SN Computer Science (2024) 5:710
https://doi.org/10.1007/s42979-024-03024-8

ORIGINAL RESEARCH

Certifying Accuracy, Privacy, and Robustness of ML-Based Malware Detection

Nicola Bena[1] · Marco Anisetti[1] · Gabriele Gianini[2] · Claudio A. Ardagna[1]

Received: 29 December 2023 / Accepted: 31 May 2024

Abstract
Recent advances in artificial intelligence (AI) are radically changing how systems and applications are designed and developed. In this context, new requirements and regulations emerge, such as the AI Act, placing increasing focus on strict non-functional requirements, such as privacy and robustness, and how they are verified. Certification is considered the most suitable solution for non-functional verification of modern distributed systems, and is increasingly pushed forward in the verification of AI-based applications. In this paper, we present a novel dynamic malware detector driven by the requirements in the AI Act, which goes beyond standard support for high accuracy, and also considers privacy and robustness. Privacy aims to limit the need of malware detectors to examine the entire system in depth requiring administrator-level permissions; robustness refers to the ability to cope with malware mounting evasion attacks to escape detection. We then propose a certification scheme to evaluate non-functional properties of malware detectors, which is used to comparatively evaluate our malware detector and two representative deep-learning solutions in literature.

Keywords Machine learning · Malware detection · Certification · Accuracy · Privacy · Robustness

Introduction

The widespread and pervasive adoption of ICT technologies is at the basis of today's digital society where every aspect of human life builds on digital technologies. This success attracted (cyber)criminals that increasingly attacked the digital society and its technologies either for political or financial reasons. In this context, cybersecurity has received a lot of attention, involving academic and industrial communities in the protection of the digital society from cyberattacks. Cybersecurity solutions have been influenced by and took advantage of ICT evolution, with artificial intelligence recently gaining ground in many disparate domains [4–7, 35, 42].

According to the ENISA Threat Landscape 2022 [18], malware is one of the most common attack vectors and represents the main cyberattack, with ransoms rising up to $50 M and individual malware infections costing up to $1 M per incident [32]. The fight between security researchers and professionals, who implement new approaches for detecting malware as quickly as possible, and malware developers, who create complex malware using evasive strategies to avoid detection, is increasingly played on a day-to-day basis and with alternating fates.

Modern malware detection applications (malware detectors in the following) replace and complement traditional signature-based detection with static analysis techniques based on machine learning (ML). Static analysis focuses on features that can be extracted from the malware code itself, without executing it. These features include the API (application programming interface)/system calls [22], assembly instructions [26], control flow graphs [23, 31], as well as non-traditional representations such as images [36]. Dynamic analysis complements and gradually substitutes

✉ Nicola Bena
nicola.bena@unimi.it

Marco Anisetti
marco.anisetti@unimi.it

Gabriele Gianini
gabriele.gianini@unimib.it

Claudio A. Ardagna
claudio.ardagna@unimi.it

[1] Department of Computer Science, Università degli Studi di Milano, Via Celoria 18, 20133 Milan, Italy

[2] Dipartimento di Informatica, Sistemistica e Comunicazione (DISCo), Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126 Milan, Italy

Published online: 11 July 2024

static analysis, starting from the assumption that the malware run-time behavior cannot be easily changed. In this case, the malware is executed in a sandbox and its behavior is observed and analyzed. Features include the sequence of invoked system calls [39, 43, 47], process- [2], and network flow-level data [13], to name but a few. Hybrid analysis combines the two approaches.

Existing approaches show excellent performance (e.g., accuracy ≥ 0.99), though they suffer from two main drawbacks. On the one hand, the rise of evasion attacks (often known as *adversarial attacks*) has demonstrated that existing malware detectors can be easily bypassed. Even worse, these attacks are feasible in the real world, meaning that the adversarial perturbation is applicable to the malware source code/executable file and does not change the malware behavior (e.g., [17, 44]). Being malware detection an adversarial environment, the lack of robustness against these attacks is causing increasing concerns, questioning the practical usability of malware detectors in the real world. On the other hand, granting malware detectors permission to collect the necessary data often requires the company producing the detectors to have complete control over the monitored system. Users may be reluctant to grant such permissions to third parties because they can violate their company's privacy policies and increase the risk that such detectors can be used as an attack vector themselves.[1]

Furthermore, existing approaches to malware detection conflict with recent guidelines and regulations on artificial intelligence, which increasingly point to ethics, privacy, and robustness. In particular, the European Parliament has recently approved the Artificial Intelligence Act (referred to as the AI Act), which emphasizes the departure from AI solutions where accuracy is the sole concern. [49]. Rather, the AI Act mandates AI solutions to be ethical, transparent, robust, secure, and privacy-preserving, and certified to prove the continuous support of these properties.

