness certificates for classifiers that operate on manually extracted feature representations rather than raw byte sequences. Gao & Yan [8] proposed a certified defense for PDF malware detectors. However, these works operate in fundamentally different settings from ours: feature-based detectors are not subject to the same byte-level manipulation constraints as end-to-end detectors, and the PDF file format presents a distinct set of structural properties and adversarial manipulation strategies compared to Windows PE executables. As a result, the certification challenges and threat models addressed by these works are not directly comparable to ours, and their methods do not straightforwardly transfer to the end-to-end Windows malware detection setting we consider.

Table 1. (a) Original Piece of Windows Executable, and (b) How it is Altered with Ablation-Based [12] and (c) Deletion-Based [13] Smoothing Strategies

(a)		(b)		(c)	
File offset	Byte	File offset	Byte	File offset	Byte
00000000	77	00000000	77	00000000	77
00000001	90	00000001	256	00000001	-
00000002	144	00000002	144	00000002	144
...	...	...	...	...	...
00000400	85	00000400	85	00000400	85
00000401	139	00000401	256	00000401	-
00000402	236	00000402	256	00000402	-
00000403	131	00000403	131	00000403	131
00000404	236	00000404	236	00000404	236
00000405	92	00000405	256	00000405	-

ineffective in defending against content-injection attacks [5, 6, 28], as they include hundreds of kilobytes into malware to evade detection. Furthermore, since adversarial examples are generated by exploiting specific locations within the PE format, applying random manipulations may overlook these carefully placed adversarial payloads, thereby reducing robustness.

To circumvent the limitations of randomized smoothing, in [11] we proposed a deterministic strategy specifically designed to defend against adversarial EXEmples in the malware domain. The proposed defense splits executables into non-overlapping chunks, classifies each chunk independently using a base malware detector, and aggregates the predictions through majority voting to obtain the final decision. This defense is effective because adversarial manipulations are typically localized within a limited region of the executable and therefore influence only a subset of chunks. Since the final decision is obtained through majority voting, an attacker must modify a sufficiently large number of chunks to change the classifier's output. A related approach was also proposed in [26], which leverages a similar splitting strategy. In [11], the byte sequence is divided into fixed-size chunks (e.g., 512 bytes), whereas [26] partitions the executable into K chunks, resulting in chunk sizes that vary across samples. Furthermore, [11] derives robustness guarantees against both patch and append attacks, while [26] provides certification guarantees only for patch attacks.

In this work, we extend the idea introduced in [11] into a more general certification framework that enables deterministic robustness guarantees against patch and content-injection attacks in end-to-end malware detectors.

3 (De)Randomized Smoothing for End-to-End Malware Detection

Building on our previous work [11], we introduce a refined (de)randomized smoothing framework that leverages domain knowledge about the structure of executable files to derive formal robustness guarantees against patch and content-injection attacks. In particular, our approach is designed to limit the influence of localized adversarial manipulations by decomposing executables into smaller chunks that can be analyzed independently. This design allows us to isolate the impact of adversarial perturbations and bound their impact on the final prediction of the classifier.

The idea behind our approach, illustrated in Figure 2, is to split an input program into contiguous, non-overlapping chunks of bytes, classifying each chunk independently and aggregating the results. This way, if an attacker modifies or injects adversarial content into certain parts of the input, the changes will be confined to specific chunks rather than influencing the entire input. By aggregating all predictions through a majority vote to determine the model's final decision, we mitigate the

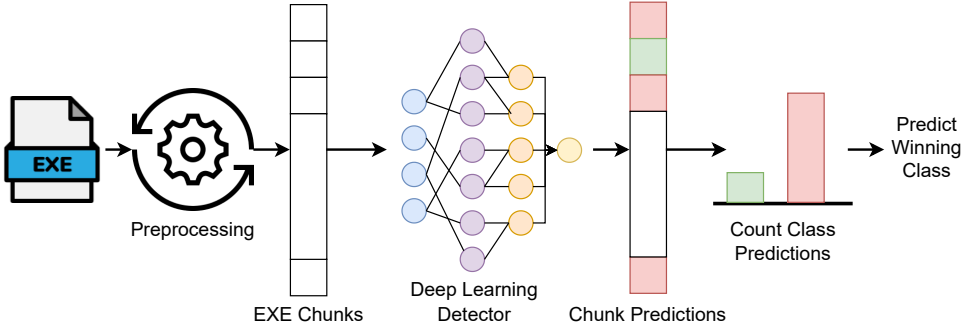


Fig. 2. An overview of the (de)randomized smoothing scheme. At inference time, a given input example x is preprocessed, as described in Algorithm 3, and split into consecutive, non-overlapping chunks. Each chunk is then independently classified by a base malware detector. Chunks predicted as malicious are represented in red, whereas chunks predicted as benign are shown in green. Afterwards, the final classification is calculated using the majority voting mechanism defined in Equation (2).

impact of adversarial modifications as they will only influence a small minority of chunks. Moreover, we leverage domain knowledge to design the chunking process, ensuring that the components of the input file, such as headers and sections, are processed separately. This prevents adversarial payloads from influencing more than a single component at a time and thus, ensuring that no chunk contains information from multiple components.

By leveraging this chunking mechanism, we can derive deterministic robustness certificates for our classifier under different types of attacks if the number of predictions for the correct class exceeds the other one by a large enough margin. These certificates guarantee that the overall classification will remain unchanged even if an adversary manipulates certain chunks.

3.1 Implementation Details

Our (de)randomized smoothing defense operates differently during training and inference. The implementation details of each stage are described below.

Training phase. In our (de)randomized smoothing scheme, a base malware classifier f is trained to make classifications based on chunks of bytes $\tilde{x}$ of a given executable file x . This chunk of bytes $\tilde{x}$ consists of only a subset of contiguous bytes of size z randomly selected from the original file x . The rest of the file is not used for classification. This functionality is implemented by the operation $\text{CHUNK_EXTRACTION}(x, z)$. The training procedure, defined in Algorithm 1, involves training a base malware classifier with small chunks of bytes of the files in the training dataset, and not with the whole byte sequences. Note that, during training, the starting locations of the chunks of bytes are randomly selected. For a description of the operation $\text{PREPROCESS}(x)$ we refer readers to Section 3.2.2. In our work, instead of ablating the bytes with a NULL value like in Levine et al. [19], we remove the bytes outside the selected chunk. Removing the bytes ensures that no artificial placeholders are introduced and reduces the input size of the classifier to only that of the selected chunk, resulting in greater computational efficiency while maintaining similar detection performance.

Inference phase. During inference, an input example x is split into consecutive, non-overlapping chunks of size z starting from the byte located at position zero, i.e., $x = \{\tilde{x}^{(0)}, \tilde{x}^{(1)}, \tilde{x}^{(2)}, \dots, \tilde{x}^{(N)}\}$, where $\tilde{x}^{(j)}$ is a chunk of bytes from x starting and ending at byte $j * z$ and $(j + 1) * z$, respectively.

ALGORITHM 1: (De)randomized Smoothing-Based Classifier's Training Procedure.

Require: training dataset D_{train} , malware detector f with parameters θ , chunk size $z \in \mathbb{N}_+$, learning rate γ
Initialize parameters θ
for each epoch i from 1 to MAX_EPOCHS **do**
 for each training example (x, y) in D_{train} **do**
 $\hat{x} \leftarrow \text{PREPROCESS}(x, z)$
 $\tilde{x} \leftarrow \text{CHUNK_EXTRACTION}(\hat{x}, z)$
 $\theta \leftarrow \theta - \gamma \nabla L(f(\tilde{x}), y)$ ▷ Update parameters with gradient descent strategy of choice
 end for
end for

Afterwards, each chunk $\tilde{x}^{(j)}$ is independently classified by the base classifier f . In other words, $f(\tilde{x}^{(j)})$ is the base classification, where the classifier takes as input the chunk of bytes $\tilde{x}^{(j)}$. For the task of malware detection, $C = \{c_1, c_2\}$ is the set of all possible classes, where c_1 and c_2 correspond to the benign and malicious classes, respectively. For each class $c \in C$, $f_c(\tilde{x}^{(j)})$ will be either 0 or 1. To make the final classification, we count the number of chunks the base classifier returns for each class:

$$\forall c \in C, n_c(x) := \sum_{j=0}^N f_c(\tilde{x}^{(j)}). \quad (1)$$

The final smoothed classification is the plurality class returned:

$$g(x) := \operatorname{argmax}_c n_c(x). \quad (2)$$

In the case of ties, we deterministically return the larger-indexed class, which corresponds to the malicious class. The probability $P(c|x)$ of an input example x belonging to class c can be estimated as:

$$P(c|x) = \frac{n_c(x)}{N}, \quad (3)$$

where $n_c(x)$ is the number of chunks of x that belong to class c according to the base classifier f and N is the total number of chunks.

Remark. Our approach is architecture-agnostic, meaning that any end-to-end model [10, 16, 23, 24] can be employed as base classifier. In Section 4, we demonstrate this versatility by evaluating four distinct architectures. Furthermore, our approach can be extended to multi-class classification tasks, such as classifying malware into families. In this setting, the cardinality $|C|$ corresponds to the number of families. In the context of malware classification, if a tie occurs, the function f would return either a predetermined class or the class with the smallest index. For malware classification, it is not necessary to require that $f_c(\tilde{x}^{(j)}) = 1$ for any specific class c , as the base classifier may abstain, returning 0 for all classes when none exceeds a predefined probability threshold, or may output 1 for multiple classes. We note that this functionality has not been implemented, as our current work focuses solely on the malware detection task.

3.2 Robustness Guarantees of (De)Randomized Smoothing

The proposed defense strategy allows us to provide deterministic robustness certificates against content manipulation attacks specifically designed to evade end-to-end malware detectors. A deterministic robustness certificate is a formal, verifiable guarantee that the malware classifier will maintain correct predictions under specific bounded perturbations of its input. These certificates are critical for addressing the vulnerability of the end-to-end malware detectors to adversarial attacks,

where an attacker manipulates a malicious executable to evade detection. As far as we are aware, no other deterministic certified defense has been proposed for end-to-end malware detection.

Based on the number of chunks classified for each class (benign or malicious), we follow a similar reasoning to Levine et al. [19] to obtain a deterministic robustness certificate for our classifier under different types of attacks (patch and content-injection attacks) if the number of predictions for the correct class exceeds the second most commonly predicted class by a large enough margin. Depending on the type of attack, the margin between the number of predictions for the correct class and the other will differ.

Assuming that an input example x has size L and a chunk has size z , our (de)randomized smoothing strategy splits the x into $\lceil \frac{L}{z} \rceil$ non-overlapping chunks of size z , except for the last chunk, which might be smaller and will be padded accordingly. With our (de)randomized smoothing approach it is tractable to use the base classifier f to classify all chunks of bytes extracted from an input example x , allowing us to exactly compute the smoothed classifier $g(x)$, yielding deterministic certificates. This is in stark contrast to randomized smoothing-based defenses [12, 13], which approximate the smoothed classifier $g(x)$ with Monte Carlo sampling. Monte Carlo sampling estimates $g(x)$ based on a subset of possible ablations, rather than exhaustively evaluating all of them.

3.2.1 Patch Attacks. In the context of malware detection, a patch attack involves manipulating and replacing existing bytes within a given executable without inserting new bytes or removing existing bytes. If the attacker perturbs a byte file with an adversarial patch of size $p > 1$, they can intersect with at most $\Delta = \lceil \frac{p}{z} \rceil + 1$ chunks, and thus, it can only alter the prediction of Δ chunks. This yields the following guarantee:

THEOREM 1. *For any input example x , base classifier f , (de)randomized smoothing classifier g , chunk size z , and patch size p , such that $\Delta = \lceil \frac{p}{z} \rceil + 1$, if:*

$$n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + 2\Delta, \quad (4)$$

where $n_{c'}(x)$ and $n_{c''}(x)$ denote the most commonly and second most commonly predicted classes for an input example x , then for any adversarial EXExample x' which differs from x only in a patch of size p , $g(x') = c'$.