In this paper, we extend our previous work in [11] to fill in the above needs. We first develop an approach to malware detection that targets properties accuracy, privacy, and robustness. To this aim, our approach relies on easily accessible system-level performance (CPU, RAM, and I/O usage) data and requires low-level permissions. It creates an initial dataset modeling data points as multi-valued time-series. It augments the dataset and fully exploits the extracted features using an LSTM (long short-term memory) network, a model capable of dealing with temporal information, achieving 0.99 accuracy. We then propose a preliminary certification scheme for evaluating non-functional properties of malware detectors. The scheme is used to compare the proposed approach with two representative deep-learning solutions in literature, a static detector [41] and a hybrid detector [44], according to the following properties: *accuracy, privacy by collected data and access permissions minimization, and robustness against evasion attacks*.

The remainder of the paper is structured as follows. Section "Motivations" discusses the motivations at the basis of our work. Section "Non-Functional Properties" presents the three target non-functional properties supported by our malware detector [11] in Sect. "Lightweight Malware Detection". Section "A Certification Scheme for Malware Detectors" describes the certification scheme used to verify the three properties. Section "Certification Results" presents a comparative evaluation of three malware detectors (including the one in this paper) based on certification. Section "Discussion" discusses our findings. Section "Related Work" presents related work. Finally, Sect. "Conclusions" draws our conclusions.

Motivations

On March 13th, 2024, the European Parliament approved the AI Act, the first worldwide law regulating AI systems.[2] The AI Act adopts a risk-based approach, requiring AI systems to satisfy different (non-functional) requirements according to the risk the system entails.[3] For instance, systems with *unacceptable risk* contravening EU values are prohibited, while systems with *high risk* must be "*subject to a conformity assessment*" to demonstrate compliance with requirements such as "*accuracy, cybersecurity, and robustness*" [14]. Even *minimal-risk* AI systems, which would only need to comply with basic transparency obligations, may voluntarily comply with such requirements [14] and thus increase their safety, trustworthiness, and market legitimacy [28]. In addition to the AI Act, many other recommendations are proliferating, for instance, the NIST's Artificial Intelligence Risk Management Framework (NIST AI RMF) [38].

The AI Act represents a paradigm shift in the AI domain, where *demonstrable compliance to non-functional requirements is mandated by law*.

In this paper, we start from the AI Act and apply its statements in the domain of malware detection built on artificial intelligence. Traditionally, malware detectors are defined to maximize accuracy (achieving remarkable accuracy ≥ 0.99), while often disregarding other requirements (e.g., on their robustness and privacy) recently mandated by the AI Act. In general, ML-based malware detectors can be classified as *static*, *dynamic*, and *hybrid*. Static detectors

[1] https://www.ccleaner.com/knowledge/security-notification-ccleaner-v5336162-ccleaner-cloud-v1073191.

[2] https://data.consilium.europa.eu/doc/document/ST-5662-2024-INIT/en/pdf.

[3] https://www.europarl.europa.eu/RegData/etudes/BRIE/2021/698792/EPRS_BRI(2021)698792_EN.pdf.

analyze the executable file without executing it. For instance, they extract features such as API/system calls and assembly instructions [22, 26], control-flow graphs [23, 31], and even image representation [3, 25, 36, 50]. Dynamic detectors, instead, observe the run-time malware behavior during its execution. They consider features such as, for instance, sequence of system calls [39, 43], process-level information [2], and image representation [15, 47]. Finally, hybrid detectors fuse static and dynamic analysis, including additional information such as metadata of the executable file [30, 33].

The goal of this paper is to define a novel malware detector following the requirements of the AI Act. In particular, we design a malware detector targeting the support of a variety of non-functional properties that go beyond vanilla accuracy and also embraces privacy and robustness. To this aim, we define properties accuracy, privacy, and robustness, (Sect. "Non-Functional Properties"), driving the development of our malware detector (Sect. "Lightweight Malware Detection"). We further introduce the certification scheme for malware detector certification (Sect. "A Certification Scheme for Malware Detectors") and apply it to the detector in this paper and two additional detectors in literature (Sect. "Certification Results").

Non-Functional Properties

We define the non-functional properties accuracy, privacy, and robustness driving the definition of a malware detector following the requirements in the AI Act.

Property accuracy models the need of retrieving accurate results by malware detectors in operation. It is a standard property for malware detection, conventionally assumed to take value in [0 , 1].

Property privacy models the need to minimize the intrusiveness and the amount of data collected by the detector (*data minimization principle*[4]). The property refers to the data collected for training and inference, such as data referred to individual processes (higher intrusiveness) or to the system as a whole (lower intrusiveness). In turn, data collection requires a variety of specific permissions to be granted to the malware detector, which are also important to minimize (*data protection by design and by default*[5]).

Property robustness models the need to protect the malware detector against malware that actively attempts to escape ML classification by injecting *adversarial perturbations* [46]. Robustness can be supported in different configurations and strengths. Figure 1 shows an excerpt of the *hierarchy* for property robustness; grey-filled boxes denote

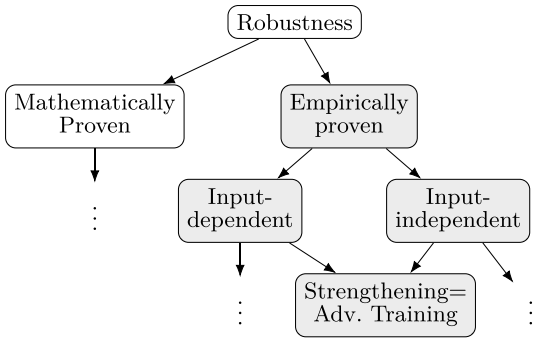

Fig. 1 Hierarchy of property robustness (excerpt). Grey-filled boxes denote the properties on which the present paper focuses

the focus of this paper. In some cases, robustness can be *mathematically proven* (e.g., [24, 44, 45]), meaning that it is possible to compute a bound on prediction correctness as a function of the extent of the adversarial perturbation (this is also known as *certified robustness*). Robustness can also be *empirically proven* (the focus of this paper), that is, evaluated using, for instance, testing procedures (e.g., [16, 21, 48]). The latter approach can assume two forms (corresponding to as many properties): *(i) input-dependent*: robustness is guaranteed by the fact that it is extremely difficult for an attacker to perturb the data points in practice (this property applies for instance in case access to the entire system is required before infection can be spread); *(ii) input-independent*: the attacker can execute the perturbation in the real world (e.g., in case the attacker needs to perturb the malware executable only). Finally, empirically-proven robustness can be further refined by specific strengthening techniques (e.g., *adversarial training* [46]).

Table 1 shows how related work in malware detection (discussed in detail in Sect. "Related Work") supports properties accuracy, privacy, and robustness. ✓ means that the property is fully supported, ≈ partially supported, ✗ not supported. ✗ is also used when the property is not discussed/evaluated. As a matter of fact, all malware detectors target property accuracy, while just a few consider property robustness. Property privacy instead is typically neglected. To the best of our knowledge, *no malware detectors simultaneously focus on accuracy, privacy, and robustness*: this reduces their real-world applicability and falls short in providing AI Act compliance.

Furthermore, we emphasize a common fact for several published works: simply claiming that a malware detector supports a given set of properties is insufficient; such claims must be substantiated. For instance, the AI Act requires some AI systems to run a "conformity assessment procedure" and, in some cases, be subject to "state-of-the-art tests and models evaluations" [14]. Similarly, the NIST AI RMF requires AI systems to be tested regularly [38].

[4] https://eur-lex.europa.eu/eli/reg/2016/679/oj, Art. 5.

[5] https://eur-lex.europa.eu/eli/reg/2016/679/oj, Art. 25.

Table 1 Comparison of malware detectors with respect to properties accuracy, robustness, and privacy

References	Type	Accuracy	Privacy	Robustness	
				Math.	Emp.
[27]	Static	✓	✗	✗	✗
[36]	Static	✓	✗	✗	✗
[43]	Dynamic	✓	✗	✗	✗
[22]	Static	✓	✗	✗	✗
[33]	Hybrid	✓	✗	✗	✗
[34]	Dynamic	✓	✗	✗	✗
[21]	Static	✓	✗	✗	✓
[25]	Static	✓	✗	✗	✗
[26]	Static	✓	✗	✗	✗
[50]	Static	✓	✗	✗	✗
[2]	Dynamic	✓	✗	✗	✗
[15]	Dynamic	✓	✗	✗	≈
[31]	Static	✓	✗	✗	✗
[39]	Dynamic	✓	✗	✗	✓
[30]	Hybrid	✓	✗	✗	✗
[16]	Static	✓	✗	✗	✓
[13]	Dynamic	✓	✗	✗	✗
[48]	Dynamic	✓	✗	✗	✓
[23]	Dynamic	✓	✗	✗	✗
[47]	Dynamic	✓	✗	✗	✗
[3]	Static	✓	✗	✗	✗
[19]	Static	✓	✗	✓	✗
[24]	Static	✓	✗	✓	✗
[45]	Static	✓	✗	✓	✗
This	Dynamic	✓	✓	✗	✓