In Theorem 1, the indicator function term ($1_{c' < c''}$) is present because we break ties deterministically by label index during the final classification. Proofs are provided in Appendix 5.1.

Since the problem of malware detection is a binary classification problem, the formulation for predicting whether an input file is malware can be rewritten as follows :

$$n_m(x) \geq n_b(x) + 2\Delta, \quad (5)$$

where $n_m(x)$ and $n_b(x)$ refer to the number of chunks predicted as malware and as benign by the (de)randomized smoothed classifier.

If, instead of perturbing a single adversarial patch, the attacker perturbs a byte file with K adversarial patches of sizes $p_1, p_2, \dots, p_K$, then Δ can be expressed as:

$$\Delta = \sum_{i=1}^K \Delta_i = \sum_{i=1}^K \left(\left\lceil \frac{p_i}{z} \right\rceil + 1 \right), \quad (6)$$

where Δ_i is the number of chunks the i th adversarial patch can intersect with and p_i is the size of the i th adversarial patch. Generally speaking, the decision of the classifier will only change depending on the amount of chunks that the attacker modifies. If the number of chunks that the attacker modifies is not large enough, then we can certify that the prediction of our classifier will not change. We summarize the certification procedure for a single example in Algorithm 2.

ALGORITHM 2: Certify Whether Patch (or Injection) Attack Can Change the Predicted Class y of a Malicious Example x

Require: Example x with size L , label y , end-to-end malware detector f , chunk size z

```

 $\hat{x} \leftarrow \text{PREPROCESS}(x, z)$  ▷ Preprocess the input example
for each chunk starting position  $j$  from 0 to  $\lceil \frac{L}{z} \rceil$  do
     $\tilde{x}^{(j)} \leftarrow (x_{z*j}, \dots, x_{z*(j+1)})$ 
     $\tilde{y}^{(j)} \leftarrow f(\tilde{x}^{(j)})$  ▷ Predicted class of chunk  $\tilde{x}^{(j)}$ 
end for
 $\tilde{y}', \tilde{y}'' \leftarrow \text{majority and second majority of } \{\tilde{y}^{(j)}\}_{j=1}^n$ 
 $n_{c'}(x) \leftarrow \text{number of chunks classified as } \tilde{y}'$ 
 $n_{c''}(x) \leftarrow \text{number of chunks classified as } \tilde{y}''$ 
if  $n_{c'}(x) \geq n_{c''}(x) + 2\Delta$  then
    return  $x$  is certified against patch attacks
end if
if  $n_{c'}(x) \geq n_{c''}(x) + \Delta$  then
    return  $x$  is certified against injection attacks
end if
return  $x$  is not certified

```

3.2.2 Content-injection Attacks. In the context of malware detection, content-injection attacks refer to attacks that introduce new content into an executable. These attacks can be divided into two: (1) append attacks, and (2) insertion attacks. On the one hand, append attacks involve adding extra bytes or the adversarial payload at the end of a given executable, known as its overlay. This manipulation has long been used by malware authors to hide malicious code or data within seemingly benign files. On the other hand, insertion attacks insert the adversarial payload somewhere within the executable file. This can be done by injecting the adversarial payload into newly-created sections [5], creating space between the headers and the sections [6], or extending the space between sections [30].

Similarly to patch attacks, if the attacker perturbs the file by appending an injected adversarial payload of size q , the maximum number of chunks from the adversarial EXEmple x' that will differ from x will be $\Delta = \lceil \frac{q}{z} \rceil + 1$ chunks. This yields the following guarantee:

THEOREM 2. For any input example x , base classifier f , (de)randomized smoothing classifier g , chunk size z and an injected adversarial payload of size q , such as $\Delta = \lceil \frac{q}{z} \rceil + 1$, if:

$$n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + \Delta, \quad (7)$$

where $n_{c'}(x)$ and $n_{c''}(x)$ denote the most commonly and second most commonly predicted classes for an input example x , then for any adversarial EXEmple x' which differs from x only in an adversarial payload of size q , $g(x') = c'$.

This can be rewritten as follows:

$$n_m(x) \geq n_b(x) + \Delta. \quad (8)$$

Note that for append attacks the margin between the number of predictions for the correct class and the others does not need to be as large as in patch attacks. This is because, during a patch attack, the attacker patches the original content of an executable and, thus, a patch attack can flip the prediction of existing chunks. To this end, the margin between the predictions of the class and the others must be at least 2Δ . Otherwise, we cannot certify that the EXEmple has not been tampered with.

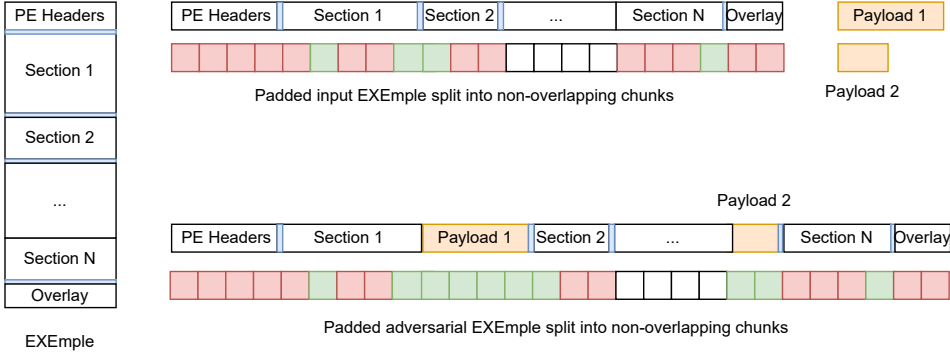


Fig. 3. An illustration of the PREPROCESS operation. The PREPROCESS operation pads the headers and sections of PE files to a multiple of the chunk size z used to split the executable into non-overlapping chunks. By doing so, we ensure that the predictions for the chunks corresponding to the headers and sections not manipulated by the attacker in both the original and adversarial EXEs remain consistent, allowing us to extend the certification procedure to content-injection attacks.

Example: Suppose a malicious example is divided into 10 chunks, and the malicious class currently has 6 votes while the benign class has 4. If an attacker alters the prediction of two chunks, resulting in 4 malicious votes versus 6 benign votes, it would flip final classification. If the margin had been larger—for instance, 8 malicious votes versus 2 benign votes—flipping two chunks would not have been enough to change the final classification (6 malicious votes versus 4 benign votes).

In contrast, Theorem 2 addresses scenarios where new content is inserted at the end of an executable rather than manipulating existing content. Therefore, the margin required for certifying the robustness against these attacks is less demanding, i.e., Δ instead of 2Δ , as new content will not flip the prediction of existing chunks, except at most one chunk (i.e., the chunk that intersects with both the original and the inserted content). The only case where no predictions from existing chunks are influenced during an append attack is when the chunks of the original example line up exactly with a multiple of the chunk size.

Example: Suppose a malicious example is divided into 10 chunks, and the malicious and benign classes have 8 and 2 votes, respectively. If an attacker adds new content, that is, the size of 2 chunks, it would not flip the final classification from malicious to benign. In the worst-case scenario, this new content will be classified as benign, adding 2 more benign votes. However, if this added content overlaps with the last chunk of the original file, it could flip the prediction for that chunk, leading to 3 benign votes (7 malicious votes and 5 benign votes). To change the final classification, the attacker would need to inject an adversarial payload, that is, the size of 5 chunks as this would result in 7 malicious votes and 8 benign votes.

Unfortunately, the certification procedure employed for append attacks does not directly translate to insertion attacks. The reason behind this disparity is that insertion attacks introduce new content within the executable and not at the end, disrupting the file's original structure. Consequently, the content within the original file can be displaced, leading to discrepancies in the content of the chunks from the original and adversarial EXEs, as seen in Figure 3.

To address this challenge, we introduce a preprocessing step, denoted PREPROCESS. The preprocessing step ensures that the chunk partitioning of the original and adversarial EXEs remain consistent when adversarial content is inserted into the file. Without preprocessing, inserting bytes inside an executable would shift the subsequent content of the file. This displacement would

ALGORITHM 3: PREPROCESS Step. The PAD Operation Pads a Given Sequence of Bytes to a Multiple of z .

Require: Example x , chunk size z

$h \leftarrow \text{read_headers_content}(x)$

$\hat{h} \leftarrow \text{PAD}(h, z)$

▷ Pad the headers to a multiple of z

$S \leftarrow \text{read_sections_content}(x)$

▷ $S = \{s_0, s_1, \dots, s_n\}$ where n is the number of sections in x

$\hat{S} \leftarrow []$

▷ Initialize empty list of sections

for each section s in S **do**

$\hat{s} \leftarrow \text{PAD}(s, z)$

▷ Pad section s to a multiple of z

 Append the padded section $\hat{s}$ to $\hat{S}$

end for

$\hat{x} = \text{build_PE_file}(\hat{h}, \hat{S})$

▷ Rebuild PE file with the padded headers $\hat{h}$ and sections $\hat{S}$

return $\hat{x}$

alter the boundaries of the chunks used by the classifier, causing many chunks in the adversarial EXEmple to contain different byte sequences than their counterparts in the original file, even if the attacker only inserted a small payload. As a result, the attacker could indirectly influence the predictions of a large number of chunks, which would violate the assumptions required to derive our robustness guarantees.

To address this issue, the PREPROCESS operation pads the headers and each section of the executable with zero bytes so that their sizes become multiples of the chunk size z . This alignment ensures that each structural component (i.e., headers and sections) of the executable is confined to chunk boundaries. Under this transformation, inserting an adversarial payload does not shift the content of the remaining parts of the executable. Consequently, the predictions of chunks that do not intersect with the injected payload remain unchanged, as illustrated in Figure 3.

From the perspective of the threat model, this preprocessing step constrains the impact of an insertion attack to a bounded number of chunks, i.e., at most the chunk intersecting with the beginning of the payload and the chunks containing the injected content. This property is crucial for our certification procedure, as it allows us to precisely bound the number of chunk predictions that an attacker can influence. Therefore, the preprocessing step enables us to extend the robustness guarantees derived for append attacks (Theorem 2 and Equation (8)) to insertion attacks while maintaining the validity of the certification bounds. Furthermore, if the attacker injects M adversarial payloads with sizes $q_1, q_2, \dots, q_M$, then Δ can be expressed as:

$$\Delta = \sum_{j=1}^M \Delta_j = \sum_{i=j}^M \left(\left\lceil \frac{q_j}{z} \right\rceil + 1 \right). \quad (9)$$

3.2.3 Combined Patch and Content-Injection Attacks. We now consider the general case where an attacker simultaneously applies patch and content-injection manipulations. This scenario subsumes the two attack types analyzed in Sections 3.2.1 and 3.2.2, and directly models adaptive adversaries that exploit multiple modification strategies such as the all-in-one attack described in Appendix 5.1.