To address the aforementioned gaps, researchers and practitioners are clearly pointing to *certification* [10, 12], as a way to demonstrate (*certify*) that the given *target* (e.g., a malware detector) supports a given *property* (e.g., robustness), backed by some *evidence* (e.g., testing, mathematical proofs) [12]. The objective of this paper is therefore twofold:

- design and implement an ML-based malware detector that *jointly supports and balance high detection accuracy, privacy,* and *(empirically proven) robustness* (last row in Table 1); and
- adopt a certification scheme for AI [10] to demonstrate the non-functional properties of different malware detectors, including the one in this paper.

Lightweight Malware Detection

Figure 2 shows an overview of our approach to lightweight malware detection introduced in [11] and driven by the properties in Sect. "Non-Functional Properties", which minimizes the collected data and requires low permissions for execution. First, it creates a sandbox where an initial dataset of legitimate and malicious software executions is collected (Sect. "Sandbox Implementation"). The dataset contains system-level performance metrics in the form of time-series. Second, it augments the collected dataset using a generative adversarial network (GAN) (Sect. "Dataset") to meet the requirements of modern deep learning (DL) models. Finally, an LSTM model is trained to fully exploit the temporal structure of our dataset (Sect. "LSTM Model"). The LSTM model drives the behavior of our malware detector.

Sandbox Implementation

Running malware to analyze its behavior is fundamental to design a malware detector, although it introduces the risk of self-infection. The use of an isolated environment where the malware can be safely executed (sandbox) can mitigate or remove this risk. This approach is not always feasible, because some malware may be able to understand whether they are running inside a sandbox. When this happens, they may change their behavior or interrupt their execution; advanced malware may even escape the sandbox, causing the infection of the system where the sandbox is installed.

For this reason, we used a combination of Linux and Windows machines as shown in Fig. 3. Specifically, we tested malware and legit software on a Windows 7 virtual machine (VM) hosted by a Linux machine. The Windows VM is isolated from the Internet using a *host-only connection*. This way, the VM does not have access to the physical network card of the host machine, preventing any malware connections to the Internet.

Executing malware on a machine that cannot communicate on the Internet, however, has some limitations; for instance, some malware need to connect to remote hosts to carry out their activities (e.g., Wanna-Cry). We then set up a second Linux VM running *iNetSim* (https://www.inetsim.org/), a software to simulate Internet connections. The malware executed on the Windows VM can send Internet requests and obtain corresponding responses without reaching the Internet. We note that, to permit an effective execution of the malware inside the Windows VM, all protection controls like firewalls, Windows update, and Windows defender have been disabled and group policies

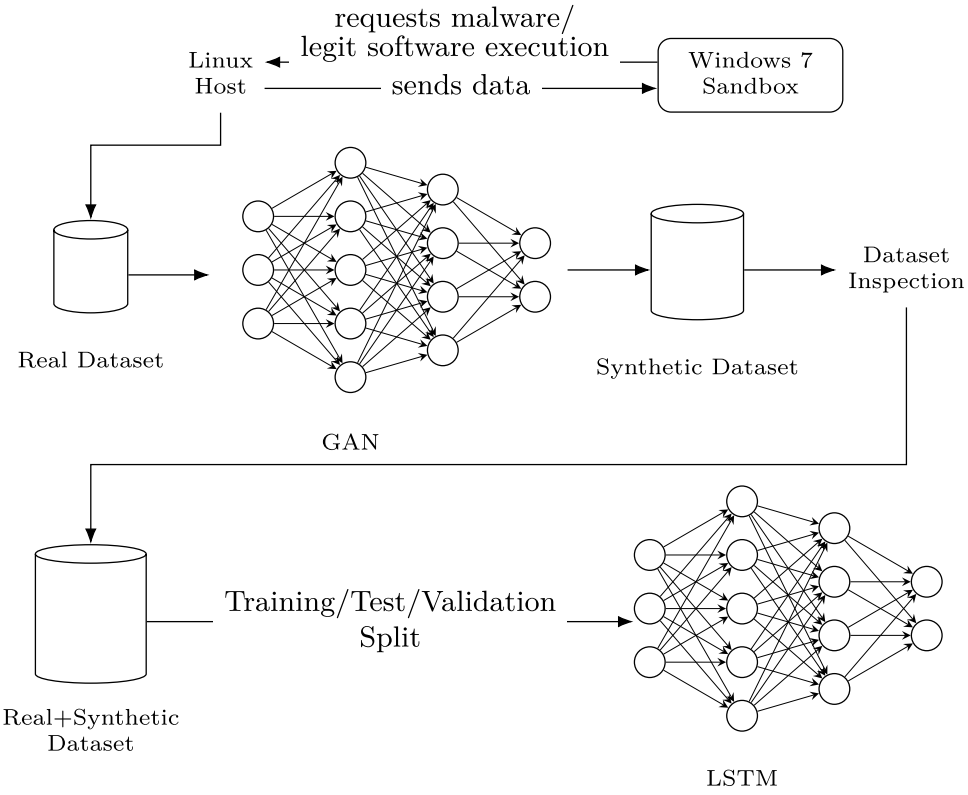

Fig. 2 Overview of our approach

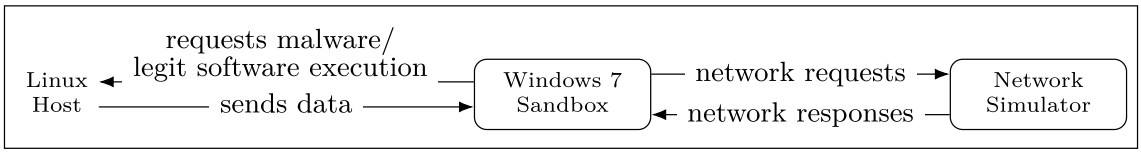

Fig. 3 The sandbox

have been changed to give the malware the capability to act as an administrator. The need to modify these policies motivates the use of an old Windows version, namely Windows 7.

Dataset

We generated the dataset used for training in two phases as follows. Phase dataset creation (Sect. "Dataset Creation") collects an initial dataset of legit and malware executions. Phase dataset augmentation (Sect. "Dataset Augmentation") augments the initial dataset using a GAN.

Dataset Creation

Phase dataset creation starts from the configuration of the Window 7 VM. It first installs commonly used software (e.g., Internet Explorer, Firefox, Mozilla Thunderbird, Spotify, WinRAR) on the Windows VM, to make the environment as realistic as possible. It then retrieves the ≈ 5000 malware PE files compatible with Windows 7 from *VirusShare* (https://virusshare.com).

Malware and legit software are executed for a fixed amount of time while collecting performance metrics. In this paper, we considered a time span of 60 s[6] and ran 10,000 executions varying between malware and legit software. At each execution, the Windows VM is restored from a clean snapshot (following the state of the art [34]) and the chosen software is run for the given time span. During each execution, we collected the multi-valued time-series consisting of 6 features: *(i)* CPU usage percentage, *(ii)* RAM usage percentage, *(iii)* bytes written to and *(iv)* bytes read from the disk, *(v)* bytes received and *(vi)* sent to the network. The

[6] Experiments have been run on a laptop featuring 8 CPUs Intel i7-11800H @ 2.30 GHz, 16 GBs of RAM, operating system Ubuntu 22.04.1. Virtualization based on VirtualBox.