Formally, we assume that the attacker applies K adversarial patches of sizes $p_1, p_2, \dots, p_K$ and M adversarial injected payloads of sizes $q_1, q_2, \dots, q_K$ simultaneously, where each patch and injection payload affects a disjoint set of bytes within the executable. This disjointness assumption is naturally satisfied by functionality-preserving attacks, as overlapping a patch with an injected payload would require modifying the injected content itself, which is redundant and provides no additional adversarial advantage. Under these conditions, the total number of chunks whose predictions can

be influenced by the attacker is bounded as follows. By the reasoning established in Theorem 1, each patch of size p_i can intersect with at most $\Delta_i^{patch} = \left\lceil \frac{p_i}{z} \right\rceil + 1$ chunks, and since a patch replaces existing bytes, it can alter the predictions of those chunks in either direction, i.e., from malicious to benign or vice versa. Therefore, each patch contributes a factor of $2\Delta_i^{patch}$ to the required majority-vote margin. By the reasoning established in Theorem 2, each injected payload of size q_i , after the PREPROCESS alignment step, introduces at most $\Delta_j^{inject} = \left\lceil \frac{q_i}{z} \right\rceil + 1$ new or boundary-affected chunks. Since injected content does not displace or alter the byte content of existing chunks, each injection contributes a factor of Δ_j^{inject} to the required margin. Because the patch and injection regions are disjoint by assumption, their effect on chunk prediction counts are independent, and the total adversarial influence is bounded by their sum:

$$\Delta_{combined} = 2 \sum_{i=1}^K \left(\left\lceil \frac{p_i}{z} \right\rceil + 1 \right) + \sum_{j=1}^M \left(\left\lceil \frac{q_j}{z} \right\rceil + 1 \right). \quad (10)$$

This yields the following combined robustness guarantee:

THEOREM 3. *For any input example x , base classifier f , (de)randomized smoothing classifier g , chunk size z , K adversarial patches $p_1, \dots, p_K$, and M adversarial injected payloads of sizes $q_1, \dots, q_M$ applied at disjoint locations, such that $\Delta_{combined}$ is defined as in Equation (10), if:*

$$n_{c'}(x) \geq \max_{c'' \neq c'} \left[n_{c''}(x) + 1_{c' < c''} \right] + \Delta_{combined}, \quad (11)$$

where $n_{c'}(x)$ and $n_{c''}(x)$ denote the most commonly and second most commonly predicted classes for an input example x , then for any adversarial EXExample x' obtained from x by simultaneously applying all K patches and M injections, $g(x') = c'$.

For a binary malware detection task, Equation 9 can be rewritten as:

$$n_m(x) \geq n_b(x) + \Delta_{combined}, \quad (12)$$

where $n_m(x)$ and $n_b(x)$ denote the number of chunks predicted as malicious and benign, respectively.

4 Evaluation

Following we provide a comprehensive description of the experimental setup used to evaluate our defense. We begin by specifying the data, end-to-end models, and hardware used for our experiments. Next, we present the experiments performed on clean and adversarial data to test the robustness of the proposed defense against a variety of adversarial attacks.

4.1 Experimental Setup

The experiments were run on a machine with an Intel Core i7-7700k CPU, 1x Geforce GTX1080Ti GPU, and 64 GB RAM. The code has been implemented with PyTorch [22].⁴ We now detail the data and end-to-end models we have used in our experiments.

4.1.1 BODMAS Dataset. The BODMAS dataset [29] comprises a total of 57,293 malware samples, including information on 581 malware families, along with 77,142 benign Windows PE files collected between August 2019 and September 2020. The dataset has been partitioned into three sets: training, validation and test sets, based on the timestamps associated with each sample. Consequently, the training set contains older executables, while the test set comprises the most recent ones.

⁴All the code will be available at this link after acceptance: https://github.com/danielgibert/certified_robustness_of_end_to_end_malware_detectors

Since the end-to-end architectures considered in our study operate on fixed-length byte sequences (e.g., MalConv operates with an input length of 1,000,000 bytes, and both AvastConv and ShallowConv with 512,000 bytes), we extended the input size of AvastConv and ShallowConv to 1,000,000 bytes to ensure a fair comparison across all models. Consequently, we constrained our analysis to executables that are less than or equal to 1 megabyte in size, allowing all architectures to process inputs of identical length without the need to truncate or discard significant portions of the binaries. At test time, however, the input length of all models was extended to 2 MB to accommodate adversarial examples whose size could increase by up to 100% of the original file, ensuring that perturbed samples were fully processed without truncation.⁵

This constraint is further motivated by computational considerations. Randomized smoothing methods [13] are inherently demanding, as they require generating $N = 4000$ noisy version of each input sample to estimate class probabilities and compute probabilistic robustness certificates. Furthermore, processing and manipulating large binaries is computationally expensive, especially when optimizing the adversarial payloads using genetic algorithms [5, 30]. Importantly, this restriction does not compromise the validity of our analysis. Prior studies [4] have found that CNNs for malware detection learn the majority of features from the PE headers and not the data and code sections, indicating that extending the input length beyond this region does not yield significant performance improvements [25]. Moreover, the file sizes in our dataset are consistent with those used in previous works [5, 6, 10, 12–14, 16, 23, 28, 30, 31], ensuring comparability with established literature. The resulting reduced dataset comprises 39,360 benign and 37,742 malicious executables.⁶

4.1.2 End-to-End Malware Detectors. In the context of malware detection, several CNN architectures have been proposed to develop end-to-end detectors, i.e., models that learn to classify executable files directly from raw byte sequences:

- *MalConv* [23]. This architecture consists of an embedding layer, a gated convolutional layer with 100 filters of size 512, a global max-pooling layer, and a fully connected layer.
- *MalConvGCT* [24]. This architecture consists of two sub-networks, a context extractor and a feature extractor sub-network. The context extractor sub-network learns global context from the input examples. The feature extractor sub-network performs initial feature extraction. Following, a global channel gating layer is used to selectively suppress regions of the input. This is followed by temporal max-pooling, a fully connected layer, and a softmax layer.
- *AvastConv* [16]. This architecture comprises four convolutional layers with kernel sizes equal to 32, 32, 16, and 16, respectively. These convolutional layers are separated by a max-pooling layer. After the last convolutional layer, a global average-pooling layer is applied. Following, the architecture includes three fully connected layers, culminating in a softmax layer.
- *ShallowConv* [10]. This architecture consists of three convolutional layers with kernel sizes equal to 3, 5, and 7, respectively, that learn to extract N-gram like features from the input

⁵We note that extending the maximum input length of these architectures is straightforward in practice. MalConv, MalConvGCT, AvastConv, and ShallowConv all employ global pooling operations (i.e., global max-pooling and global average-pooling), after their convolutional layers, which aggregate features across the entire input sequence into a fixed-size representation regardless of the input length. As a result, these architectures are inherently length-agnostic and extending the input from 1 MB to 2 MB at test time requires no architectural modifications, no retraining, and no changes to the model weights. The global pooling layer simply operates over a larger number of positions, producing the same fixed-size feature vector, that is, passed to the classification head. This property also implies that our defense can be extended to larger executables without any modification to the base classifiers, since the chunk-based inference pipeline classifies each chunk independently and the majority voting aggregation is similarly length-agnostic by construction.

⁶The SHA-256 hashes associated with our training, validation and test splits from the BODMAS dataset will be provided to allow researchers to reproduce our work.

bytes sequences. Afterwards, a global max-pooling layer is applied, followed by a feature concatenation layer and a softmax layer.

Using the aforementioned architectures as backbone, we trained 20 end-to-end malware detectors:

- *MalConv*, *MalConvGCT*, *AvastConv*, and *ShallowConv*. These detectors correspond to undefended versions of the *MalConv*, *MalConvGCT*, *AvastConv*, and *ShallowConv* models, respectively.
- *RS-MalConv*, *RS-MalConvGCT*, *RS-AvastConv*, and *RS-ShallowConv*. These detectors implement the randomized ablation scheme defined by Gibert et al. [12] with a probability of ablating a byte equals to 20%.
- *RsDel-MalConv*, *RsDel-MalConvGCT*, *RsDel-AvastConv*, and *RsDel-ShallowConv*. These detectors implement the randomized deletion scheme defined by Huang et al. [13] with a probability of deleting a byte equals to 3%.
- *DRSM-MalConv*, *DRSM-MalConvGCT*, *DRSM-AvastConv*, and *DRSM-ShallowConv*. These detectors implement the (de)randomized smoothing scheme defined by Saha et al. [26] with $K = 12$ partitions.
- *DRS-MalConv*, *DRS-MalConvGCT*, *DRS-AvastConv*, and *DRS-ShallowConv*. These detectors implement the (de)randomized smoothing scheme proposed in Section 3 using *MalConv*, *MalConvGCT*, *AvastConv*, and *ShallowConv* as base detectors.

The randomized smoothing methods (e.g. RS and RsDel) work by sampling a large number N (commonly 100 or even 1,000) of noisy (or edited) versions of the input, classifying each independently, and then aggregating via majority vote. As a consequence, these approaches impose a substantial computational burden at inference time (exceeding 10 seconds per instance when $N = 1000$ as documented by Huang et al. [13]) making them impractical for applications that demand fast predictions. Moreover, the higher value $N = 4000$ is typically required in order to derive probabilistic certificates of robustness. Yet even when used, those certificates only guarantee robustness within a rather small edit radius (i.e. limited number of byte edits), which reduces their utility in a real-world scenario. Given the slow speed and the narrow scope of certification (which cannot certify the number of byte edits of the attacks evaluated in Section 4.5), we decided not to generate those certificates in our setting. Instead, we reduced N to 20, and empirically observed that the resulting accuracy remains nearly unchanged compared to $N = 100$, while inference becomes much more efficient. For a full comparison of prediction times across methods, see Section 4.3.

Our (de)randomized smoothing defense splits executable files into chunks of size z , classify each chunk independently, and determine the final prediction through a majority voting. The choice of $z \in \{512, 1024, 2048, 4096\}$ is guided by (i) theoretical considerations from the certification framework, (ii) alignment with the structure of PE files, and (iii) empirical trade-offs between classification accuracy and robustness guarantees. The chunk size z directly impacts the robustness certificate margin (cf. Section 4.4). A smaller z yields a larger number of chunks, which, in turn, increases the minimum number of chunks that an attacker must manipulate to flip the majority vote. Conversely, a larger z provides more information per chunk, improving classification accuracy but reducing the robustness margin. Furthermore, PE files are organized in sections with alignment boundaries typically of 512, 1,024, 2,048, 4,096 or 8,192 bytes, and most structural elements (headers, sections, slack space) occur at these granularities.