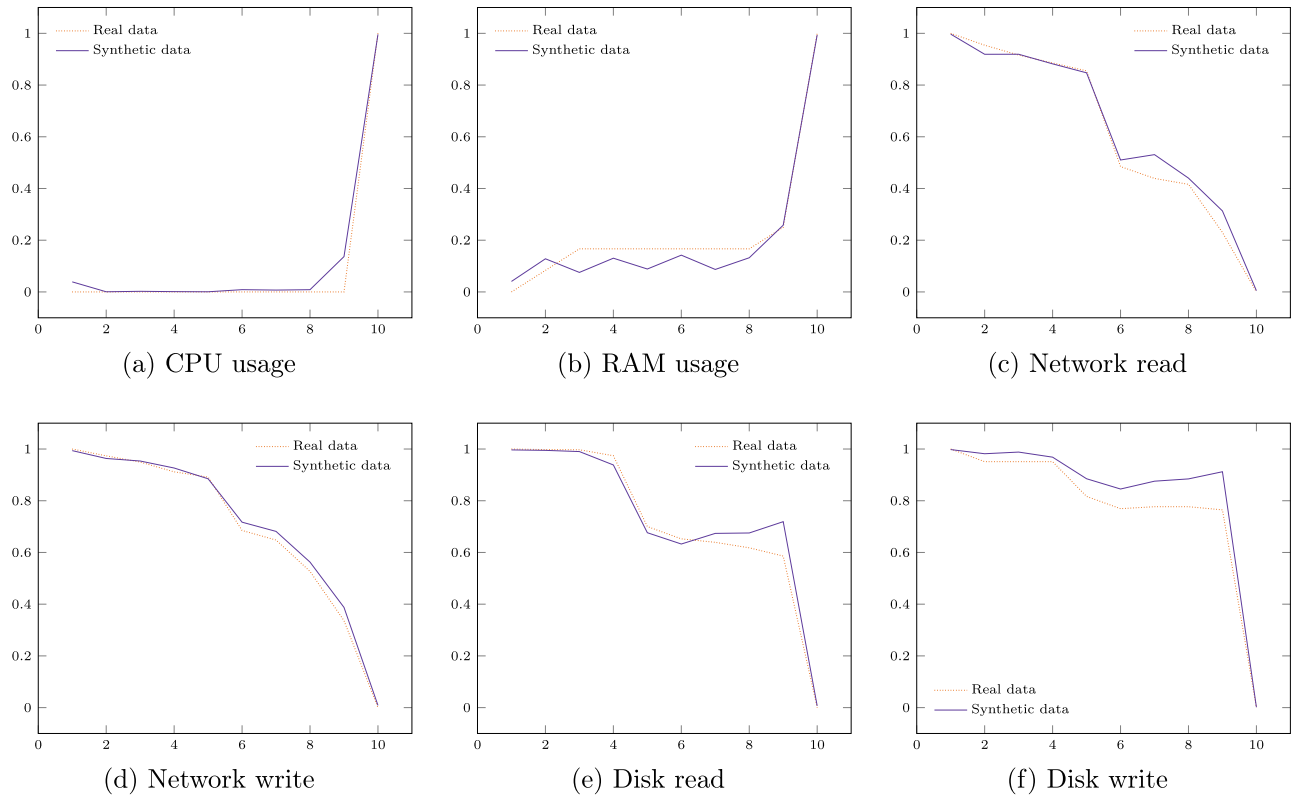

Fig. 4 Comparison of features value of real and synthetic malware samples

collected data are sent back to the Linux host where they are saved.

We note that the use of an LSTM model requires all time-series to have the same length. We then preprocessed the collected data normalizing the time series to a fixed length, by padding the shorter time series and pruning the longer ones. Each resulting time-series contained 10 items each associated with the 6 aforementioned features. Having a time span of 60 s, the sampling time was of 6 s. Although this time is slightly shorter than similar approaches [2], it proved to be effective.

Dataset Augmentation

Phase dataset augmentation aims to support the peculiarities of DL, requiring a large number of training samples to be effective. It augments the dataset created in Sect. "Dataset Creation" using synthetic data so that they show the same statistical properties of real-world data.

Our dataset of ≈ 10,000 samples was augmented using *TimeGAN* (https://pypi.org/project/ydata-synthetic/) [51], a GAN specifically designed to generate time-series data. It extends the traditional GAN architecture, and includes *(i)* a generator implemented as a recurrent network, *(ii)* a discriminator implemented as a bidirectional recurrent network with a final feedforward layer, *(iii)* two additional components called *embedding* and *recovery* functions, and *(iv)* two specific loss functions.

Embedding and recovery functions are implemented as recurrent and feedforward networks, respectively, and map the time-series features to a low-dimensional space where the generator and discriminator operate. Finally, loss functions jointly ensure that the generator learns realistic sequences with accurate temporal patterns.

We first fed the dataset created in Sect. "Dataset Creation" to the GAN so that the model could learn its statistical characteristics and replicate them in the synthetic data. To this aim, *(i)* we separated the real dataset into malware and legit software, and sent each individual dataset to a separate instance of TimeGAN; *(ii)* we generated two synthetic datasets of 50,000 samples each, later merged in a single dataset of 100,000 samples (tenfold increase).

We then validated the quality of the synthetic dataset according to several comparisons as follows.