4.2 Detection Performance on Clean Examples

Table 2 presents the detection accuracy (clean accuracy) and ROC AUC of the undefended and randomized smoothing models on the validation and test sets. The clean accuracy refers to the ability of the detectors to correctly classify the benign and malicious EXE files in the absence of any adversarial modification or structural manipulation. Our approach trains the classifier on

Table 2. Performance Metrics of the Detectors on the Validation Set, the Test Set and the Sub-Test of 500 Malware Examples used for the Adversarial Attacks Evaluation

	Validation set		Test set		Test sub-set
	Accuracy	ROC AUC	Accuracy	ROC AUC	Accuracy
MalConv	96.80	96.74	96.96	96.91	94.20
MalConvGCT	95.71	95.65	96.71	96.67	95.00
AvastConv	98.43	98.43	98.69	98.68	97.80
ShallowConv	99.33	99.32	99.54	99.54	99.60
RS-MalConv	98.77	98.75	98.60	98.57	97.60
RS-MalConvGCT	98.22	98.23	99.17	99.17	99.00
RS-AvastConv	98.40	98.39	98.36	98.34	98.40
RS-ShallowConv	99.07	99.05	99.26	99.25	98.80
RsDel-MalConv	98.17	98.17	98.09	98.07	97.40
RsDel-MalConvGCT	99.00	99.00	99.36	99.36	98.80
RsDel-AvastConv	98.53	98.51	98.65	98.63	97.40
RsDel-ShallowConv	99.05	99.06	99.49	99.50	100.00
DRSM-MalConv(k=12)	93.52	93.61	94.62	94.65	96.00
DRSM-MalConvGCT(k=12)	94.14	94.23	96.71	96.74	99.00
DRSM-AvastConv(k=12)	85.83	85.74	93.29	93.34	94.20
DRSM-ShallowConv(k=12)	95.34	95.42	97.75	97.77	98.80

	Validation set		Test set		Test sub-set
	Accuracy	ROC AUC	Accuracy	ROC AUC	Accuracy
DRS-MalConv (z=512)	94.36	94.42	96.06	96.06	96.80
DRS-MalConv (z=1024)	91.71	91.74	92.90	92.85	91.20
DRS-MalConv (z=2048)	93.63	93.70	95.64	95.66	96.80
DRS-MalConv (z=4096)	94.36	94.40	96.23	96.22	96.00
DRS-MalConvGCT (z=512)	94.18	94.24	95.75	95.75	95.80
DRS-MalConvGCT (z=1024)	94.27	94.33	96.46	96.47	96.80
DRS-MalConvGCT (z=2048)	94.28	94.35	96.29	96.31	97.00
DRS-MalConvGCT (z=4096)	93.84	93.89	93.26	93.27	97.00
DRS-AvastConv (z=512)	86.89	86.78	96.24	96.25	95.80
DRS-AvastConv (z=1024)	87.08	86.94	96.05	96.04	94.20
DRS-AvastConv (z=2048)	87.08	86.92	95.85	95.82	93.20
DRS-AvastConv (z=4096)	86.54	86.39	95.10	95.06	92.00
DRS-ShallowConv (z=512)	95.28	95.33	96.81	96.81	97.00
DRS-ShallowConv (z=1024)	93.76	93.87	96.81	96.85	99.00
DRS-ShallowConv (z=2048)	95.32	95.38	97.17	97.18	97.60
DRS-ShallowConv (z=4096)	93.85	93.97	97.15	97.19	99.40

With the sub-test set being composed of only malware we simply report the accuracy.

Highlighted values indicate the highest accuracy and ROC AUC for each set.

randomly sampled chunks extracted from executables rather than on full binaries. As a consequence, the classifier observes only partial byte sequences during training, which may not always contain complete malicious semantics. This reduces the prediction accuracy of individual chunks (between 86% and 89% depending on the architecture and chunk size) compared to models trained on larger byte sequences or full executables. However, this design choice is intentional and enables the majority-voting mechanism used to derive robustness guarantees. During inference, all chunks of

Table 3. Inference Time of the Malware Detectors

Malware Detector	CPU Inference Time	GPU Inference Time
MalConv	0.0132	0.0037
MalConvGCT	0.0579	0.0093
AvastConv	0.0542	0.0122
ShallowConv	0.2212	0.1597
RS-MalConv (N=20)	0.2882	0.0287
RS-MalConvGCT (N=20)	1.4038	0.0404
RS-AvastConv (N=20)	1.4075	0.0309
RS-ShallowConv (N=20)	3.7467	0.7644
RsDel-MalConv (N=20)	0.2736	0.0275
RsDel-MalConvGCT (N=20)	1.3270	0.0408
RsDel-AvastConv (N=20)	1.2751	0.0299
RsDel-ShallowConv (N=20)	3.6689	0.8259
DRSM-MalConv (k=12)	0.0450	0.0099
DRSM-MalConvGCT (k=12)	0.2821	0.0397
DRSM-AvastConv (k=12)	0.1750	0.0167
DRSM-ShallowConv (k=12)	0.2275	0.0478
DRS-MalConv (z=512)	0.0543	0.0241
DRS-MalConvGCT (z=2048)	0.5537	0.0414
DRS-AvastConv (z=1024)	0.6734	0.0220
DRS-ShallowConv (z=2048)	0.1839	0.009

Bold values denote the lowest inference time for CPU and GPU for each type of classifier, respectively.

an executable are classified independently and the final decision is obtained through majority voting. As shown in our experiments, this aggregation mitigates the impact of individual chunk-level misclassifications and restores strong detection performance at the file level. As shown in Table 2, the undefended models (MalConv, MalConvGCT, AvastConv, and ShallowConv) achieve slightly higher clean accuracy and ROC AUC ($\approx 2\%$) than their (de)randomized smoothing counterparts. For instance, the MalConv model achieved a ROC AUC score of 96.91 while its (de)randomized smoothing counterpart ($z = 512$) ROC AUC score is 96.06. However, as demonstrated in Section 4.5, the undefended models are vulnerable to evasion attacks that alter the structure of PE files. Moreover, they offer no formal certification guarantees against patch or content-injection attacks.

4.3 Inference Time of the End-to-End Malware Detectors

In the context of malware detection, rapid and efficient detection of potentially malicious software is imperative for real-time threat detection, and more importantly, within resource-constrained environments, including edge devices or systems where computational resources are limited. Therefore, we have analyzed the time required by all the models we have trained to quantify the toll imposed by our certification approach.

As shown in Table 3, the most efficient baseline architecture, regardless of the defense scheme, is MalConv. This is mainly due to two reasons: (1) MalConv uses very large strides compared to AvastConv; (2) MalConv employs the Temporal Max-Pooling strategy presented by Raff et al. [24], which leverages sparse gradients to significantly reduce runtime and memory constraints; (3) MalConv uses feed-forward layers instead of the Attention layer used in MalConvGCT. In contrast, the computational time of the randomized smoothing-based models significantly surpasses that of both the baseline and (de)randomized smoothing models. Furthermore, Table 4 provides a

Table 4. Comparative Analysis of Randomized Smoothing-Based Approaches with Different Number of Noisy Versions $N \in \{20, 100\}$

Malware detectors	Randomized smoothing (N=20)		
	ROC AUC	CPU inference time (seconds/example)	GPU inference time (seconds/example)
RS-MalConv	98.75	0.2882	0.0287
RS-MalConvGCT	98.23	1.4038	0.0404
RS-AvastConv	98.39	1.4075	0.0309
RS-ShallowConv	99.05	3.7467	0.7644
RsDel-MalConv	98.17	0.2736	0.0275
RsDel-MalConvGCT	98.99	1.3270	0.0408
RsDel-AvastConv	98.51	1.2751	0.0299
RsDel-ShallowConv	99.06	3.6689	0.8259
Malware detectors	Randomized smoothing (N=100)		
	ROC AUC	CPU inference time (seconds/example)	GPU inference time (seconds/example)
RS-MalConv	98.70	1.5536	0.1564
RS-MalConvGCT	98.32	7.9180	0.3146
RS-AvastConv	98.39	7.3287	0.2312
RS-ShallowConv	99.11	-	3.797
RsDel-MalConv	98.28	1.4236	0.1473
RsDel-MalConvGCT	99.05	7.3082	0.3247
RsDel-AvastConv	98.60	5.4106	0.2205
RsDel-ShallowConv	99.06	-	4.1566

Computational constraints prevented the calculation of {RS, RsDel}-ShallowConv ($N = 100$) on the CPU. Best results per metric and type of classifier are shown in bold.

comparative analysis between randomized smoothing methods based on ablations and deletions using 20 and 100 noisy versions. Results suggest that using 100 noisy versions marginally improve the performance while greatly increasing the inference time. For this reason, we used 20 noisy versions in our experiments.

4.4 Certified Accuracy Against Patch and Content-Injection Attacks

In Section 3, we proposed a certification procedure to verify and guarantee whether or not the model's predictions will vary in the face of potential adversarial attacks or variations in the input space produced by patch and content-injection attacks. Here, we provide the certified accuracy of the proposed (de)randomized smoothing scheme for the task of malware detection calculated with the aforementioned certification procedure. The certified accuracy is a metric used to quantitatively assess the model's reliability or robustness against a particular type of attack. In the context of malware detection, we assess the robustness of the malware detectors against patch and content-injection attacks.

Tables 5 and 6 present the certified accuracies of the malware detectors against patches and payloads of different sizes. As PE files exhibit significant variability in size, we compute the certified accuracy of the detectors with respect to adversarial patches and payloads whose sizes are defined as a percentage of the original file's size.

We compare the proposed scheme, DRS, against RS [12, 13], RsDel [13], and DRSM [26] defenses. RS constructs a smoothed classifier by randomly ablating bytes of the input, yielding a

Table 5. Certified Accuracy of the (De)randomized Smoothing-Based Detectors Against the Patch Attack

	Certified Accuracy												
	$p=1\%$	$p=2\%$	$p=3\%$	$p=5\%$	$p=10\%$	$p=15\%$	$p=20\%$	$p=25\%$	$p=30\%$	$p=35\%$	$p=40\%$	$p=45\%$	$p=50\%$
RS-MalConv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RsDel-MalConv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRSM-MalConv-12	87.11	87.11	87.11	87.11	82.57	82.57	75.03	75.03	56.70	0.0	0.0	0.0	0.0
DRS-MalConv ($z=512$)	95.36	95.20	95.07	94.77	93.49	91.22	88.21	85.15	80.46	74.53	62.49	40.26	0.0
DRS-MalConv ($z=1024$)	91.55	91.33	91.18	90.91	89.64	87.74	84.71	79.25	72.68	64.76	52.86	31.73	0.0
DRS-MalConv ($z=2048$)	93.88	93.42	93.37	92.89	91.63	89.71	84.98	81.59	75.88	67.63	55.93	31.73	0.0
DRS-MalConv ($z=4096$)	91.96	91.90	91.68	91.51	89.77	86.15	83.52	79.48	71.59	63.13	49.06	25.34	0.0
RS-MalConvGCT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RsDel-MalConvGCT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRSM-MalConvGCT-12	92.94	92.94	92.94	92.94	90.01	90.01	83.15	83.15	68.11	0.0	0.0	0.0	0.0
DRS-MalConvGCT ($z=512$)	95.18	95.08	94.81	94.27	92.86	90.78	88.63	84.27	77.83	68.84	57.58	37.69	0.0
DRS-MalConvGCT ($z=1024$)	95.47	95.21	95.08	94.92	93.78	92.29	89.22	85.90	80.87	75.73	65.75	42.37	0.0
DRS-MalConvGCT ($z=2048$)	94.25	94.03	93.93	93.67	92.64	91.14	88.22	84.06	76.62	68.05	56.87	32.44	0.0
DRS-MalConvGCT ($z=4096$)	92.83	92.54	92.48	92.18	90.66	88.27	85.16	78.26	69.62	60.16	44.73	22.55	0.0
RS-AvastConv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RsDel-AvastConv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRSM-AvastConv-12	85.20	85.20	85.20	85.20	79.88	79.88	70.99	70.99	54.59	0.0	0.0	0.0	0.0
DRS-AvastConv ($z=512$)	95.62	95.45	95.45	94.93	94.06	91.73	88.62	84.72	80.64	73.54	61.55	39.38	0.0
DRS-AvastConv ($z=1024$)	94.72	94.58	94.45	94.04	93.04	91.16	88.15	84.27	79.28	72.76	60.10	37.59	0.0
DRS-AvastConv ($z=2048$)	94.10	93.82	93.74	93.47	92.09	89.78	87.80	84.21	80.07	71.96	59.26	31.66	0.0
DRS-AvastConv ($z=4096$)	91.60	91.42	91.21	90.08	88.73	84.33	80.88	77.00	68.86	59.54	42.08	20.99	0.0
RS-ShallowConv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
RsDel-ShallowConv	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRSM-ShallowConv-12	95.67	95.67	95.67	95.67	92.37	92.37	88.34	88.34	77.48	0.0	0.0	0.0	0.0
DRS-ShallowConv ($z=512$)	96.50	96.42	96.33	96.15	94.54	92.47	89.93	88.58	85.72	82.21	74.28	53.01	0.0
DRS-ShallowConv ($z=1024$)	96.20	96.14	96.01	95.76	95.15	93.95	92.09	89.92	84.46	76.79	66.58	45.99	0.0
DRS-ShallowConv ($z=2048$)	95.73	95.71	95.40	95.24	94.73	94.06	92.83	89.95	84.49	78.14	67.88	44.75	0.0
DRS-ShallowConv ($z=4096$)	94.15	94.03	93.77	93.62	92.50	90.13	87.49	84.58	76.85	66.78	53.30	29.25	0.0

The patch size, p , corresponds to a percentage of the original file. Bold values denote the highest certified accuracy in each column.