- *Visual feature comparison:* we randomly drew samples from real and synthetic datasets. For each feature and extracted sample, we plotted their value to visually

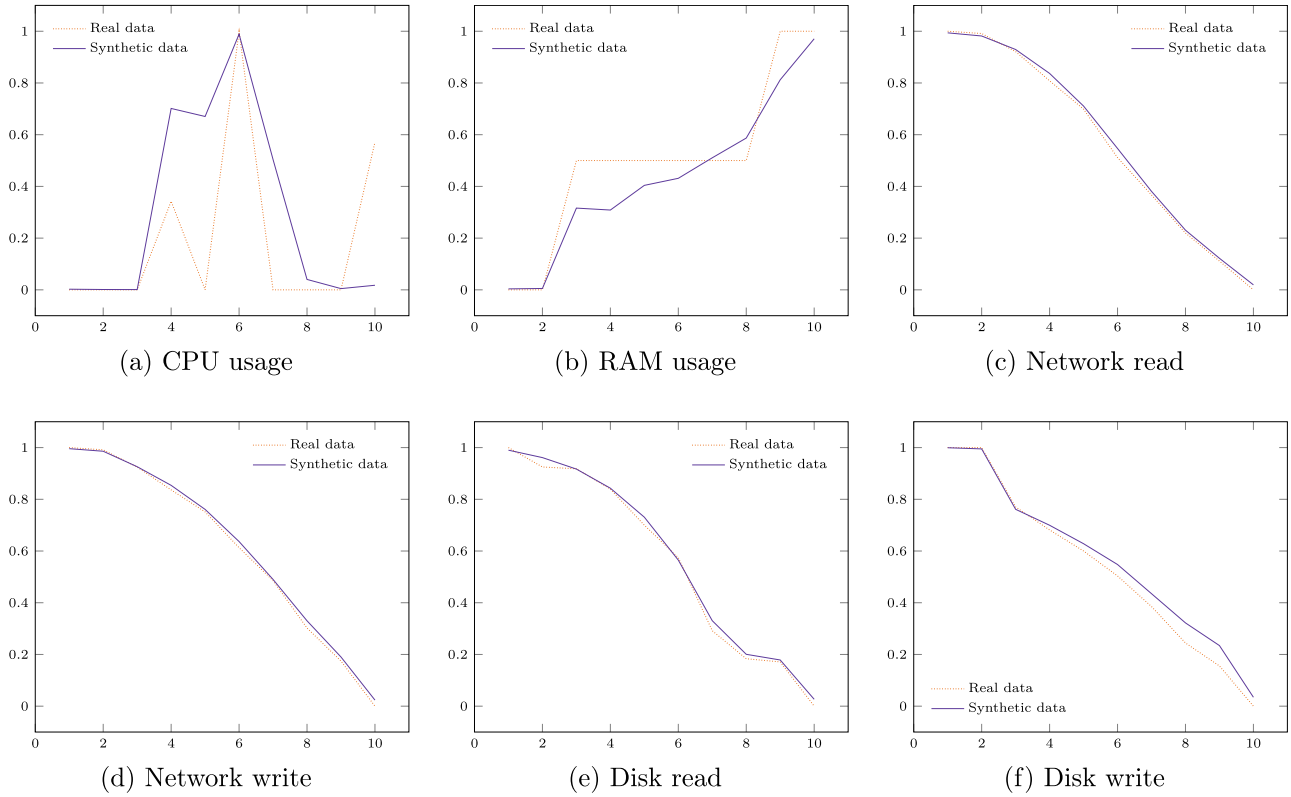

Fig. 5 Comparison of features value of real and synthetic legit software samples

compare the differences between the real and synthetic samples. Figures 4 and 5 show the similarity of two random samples of malware and legit software, respectively.

- *Comparison with reduced dimensionality (PCA):* we performed PCA reduction to a 2-dimensional space in real and synthetic datasets (limited to 500 samples), and plotted the results for visual comparisons.
- *Comparison with reduced dimensionality (t-SNE):* we performed t-SNE reduction to a 2-dimensional space on real and synthetic datasets (limited to 500 samples). Compared to PCA, t-SNE performs a non-linear transformation. We plotted and visually compared the results.

For brevity, we do not report the visual comparison using PCA and t-SNE here, and we refer the interested readers to our original paper in [11].

We finally created the overall dataset by merging the real and the synthetic datasets.

LSTM Model

Table 2(a) describes the structure of the LSTM model we trained on the overall dataset. It is composed of 4 layers (3 LSTM layers and 1 dense layer) interleaved with 3 batch normalization layers.