Table 6. Certified Accuracy of the (De)randomized Smoothing-Based Detectors Against the Content-Injection Attack

		Certified Accuracy													
		$p=1\%$	$p=2\%$	$p=3\%$	$p=5\%$	$p=10\%$	$p=20\%$	$p=30\%$	$p=40\%$	$p=50\%$	$p=60\%$	$p=70\%$	$p=80\%$	$p=90\%$	$p=100\%$
RsDel-MalConv		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRS-MalConv ($z=512$)		95.83	95.47	95.40	95.27	94.94	93.68	91.57	88.49	85.57	81.78	76.80	65.08	43.06	0.0
DRS-MalConv ($z=1024$)		92.54	92.48	92.41	92.13	91.65	90.38	88.80	85.97	81.03	75.76	67.85	56.24	34.14	0.0
DRS-MalConv ($z=2048$)		95.14	95.08	94.99	94.75	94.17	93.01	91.35	88.04	84.45	79.423	73.15	64.10	37.85	0.0
DRS-MalConv ($z=4096$)		94.30	94.27	94.21	94.02	93.50	92.18	89.54	87.10	83.62	79.28	72.44	58.11	32.04	0.0
RsDel-MalConvGCT		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRS-MalConvGCT ($z=512$)		95.58	95.34	95.33	95.20	94.45	93.09	91.31	89.08	84.98	80.13	71.40	59.30	39.45	0.0
DRS-MalConvGCT ($z=1024$)		96.27	96.19	96.15	95.89	95.50	94.46	92.86	89.92	87.50	83.37	78.34	69.36	48.04	0.0
DRS-MalConvGCT ($z=2048$)		95.77	95.58	95.47	95.18	94.85	93.91	92.43	90.01	87.20	80.95	73.91	64.25	38.54	0.0
DRS-MalConvGCT ($z=4096$)		95.10	95.07	95.07	94.93	94.40	93.24	91.29	88.63	83.90	77.27	69.82	53.55	28.89	0.0
RsDel-AvastConv		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRS-AvastConv ($z=512$)		96.14	95.86	95.81	95.64	95.03	94.23	92.13	88.97	85.49	82.41	76.24	64.38	41.96	0.0
DRS-AvastConv ($z=1024$)		95.59	95.57	95.51	95.37	94.75	94.07	92.15	89.12	85.80	81.76	75.99	64.50	41.21	0.0
DRS-AvastConv ($z=2048$)		95.51	95.46	95.42	95.15	94.77	93.42	91.17	88.99	86.96	83.29	77.54	66.98	40.82	0.0
DRS-AvastConv ($z=4096$)		93.98	93.94	93.90	93.52	92.57	91.33	88.35	84.66	81.40	75.76	68.88	52.35	26.94	0.0
RsDel-ShallowConv		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DRS-ShallowConv ($z=512$)		96.79	96.72	96.53	96.42	96.22	94.80	92.68	90.16	88.92	86.72	83.75	76.63	57.91	0.0
DRS-ShallowConv ($z=1024$)		96.76	96.74	96.40	96.37	96.06	95.45	94.49	92.87	91.43	87.02	80.34	70.30	50.90	0.0
DRS-ShallowConv ($z=2048$)		97.02	97.01	96.98	96.92	96.45	95.85	95.37	94.37	92.33	87.45	83.75	75.96	54.14	0.0
DRS-ShallowConv ($z=4096$)		96.53	96.51	96.46	96.35	95.86	94.92	93.73	90.83	88.80	84.90	76.36	62.44	37.55	0.0

The adversarial payload, p , corresponds to a percentage of the original file. Bold values denote the highest certified accuracy per column.

Hamming-distance certificate over substitutions, and thus covering only patch attacks (it is therefore excluded from Table 6). RsDel instead randomly deletes bytes, yielding a Levenshtein-distance certificate over edit operations $O=\{\text{insertion, deletion, substitution}\}$, thereby covering both patch and content-injection attacks. DRSM partitions each input into a fixed number of K contiguous chunks, classifies each chunk independently, and aggregates the predictions by majority vote. In contrast, our DRS partitions each input into chunks of a fixed size z , so that the number of chunks grows with the file length. This distinction is crucial: because z is constant, a content-injection attack leaves the bytes of the original chunks unaltered and their predictions pinned, whereas DRSM's

fixed- K partitioning displaces every chunk boundary under injection and therefore cannot certify against content-injection attacks (DRSM is consequently, excluded from Table 6).

The results reveal three qualitative different certification behaviors. First, RS and RsDel yield a certified accuracy of 0.0 across all payload sizes. This is a direct consequence of the magnitude of their certified radii relative to the perturbations we consider. Both methods certify only very small edit budgets on the order of a hundred bytes (e.g., the certified radius of RsDel drops to zero at approximately 137 bytes in the original work [13], whereas a perturbation of even 1% of a PE file typically amounts to thousands of bytes, far exceeding what randomized ablation or deletion can certify. Second, DRSM [26] certified accuracy against a patch attack exhibits a staircase pattern and an abrupt cutoff. DRSM divides every file into a fixed number of $K = 12$ chunks, with each chunk spanning roughly 8.33% of the file. As a result, the certificate, which depends only on how many chunks a patch can overlap, changes only when a patch grows by about one chunk's worth. This produces the plateaus visible in Table 5, e.g., identical values from $p = 1\%$ to $p = 5\%$. Furthermore, certification requires the clean vote margin to exceed 2Δ , where Δ is the number of corrupted chunks. Since the margin cannot exceed $K = 12$, no sample can be certified once a patch can corrupt $\Delta = K/2 = 6$ chunks in the worst case. Because a patch may begin at any byte offset, it overlaps at most $\Delta = \lceil pK/100 \rceil + 1$ chunks, where each chunk spans $100/K \approx 8.33\%$ of the file. This bound reaches 6 as soon as $\lceil pK/100 \rceil = 5$, i.e., for $p > 100 \times 4/K \approx 33\%$. This is why DRSM certified accuracy collapses to 0.0 at $p = 35\%$, where $\Delta = \lceil 4.2 \rceil + 1 = 6$. Finally, the proposed DRS achieves the highest certified accuracy across nearly all settings and degrades as the perturbation grows, for both patch and content-injection attacks. Because DRS fixes the chunk size z rather than the chunk count, the number of chunks a perturbation of size p can affect scales smoothly with the file, producing the gradual decline seen in both tables rather than DRSM's abrupt plateaus and cutoff. In the case of patch attacks, the certified accuracy drops to 0 when the adversarial patch matches half the size of the original executable. This decline is attributed to the circumstance where modifying half the size of the executables leads to having a higher number of mislabeled chunks. In the worst-case scenario, i.e., when the beginning or/and end of the adversarial patch intersects with two chunks instead of one, then half of the chunks plus one will be mislabeled, and thus, the number of chunks that will be correctly labeled will be surpassed, flipping the prediction of the classifier. Similarly, in the context of content-injection attacks, the certified accuracy of the detectors reaches 0 when inserting an adversarial payload of the same size as the original executables. In contrast to patch attacks, the content-injection attack does not modify the original content of the executable but adds new content. Consequently, the injection of a payload with size equals to the original executable's size leads to a situation where the total file size doubles and the injected payload accounts for half of all chunks, driving the certified accuracy to 0.0.

4.5 Certified Robustness Against State-of-the-Art Evasion Attacks

We now briefly outline the functionality-preserving attacks used to evaluate our defense. The attacks can be divided into (1) patch attacks and (2) content-injection attacks. Regarding the patch attacks, we leverage the Slack [28] attack, which manipulates the already-available empty space between sections, that is, left by the compiler to maintain the alignment. Regarding the content-injection attacks, we leverage: (i) the Padding attack [14], which includes new content at the end of the file by just appending bytes as overlay with payload sizes $p = 5000$ and $p = 10000$ bytes; (ii) the Shift attack [6], which manipulates the PE file format to inject space $p \in \{1024, 2048, 4096\}$ bytes between the optional header and the first section, and later fills this space with adversarial content; the GAMMA attack [5], which creates up to 10 new sections filled with content extracted from

the .data sections of legitimate samples in the test set;⁷ and (iv) the Code-caves attack [30], which dynamically extend the size of sections in malicious software to inject an adversarial payload. We cap the expansion at 20% of the original section size per section to avoid unrealistic growth while providing a meaningful injection budget. These parameter choices let us evaluate the detectors across a spectrum of realistic injection budgets, from small appended overlays to multi-section and section-expansion strategies.

Furthermore, we create an adaptive attack that injects multiple payloads at different regions of a PE file by combining all the manipulations described above. This attack, named all-in-one attack, proceeds in five main steps: (1) shifting content between the header and the sections by 4,096 bytes; (2) extending code caves between sections by up to 10% of the section's original size; (3) creating five new sections filled with benign content; and (4) appending 10000 bytes in the overlay. We refer the readers to Appendix 5.1 for a detailed description of the all-in-one attack.

The adversarial payloads injected by the aforementioned attacks are optimized with genetic algorithms [6] to convert the white-box attacks [5, 30] to black-box attacks, and thus, be able to attack randomized smoothing-based defenses [12, 13] and our (de)randomized smoothing-based defense. The hyperparameters of the genetic algorithm are the following: (1) the population size is equal to 10; (2) the maximum number of steps is equal to 50; (3) the probability of an individual to be mutated is 30%; (4) the probability of a byte to be mutated is 30%; and (5) we use soft label instead of hard label. As some attacks may require several hours to execute per file, we use a reduced-size test set consisting of 500 malicious examples. By employing a smaller subset, we aim to reduce the computational overhead and accelerate the overall experimentation process. This approach has allowed us to obtain meaningful insights and results within a reasonable timeframe, considering the time-intensive nature of the experiments involved. Nevertheless, we would like to denote that our evaluation set is comparable in size to prior work [5, 6, 12–14, 28].

Tables 7 and 8 present the adversarial accuracy as well as the certified accuracy (within brackets) against the aforementioned SOTA adversarial attacks. The clean accuracy refers to the accuracy of the malware detectors against non-adversarial, “clean” examples. The adversarial accuracy refers to the accuracy of the malware detectors against adversarial EXEmple generated with the adversarial attacks. In contrast, the certified accuracy measures the performance of the malware detectors against a worst-case scenario within the specified attacks. In other words, it provides a guarantee of a model's performance against the specified types of manipulations. Note that, although all four defenses (RS, RsDel, DRSM, and DRS) admit robustness certificates, certified accuracy is reported only for RsDel and DRS defenses. This is because the certificates differ in the set of byte-level operations they cover, and only RsDel and DRS certify the operations exercised by the attacks considered here as RS and DRSM only cover patch attacks. The evaluated attacks, however, are either pure content-injection attacks (Shift, GAMMA) or combinations of patch and content-injection attacks (Padding+Slack Space, Code Caves, and All-in-One). Consequently, these attacks fall outside the substitution-only threat model that RS and DRSM certify.

Observing Tables 7 and 8, we can see that the baseline models degrade significantly under even modest adversarial perturbations (e.g., appending content in the overlay or shifting content). For instance, MalConv goes from 94.2% clean accuracy to 85.00% under Padding and Slack, 35.6% under

⁷Although an attacker could in principle inject arbitrarily large payloads by creating an unlimited number of new sections, doing so would produce executables of anomalous size that are easily detectable by complementary file integrity checks or structural anomaly detectors. To keep adversarial EXEmple both realistic and stealthy, we therefore limit each adversarial EXEmple to at most twice the size of the corresponding original sample.

Table 7. Detection Accuracy (%), with Certified Accuracy Reported in Brackets, of the Malware Detectors on Adversarial EXEmples Generated with the Padding+Slack Space Attack [28] and the Shift Attack [6]

Malware detectors	Clean examples	Padding+Slack Space Attack			Shift Attack	
		$p=5000$	$p=10000$	$p=1024$	$p=2048$	$p=4096$
MalConv	94.20	87.40	85.00	42.00	35.60	35.60
RS-MalConv	97.60	90.40	90.00	75.00	59.20	49.20
RsDel-MalConv	97.80	82.60 (0.0)	73.80 (0.0)	86.60 (0.0)	80.80 (0.0)	77.20 (0.0)
DRSM-MalConv-12	96.00	94.00	93.60	95.20	95.20	95.20
DRS-MalConv (z=512)	96.80	96.20 (95.80)	95.80 (95.40)	96.80 (96.80)	96.80 (96.60)	96.60 (96.60)
MalConvGCT	95.00	72.60	70.20	79.40	76.00	76.80
RS-MalConvGCT	99.00	96.80	96.00	95.80	94.20	81.60
RsDel-MalConvGCT	98.80	81.40 (0.0)	68.80 (0.0)	95.40 (0.0)	94.20 (0.0)	84.80 (0.0)
DRSM-MalConvGCT-12	99.00	96.60	96.40	97.40	92.70	97.40
DRS-MalConvGCT (z=2048)	97.00	96.20 (94.20)	96.20 (94.40)	96.60 (96.60)	96.60 (96.60)	96.60 (96.60)
AvastConv	97.80	96.40	93.60	61.80	42.80	31.20
RS-AvastConv	98.40	94.60	85.60	64.20	63.40	52.40
RsDel-AvastConv	97.40	93.60 (0.0)	89.20 (0.0)	63.20 (0.0)	58.80 (0.0)	52.40 (0.0)
DRSM-AvastConv-12	94.20	89.20	92.00	92.20	91.20	91.80
DRS-AvastConv (z=1024)	94.20	93.60 (93.20)	93.00 (92.80)	94.20 (94.20)	94.20 (94.20)	94.20 (93.80)
ShallowConv	99.60	32.20	24.60	90.00	78.80	71.20
RS-ShallowConv	98.80	72.40	55.80	92.80	85.40	76.20
RsDel-ShallowConv	100.00	96.00 (0.0)	89.80 (0.0)	99.60 (0.0)	99.20 (0.0)	98.40 (0.0)
DRSM-ShallowConv-12	98.80	98.40	98.40	98.40	98.20	98.40
DRS-ShallowConv (z=2048)	97.60	97.40 (97.00)	97.20 (96.20)	97.60 (97.60)	97.60 (97.60)	97.60 (97.60)

The highest detection accuracy per attack is shown in bold.

Shift, 26.2% under GAMMA, 22.2% under code caves, and 33.0% under the all-in one attack. Similar patterns are observed for MalConvGCT, AvastConv, and ShallowConv.

Randomized smoothing models improve robustness moderately but is not uniform across all attack types and network architectures. For instance, RS-MalConv increases the accuracy against the padding and shift attacks but perform slightly worse against the GAMMA, code caves, and all-in-one attacks. On the other hand, RS-MalConvGCT increase the robustness against all attacks, although its performance remains well below its clean accuracy. Overall, RsDel-ShallowConv is the strongest randomized model, remaining strong under all attacks with the exception of the code caves and all-in-one attack where it falls below 80% accuracy.

Regarding the (de)randomized smoothing models, all consistently achieve higher robustness than their vanilla and randomized counterparts, with the only exception being RsDel-ShallowConv, which slightly outperforms DRS-ShallowConv against the Shift attack. Nevertheless, DRS-MalConv maintains around 96% adversarial accuracy under padding, shift, GAMMA and code caves attack, while also exhibiting certified accuracy values that exceed those of their vanilla and randomized counterparts. For example, under the Code Caves attack, DRS-MalConv achieves 96.00% accuracy and 90.40% certified accuracy, compared to 22.20%, 25.20%, and 68.60% adversarial accuracy for MalConv, RS-MalConv, and RsDel-MalConv, respectively. Similar trends are observed for other attacks and across all network architectures, making (de)randomized smoothing the most robust defense evaluated. This robustness stems from its chunking strategy: by splitting files into non-overlapping chunks, the detector can still recover the original label as long as the injected payload does not exceed the original file size (cf. Section 4.4). In such cases, most chunks still contain the content of the original sample, enabling correct classification in the presence of adversarial payloads.

4.6 Visualization Tool

The proposed (de)randomized smoothing classification system is interpretable by design. This is because the proposed chunk-based classification system assesses the maliciousness of each chunk

Table 8. Detection Accuracy (%), with Certified Accuracy Reported in Brackets, of the Malware Detectors on Clean Examples and Adversarial EXEmples Generated with Different Attacks: GAMMA [5], Code Caves [30], and the Combination Attack

Malware detectors	Clean examples	GAMMA ($s = 10$)	Code Caves Attack	All-in-One Attack
MalConv	94.20	26.20	22.20	33.00
RS-MalConv	97.60	24.20	25.20	25.00
RsDel-MalConv	97.80	91.80 (0.0)	68.60 (0.0)	66.00 (0.0)
DRSM-MalConv-12	96.00	85.00	87.00	78.60
DRS-MalConv ($z=512$)	96.80	96.20 (93.40)	96.00 (90.40)	85.20 (81.60)
MalConvGCT	95.00	93.80	62.60	81.00
RS-MalConvGCT	99.00	97.00	82.40	75.20
RsDel-MalConvGCT	98.80	96.80 (0.0)	58.00 (0.0)	66.80 (0.0)
DRSM-MalConvGCT-12	99.00	93.60	93.40	81.20
DRS-MalConvGCT ($z=2048$)	97.00	93.00 (84.40)	95.40 (84.00)	83.00 (69.60)
AvastConv	97.80	58.80	20.60	31.20
RS-AvastConv	98.40	74.20	40.00	33.40
RsDel-AvastConv	97.40	74.60 (0.0)	36.80 (0.0)	29.20 (0.0)
DRSM-AvastConv-12	94.20	79.00	80.40	73.00
DRS-AvastConv ($z=1024$)	94.20	93.60 (90.60)	93.00 (86.20)	79.20 (73.00)
ShallowConv	99.60	92.80	31.60	31.60
RS-ShallowConv	98.80	78.80	47.40	51.60
RsDel-ShallowConv	100.00	97.60 (0.0)	82.80 (0.0)	78.80 (0.0)
DRSM-ShallowConv-12	98.80	93.80	94.00	80.00
DRS-ShallowConv ($z=2048$)	97.60	96.00 (89.20)	97.00 (87.20)	84.60 (71.40)

The highest detection accuracy per attack is shown highlighted.

independently, facilitating a finer-grained analysis of the file. This allows us to identify which specific chunks within a file exhibit malicious or benign traits. As a result, our approach could be used to help cybersecurity analysts to identify which parts of a suspicious file contain the malicious code. Furthermore, the predictions for each chunk can serve as the basis for generating a visual depiction or an overview of the file, emphasizing chunks with the higher and lower maliciousness scores. For instance, Figures 4, and 5 provide a graphic depiction of the scores of examples belonging to the AutoInject and GandCrab families. You can observe that the visualization of the malicious EXEmples belonging to the same family are similar between them while distinct from the EXEmples belonging to different families.

4.7 Discussion

Section 4.4 highlights a potential limitation of our (de)randomized smoothing defense. Our defense splits an executable file into contiguous, non-overlapping chunks, classifies each chunk independently, and a majority vote calculates the final prediction. A knowledgeable attacker could exploit, in theory, the reliance of our defense on the majority voting mechanism to bypass detection. However, unlike in the image domain, where an attacker can arbitrarily modify any pixel, manipulating executable files is significantly more constrained because arbitrary byte modifications may break the functionality or executability of the program. As a result, the regions of an executable that can be safely modified without affecting functionality are limited.

For a patch attack, if the attacker modifies enough chunks to flip their predictions to benign, then the final classification will change from malicious to benign. In practice, however, the number

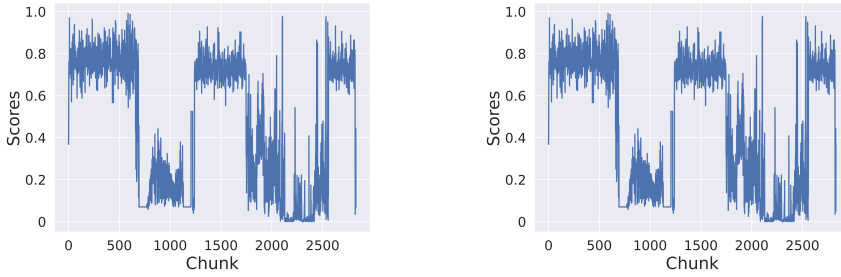


Fig. 4. Graphical visualization of the scores of malware samples belonging to the Autoinject malware family.

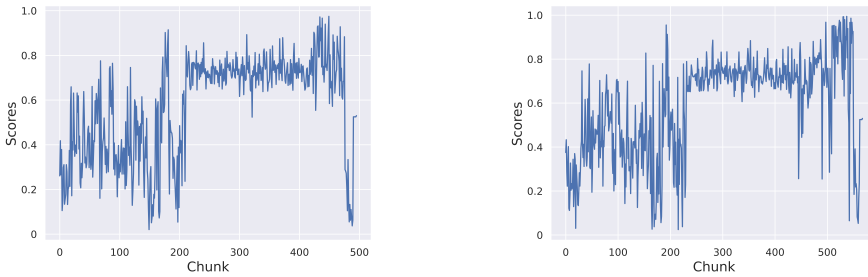


Fig. 5. Graphical visualization of the scores of malware samples belonging to the GandCrab malware family.

of bytes that can be altered without breaking the functionality of the executable file are limited to the bytes in the DOS header [6] and the bytes in the slack space [28]. This restriction imposes a natural constraint on the size of the adversarial patches, making it very challenging for the attacker to evade detection as the sum of the size of the DOS header and the slack space is relatively small compared to the total size of the executables.

For a content-injection attack, if the attacker inserts an adversarial payload large enough to influence a majority of the chunks, the final classification can be flipped from malicious to benign. However, the locations of an executable file that the attacker can freely manipulate without breaking its functionality are limited to: (1) between the headers and the first section [6], (2) in newly-created sections [5], (3) between sections [30], and (4) at the end of the file [14]. In consequence, injecting a payload large enough to dominate the majority voting may be impractical for maintaining stealth and avoiding detection by other security mechanisms. For instance, our defense could be complemented with rules that check these locations for anomalies [15, 21]. Nevertheless, to break our defense an attacker would need to inject an adversarial payload large enough to manipulate half of the chunks an executable is divided into. This in stark contrast to randomized smoothing defenses [12, 13], where payloads of a few hundreds or thousands of bytes are enough to flip the prediction of the classifier.

Therefore, an adaptive attacker is effectively limited to the aforementioned functionality-preserving manipulations. These manipulations are explicitly considered in our all-in-one attack, which combines multiple realistic modification strategies, including injecting payloads between the headers and the first section, extending existing code caves, creating new sections, and appending content to the overlay. This attack scenario represents a strong adaptive adversary that simultaneously exploits several valid modification locations within the executable while preserving program

functionality. As demonstrated in Section 4.5, even under this stronger threat model, our approach remains resilient and consistently outperforms competing defenses.

5 Conclusions

In this article, we present the first model-agnostic certified defense against functionality-preserving adversarial attacks on end-to-end malware detectors. Building upon recent research on (de)randomized smoothing, we propose a chunk-based smoothing scheme that provides deterministic robustness certificates against patch and content-injection attacks.

Our defense operates by training on subsets or chunks of bytes extracted from a given input executable. During inference, the executable is divided into contiguous, non-overlapping chunks, each of which is classified independently. The final prediction is then determined by aggregating these individual chunk predictions through a majority voting mechanism. This design is key for mitigating the impact of the injected adversarial content. By evaluating each chunk separately, the influence of the adversarial payload is effectively mitigated as it does not influence a significant proportion of the chunks, limiting its capability of altering the outcome of the classifier. This is in stark contrast to randomized smoothing defenses, where payloads of a few hundreds or thousands of bytes can flip the prediction of the classifier.

Our experiments, using as base classifier four end-to-end architectures, (1) MalConv, (2) MalConvGCT, (3) AvastConv, and (4) ShallowConv, demonstrate the superior robustness of our (de)randomized smoothing defense against a wide range of adversarial attacks. The novel application of our defense mechanism sets a new benchmark for certifiable methods in terms of detection accuracy against adversarial EXEmples and achieves a significant advancement in the field.

5.1 Future Work

Building up on top of our findings, the most apparent direction for future research involves complementing our defense with file integrity checks to identify and flag suspicious files [15, 21]. These checks could scan the file structure in search of anomalies, such as abnormal headers and overlay sizes, unusual spaces between sections, and the presence of atypical sections. This way, the attacker would be forced to inject smaller payloads, limiting its ability to influence the classifier's majority voting mechanism, as smaller payloads are less likely to alter a sufficient number of chunks to bypass our (de)randomized smoothing defense.

References

- [1] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*. 183–194. DOI : <https://doi.org/10.1145/2857705.2857713>
- [2] Hyrum S. Anderson and Phil Roth. 2018. Ember: An open dataset for training static pe malware machine learning models. (2018). DOI : <https://doi.org/10.48550/arXiv.1804.04637>
- [3] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 1310–1320. Retrieved from <https://proceedings.mlr.press/v97/cohen19c.html>
- [4] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2019. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *Proceedings of the 3rd Italian Conference on Cyber Security* (2019). Retrieved from <https://ceur-ws.org/Vol-2315/paper09.pdf>
- [5] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3469–3478. DOI : <https://doi.org/10.1109/TIFS.2021.3082330>
- [6] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial EXEmples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection. *ACM Transactions on Privacy and Security* 24 (2021). DOI : <https://doi.org/10.1145/3473039>

- [7] Ecenaz Erdemir, Jeffrey Bickford, Luca Melis, and Sergul Aydoore. 2021. Adversarial robustness with non-uniform perturbations. In *Proceedings of the Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). Retrieved from <https://openreview.net/forum?id=oi08QWks84>
- [8] Lijun Gao and Zheng Yan. 2024. CertRob: Detecting PDF malware with certified adversarial robustness via randomization smoothing. In *Proceedings of the 2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 944–951. DOI : <https://doi.org/10.1109/TrustCom63139.2024.00137>
- [9] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications* 153 (2020), 102526. DOI : <https://doi.org/10.1016/j.jnca.2019.102526>
- [10] Daniel Gibert, Carles Mateu, Jordi Planes, and Joao Marques-Silva. 2021. Auditing static machine learning anti-malware tools against metamorphic attacks. *Computers & Security* 102 (2021), 102159. DOI : <https://doi.org/10.1016/j.cose.2020.102159>
- [11] Daniel Gibert, Giulio Zizzo, and Quan Le. 2023. Certified robustness of static deep learning-based malware detectors against patch and append attacks. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*. 173–184. DOI : <https://doi.org/10.1145/3605764.3623914>
- [12] Daniel Gibert, Giulio Zizzo, and Quan Le. 2023. Towards a practical defense against adversarial attacks on deep learning-based malware detectors via randomized smoothing. In *28th European Symposium on Research in Computer Security Workshops*. Springer. DOI : https://doi.org/10.1007/978-3-031-54129-2_40
- [13] Zhuoqun Huang, Neil Marchant, Keane Lucas, Lujia Bauer, Olya Ohrimenko, and Benjamin I. P. Rubinstein. 2023. RS-Del: Edit distance robustness certificates for sequence classifiers via randomized deletion. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2023/file/3ba82362eb0aa75487069f19fde794fe-Paper-Conference.pdf
- [14] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *Proceedings of the 26th European Signal Processing Conference, EUSIPCO*. IEEE, 533–537. DOI : <https://doi.org/10.23919/EUSIPCO.2018.8553214>
- [15] Matous Kozak, Luca Demetrio, Dmitrijs Trizna, and Fabio Roli. 2025. Updating windows malware detectors: Balancing robustness and regression against adversarial EXEmples. *Computers & Security* 155 (2025), 104466. DOI : <https://doi.org/10.1016/j.cose.2025.104466>
- [16] Marek Krcál, Ondrej Svec, Martin Bálek, and Otakar Jasek. 2018. Deep convolutional malware classifiers can learn from raw executables and labels only. In *Proceedings of the 6th International Conference on Learning Representations, ICLR*. Retrieved from <https://openreview.net/pdf?id=HkHrmM1PM>
- [17] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. 2018. Deceiving end-to-end deep learning malware detectors using adversarial examples. *NeurIPS Workshop* 2018 (2018). Retrieved from <https://secml2018.github.io/malware.pdf>
- [18] Mathias Lécuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In *Symposium on Security and Privacy, SP*. IEEE, 656–672. DOI : <https://doi.org/10.1109/SP.2019.00044>
- [19] Alexander Levine and Soheil Feizi. 2020. (De) randomized smoothing for certifiable defense against patch attacks. *Advances in Neural Information Processing Systems* 33 (2020), 6465–6475. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2020/file/47ce0875420b2dbacfc5535f94e68433-Paper.pdf
- [20] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. 2019. Certified adversarial robustness with additive noise. In *Proceedings of the 32nd Advances in Neural Information Processing Systems (NeurIPS)*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 9459–9469. DOI : <https://dl.acm.org/doi/10.5555/3454287.3455136>
- [21] Liang Liu, Xinyu Kuang, Lin Liu, and Lei Zhang. 2024. Defend against adversarial attacks in malware detection through attack space management. *Computers & Security* 141 (2024), 103841. DOI : <https://doi.org/10.1016/j.cose.2024.103841>
- [22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems* 32 (2019), 8024–8035. DOI : <https://dl.acm.org/doi/10.5555/3454287.3455008>
- [23] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. 2018. Malware detection by eating a whole EXE. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 268–276. Retrieved from <https://cdn.aaai.org/ocs/ws/ws0432/16422-75958-1-PB.pdf>
- [24] Edward Raff, William Fleshman, Richard Zak, Hyrum S. Anderson, Bobby Filar, and Mark McLean. 2021. Classifying sequences of extreme length with constant memory applied to malware detection. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*. 9386–9394. DOI : <https://doi.org/10.1609/aaai.v35i11.17131>

- [25] Edward Raff, Jared Sylvester, and Charles Nicholas. 2017. Learning the PE header, malware detection with minimal domain knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISeC '17)*. 121–132. DOI : <https://doi.org/10.1145/3128572.3140442>
- [26] S. Saha, W. Wang, Y. Kaya, S. Feizi, and T. Dumitras. 2024. Drsm: De-randomized smoothing on malware classifier providing certified robustness. *ICLR 2024* (2024). Retrieved from <https://iclr.cc/virtual/2024/poster/17905>
- [27] Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. 2019. Provably robust deep learning via adversarially trained smoothed classifiers. In *Proceedings of the Advances in Neural Information Processing Systems 32* (2019). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/3a24b25a7b092a252166a1641ae953e7-Paper.pdf
- [28] Octavian Suciu, Scott E. Coull, and Jeffrey Johns. 2019. Exploring adversarial examples in malware detection. In *IEEE Security and Privacy Workshops*. IEEE, 8–14. Retrieved from <https://arxiv.org/pdf/1810.08280>
- [29] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2021. BODMAS: An open dataset for learning based temporal analysis of PE malware. In *Proceedings of the 4th Deep Learning and Security Workshop*. DOI : <https://doi.org/10.1109/SPW53761.2021.00020>
- [30] Javier Yuste, Eduardo G. Pardo, and Juan Tapiador. 2022. Optimization of code caves in malware binaries to evade machine learning detectors. *Computers & Security* 116 (2022), 102643. DOI : <https://doi.org/10.1016/j.cose.2022.102643>
- [31] Dazhi Zhan, Yexin Duan, Yue Hu, Weili Li, Shize Guo, and Zhisong Pan. 2024. MalPatch: Evading DNN-based malware detection with adversarial patches. *IEEE Transactions on Information Forensics and Security* 19 (2024), 1183–1198. DOI : <https://doi.org/10.1109/TIFS.2023.3333567>

Appendix

Proofs

We define the Proof using a rationale akin to that presented by Levine et al. [19]. Recall the definitions and statement of Theorem 1. In particular, recall the base classification counts $n_c(x)$:

$$\forall_c \in C, n_c(x) := \sum_{j=0}^N f_c(\tilde{x}^{(j)}). \quad (1)$$

And recall the definition of the (de)randomized smoothing classifier:

$$g(x) := \arg \max_c n_c(x), \quad (2)$$

where in the case of ties, we choose the larger-indexed class as the argmax solution, i.e., malicious class.

THEOREM 1. *For any input example x , base classifier f , (de)randomized smoothing classifier g , chunk size z and patch size p , such as $\Delta = \left\lceil \frac{p}{z} \right\rceil + 1$, if:*

$$n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + 2\Delta, \quad (4)$$

where $n_{c'}(x)$ and $n_{c''}(x)$ denote the most commonly and second most commonly predicted classes for an input example x , then for any adversarial EXEmple x' which differs from x only in a patch of size p , $g(x') = c'$.

PROOF. Recall that an input example x of size L is split into contiguous, non-overlapping chunks of size z , $x = \{\tilde{x}^{(0)}, \tilde{x}^{(1)}, \tilde{x}^{(2)}, \dots, \tilde{x}^{(N)}\}$, where $N = \left\lceil \frac{L}{z} \right\rceil$ and $\tilde{x}^{(j)}$ represent a chunk of bytes from x starting and ending at positions $j \times z$ and $(j + 1) \times z$, respectively.

Let k represent the starting byte or position within x' where the patch is inserted. Then, the end byte will be $k + p$, where p is the size of the patch. Note that, for all $c \in C = \{c_1, c_2\}$, the output $f_c(\tilde{x}^{(j)})$ will be equal to $f_c(\tilde{x}'^{(j)})$, unless the chunk $\tilde{x}^{(j)}$ intersects with the adversarial patch of size p starting at position k and ending at position $k + p$. This condition occurs only when $\tilde{x}^{(j)}$ is in the range k and $k + p$. Note that there will be at most $\Delta = \left\lceil \frac{p}{z} \right\rceil + 1$ chunks that will meet this condition.

Therefore, $f_c(\tilde{x}^{(j)}) = f_c(\tilde{x}'^{(j)})$ in all but Δ cases and thus, because $f_c(\cdot) \in \{0, 1\}$,

$$\forall n_c, |n_c(x) - n_c(x')| \leq \Delta. \quad (13)$$

Now consider any $c'' \neq c'$ such that $n_{c'}(x) \geq [n_{c''}(x) + 1_{c' < c''}] + 2\Delta$. There are two cases:

- $c' < c''$: In this case, in the event that $n_{c'}(x') = n_{c''}(x')$, we have that $g(x') = c''$. Therefore, a sufficient condition for $g(x') \neq c''$ is that $n_{c'}(x') > n_{c''}(x')$. By Equation (15) and triangle inequality, this must be true if $n_{c'}(x) > [n_{c''}(x)] + 2\Delta$, or equivalently, if $n_{c'}(x) \geq [n_{c''}(x) + 1_{c' < c''}] + 2\Delta$.
- $c'' < c'$: In this case, in the event that $n_{c'}(x') = n_{c''}(x')$, we have that $g(x') = c'$. Therefore, a sufficient condition for $g(x') \neq c''$ is that $n_{c'}(x') \geq n_{c''}(x')$. By Equation (15) and triangle inequality, this must be true if $n_{c'}(x) \geq [n_{c''}(x) + 1_{c' < c''}] + 2\Delta$.

Therefore, if $n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + 2\Delta$, then no class other than c can be output by $g(x')$. $\square$

The proof of Theorem 2 can be conducted in a similar manner by replacing 2Δ with Δ as follows:

THEOREM 2. *For any input example x , base classifier f , (de)randomized smoothing classifier g , chunk size z and an injected adversarial payload of size q , such as $\Delta = \lfloor \frac{q}{z} \rfloor + 1$, if:*

$$n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + \Delta, \quad (14)$$

where $n_{c'}(x)$ and $n_{c''}(x)$ denote the most commonly and second most commonly predicted classes for an input example x , then for any adversarial EXExample x' which differs from x only in an injected adversarial payload of size q , $g(x') = c'$.

PROOF. Let k represent the starting byte or position within x' where the adversarial payload is inserted. Then, the end byte will be $k + q$, where q is the size of the adversarial payload. Note that, for all $c \in C = \{c_1, c_2\}$, the output $f_c(\tilde{x}^{(j)})$ will be equal to $f_c(\tilde{x}'^{(j)})$, unless the chunk $\tilde{x}^{(j)}$ intersects with the adversarial patch of size q starting at position k and ending at position $k + q$. Due to the preprocessing step introduced in Section 3.2.2, this holds true as long as the attacker inserts the adversarial payload at the end of existing sections, between sections, in newly-created sections or at the overlay. As a result, there will be at most one chunk from x' that will intersect with the adversarial payload and $\lfloor \frac{q}{z} \rfloor$ chunks containing the newly-inserted adversarial payload. Therefore, $f_c(\tilde{x}^{(j)}) = f_c(\tilde{x}'^{(j)})$ in all but $\Delta = \lfloor \frac{q}{z} \rfloor + 1$ cases and thus, because $f_c(\cdot) \in \{0, 1\}$,

$$\forall n_c, |n_c(x) - n_c(x')| \leq \Delta. \quad (15)$$

Now consider any $c'' \neq c'$ such that $n_{c'}(x) \geq [n_{c''}(x) + 1_{c' < c''}] + \Delta$. There are two cases:

- $c' < c''$: In this case, in the event that $n_{c'}(x') = n_{c''}(x')$, we have that $g(x') = c''$. Therefore, a sufficient condition for $g(x') \neq c''$ is that $n_{c'}(x') > n_{c''}(x')$. By Equation (15) and triangle inequality, this must be true if $n_{c'}(x) > [n_{c''}(x)] + \Delta$, or equivalently, if $n_{c'}(x) \geq [n_{c''}(x) + 1_{c' < c''}] + \Delta$.
- $c'' < c'$: In this case, in the event that $n_{c'}(x') = n_{c''}(x')$, we have that $g(x') = c'$. Therefore, a sufficient condition for $g(x') \neq c''$ is that $n_{c'}(x') \geq n_{c''}(x')$. By Equation (15) and triangle inequality, this must be true if $n_{c'}(x) \geq [n_{c''}(x) + 1_{c' < c''}] + \Delta$.

Therefore, if $n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + \Delta$, then no class other than c can be output by $g(x')$. $\square$

Lastly, let's recall Theorem 3:

THEOREM 3. *For any input example x , base classifier f , (de)randomized smoothing classifier g , chunk size z , K adversarial patches $p_1, \dots, p_K$, and M adversarial injected payloads of sizes $q_1, \dots, q_M$ applied at disjoint locations, such that $\Delta_{combined}$ is defined as in Equation (10), if:*

$$n_{c'}(x) \geq \max_{c'' \neq c'} [n_{c''}(x) + 1_{c' < c''}] + \Delta_{combined}, \quad (16)$$

where $n_{c'}(x)$ and $n_{c''}(x)$ denote the most commonly and second most commonly predicted classes for an input example x , then for any adversarial EXExample x' obtained from x by simultaneously applying all K patches and M injections, $g(x') = c'$.

PROOF. Let x' be an adversarial EXExample obtained from x by applying K patches and M injections at disjoint locations. By Theorem 1, each patch of size p_i , can alter the predictions of at most Δ_i^{patch} chunks, since a patch p_i can intersect with at most $\lceil \frac{p_i}{z} \rceil + 1$ non-overlapping chunks in either direction, so the net change in $n_{c'}$ attributable to the i th patch is bounded by $2\Delta_i^{patch}$ in the worst case. By Theorem 2 and the PREPROCESS alignment, each injected payload of size q_j affects at most Δ_j^{inject} chunks, i.e. at most one boundary chunk shared with the original content plus $\lceil \frac{q_j}{z} \rceil$ chunks containing the injected payload. Since the injected content does not alter the predictions of existing unaffected chunks, the net change in $n_{c'}$ attributable to the j th injection is bounded by Δ_j^{inject} . Because the patch and injected regions are disjoint, the sets of affected chunks are disjoint, and the contributions to the total change in $n_{c'}$ are additive. Therefore:

$$\forall n_{c'} |n_c(x) - n_c(x')| \leq \Delta_{combined}. \quad (17)$$

The remainder of the proof follows the same analysis as Theorems 1 and 2: if Equation (16) holds, then for any $c'' \neq c'$, the adversary cannot accumulate enough chunk-prediction changes to make $n_{c'}(x') \geq n_{c''}(x')$ (accounting for tie-breaking), and therefore $g(x') = c'$. $\square$

All-in-One Attack

The all-in-one attack is an adaptive, multi-location attack that combines multiple manipulation strategies into a single, coordinated adversarial transformation. The rationale behind this attack is to simultaneously exploit all locations within a PE file that can be modified without breaking its functionality, thereby maximizing the adversarial payload injected into the executable while minimizing the probability that any single manipulation is sufficient to evade detection on its own. By distributing the adversarial content across multiple disjoint regions of the file, the attack aims to influence the predictions of as many chunks as possible, challenging the majority voting mechanism of our (de)randomized smoothing defense.

Concretely, the all-in-one attack proceeds in four steps. First, it applies a Shift manipulation [6] by injecting an adversarial payload of 4096 bytes between the PE headers and the first section. Second, it extends the slack space between sections by injecting adversarial content into the code caves [30], capping the expansion at 10% of each section's original size to avoid unrealistic binary growth. Third, it creates five new sections filled with benign content extracted from legitimate executables, following the GAMMA strategy [5]. Fourth, it appends an adversarial payload of 10,000 bytes to the overlay [14]. The adversarial content injected is then optimized jointly using a genetic algorithm with the following hyperparameters: (1) the population size (P) is equal to 10; (2) the maximum number of steps (T) is equal to 50; (3) the probability of an individual to be mutated (μ_{ind}) is 30%; (4) the probability of a byte to be mutated (μ_{byte}) is 30%; and (5) we use soft label instead of hard label.

ALGORITHM 4: All-in-One Attack.

Require: Example x , target classifier x , genetic algorithm G , maximum number of steps T , population size P , mutation probability of an individual μ_{ind} , byte mutation probability μ_{byte} .

Ensure: Adversarial example x' such that $g(x') = \text{benign}$, or x if no evasion is found.

```

 $x' \leftarrow x$  ▷ Initialize adversarial EXEmple
 $x' \leftarrow \text{SHIFT}(x', p_{\text{shift}} = 4096)$  ▷ Inject payload between headers and the first section
for each section  $s$  in  $x'$  do
   $x' \leftarrow \text{CODECAVES}(x', s, \alpha = 0.1)$  ▷ Extend code caves by up to 10% of  $|s|$ 
end for
for  $k = 1$  to 5 do
   $x' \leftarrow \text{NEWSECTION}(x', \text{content} \sim \text{EXTRACT}(\text{benign corpus}))$  ▷ Inject new benign section
end for
 $x' \leftarrow \text{OVERLAY}(x', p_{\text{overlay}} = 10000)$  ▷ Append payload to the overlay
 $\mathcal{P} \leftarrow \text{INITPOPULATION}(x', P)$ 
for  $t = 1$  to  $T$  do
  for each individual  $\hat{x}$  in  $\mathcal{P}$  do
     $s(\hat{x}) \leftarrow g_{\text{score}}(\hat{x})$  ▷ Evaluate adversarial fitness score
    if  $g(\hat{x}) = \text{benign}$  then
      return  $\hat{x}$  ▷ Evasion successful
    end if
  end for
   $\mathcal{P} \leftarrow \text{SELECT}(\mathcal{P})$  ▷ Select fittest individuals
   $\mathcal{P} \leftarrow \text{CROSSOVER}(\mathcal{P})$  ▷ Generate offspring via crossover
   $\mathcal{P} \leftarrow \text{MUTATE}(\mathcal{P}, \mu_{ind}, \mu_{byte})$  ▷ Mutate adversarial payload bytes
end for
return  $\hat{x}$  ▷ Return best candidate if no evasion found within  $T$  steps

```

Under the threat model of Theorem 3, this attack instantiates $K = |S|$ patch components and $M = 1(\text{shift}) + |S|(\text{code caves}) + 5(\text{section injection}) + 1(\text{overlay})$ injection components, where $|S|$ is the number of sections in an executable. The patch components correspond to the overwriting of the existing slack space bytes in each section s , each of size σ_s (the original slack space of section s). The injection components correspond to: (1) the Shift payload of size $q_1 = 4096$; (2) the extension of each section's slack space by up to $0.1 \cdot |s|$ bytes; (3) the five newly created sections filled with benign content, each of size $q_k^{\text{sec}}, k = 1, \dots, 5$; and (4) the overlay payload of size 10,000. The total adversarial budget is therefore:

$$\Delta_{\text{combined}} = 2 \sum_{s \in S} (\lceil \frac{\sigma_s}{z} \rceil + 1) + (\lceil \frac{4096}{z} \rceil + 1) + \sum_{s \in S} (\lceil \frac{0.1 \cdot |s|}{z} \rceil + 1) + \sum_{k=1}^5 (\lceil \frac{q_k^{\text{sec}}}{z} \rceil + 1) + (\lceil \frac{10000}{z} \rceil + 1),$$

as defined in Equation (10).

The pseudocode for the all-in-one attack is provided in Algorithm 4.

A few notes on the pseudocode: The SHIFT, CODECAVES, NEWSECTION, and OVERLAY operations (lines 2–9) construct the full adversarial structure of the file before optimization begins, establishing the injection locations and their maximum capacities. The genetic algorithm (lines 10–20) then optimizes the byte content of the payloads injected at all locations jointly, treating the concatenation of all payload regions as a single optimization variable. The fitness score $g_{\text{score}}(\hat{x})$ corresponds to the soft label output of the classifier.

Received 27 October 2025; revised 8 June 2026; accepted 17 June 2026