Table 2(b) describes the parameters of the training process. We used 64,871 samples for the training set and 21,624 samples for the validation and test sets in 200 epochs with optimizer *Adam*, loss function *binary cross-entropy*, and initial learning rate of 0.05. Model training is based on early stopping (if loss function value retrieved from the validation set does not improve in 30 epochs) and on dynamic decrease of the initial learning rate (by a factor of 0.5 if loss function value retrieved from the validation set does not improve in one epoch). Further details can be found in our public code available at https://doi.org/10.13130/RD_UNIMI/LJ6Z8V.

A Certification Scheme for Malware Detectors

Recalling Sect. "Motivations" and the AI Act, AI-based applications should exhibit verifiable behavior in terms of non-functional properties. Here, we adopt certification [12] as a suitable technique for verifying the behavior of malware detectors. In the following, we describe how a generic certification scheme for AI-based applications works (Sect. "Certification in a Nutshell"), and instantiate it to certify our

Table 2 Details of the LSTM training process

(a) LSTM model structure		
Layer type	Output shape	# Params
LSTM	(None, 10, 8)	480
Batch normalization	(None, 10, 8)	32
LSTM	(None, 10, 8)	544
Batch normalization	(None, 10, 8)	32
LSTM	(None, 8)	544
Batch normalization	(None, 8)	32
Dense	(None, 1)	9
(b) Training parameters		
	Parameter	Value
Epochs	200	
Batch size	32	
Optimizer	Adam	
Learning rate	0.05, halved if loss does not improve in 1 epoch, down to $1 \cdot 10^{-8}$	
Early stopping	Loss does not improve in 30 epochs	
Loss function	Binary crossentropy	

malware detector in Sect. "Lightweight Malware Detection" and two malware detectors in literature against properties *accuracy*, *privacy*, and *robustness* (Sect. "Certification Models").

Certification in a Nutshell

A certification scheme implements a certification process proving that a *non-functional property p* (e.g., empirically-proven, input-dependent robustness in Fig. 1) is supported by a given *target of certification ToC* (e.g., a malware detector) by collecting *evidence ev* according to an *evidence collection model* $\mathcal{E}$. *p*, *ToC*, and $\mathcal{E}$ define a *certification model* $\mathcal{M}$. An evaluation function $\mathcal{F}$ completes $\mathcal{M}$, determining whether a certificate $\mathcal{C}$ can be awarded for *ToC* on the basis of collected evidence *ev* (Fig. 6) [12]. $\mathcal{M}$ is prepared by a trusted third party (e.g., a certification authority—CA) and executed by an accredited lab on its behalf. We note that a certificate contains a reference to the corresponding certification model $\mathcal{M}$ and evidence *ev* supporting its release.

ML-based application certification (ML certification in the following) is based on a *multi-dimensional evaluation*, where different facets (*dimensions*) of the corresponding ML model are independently evaluated according to their peculiar life cycle. Each dimension *d* has a specific certification model $\mathcal{M}_d$ [9, 10] containing all the information needed to evaluate the application based on ML in the given dimension *d*. According to our previous work [10], ML certification must consider three dimensions: *(i) data* (d_d) related to the data used to train and test the ML model, *(ii) process* (d_p) related to the process used to train, test, and deploy the ML model, *(iii) model* (d_m) related to the ML model in operation.

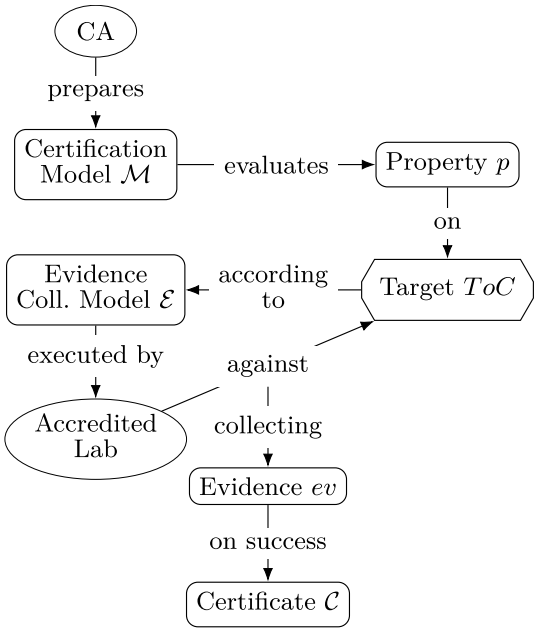

Fig. 6 Certification process [12]

Differently from traditional certification, in multi-dimensional ML certification, *(i)* the certification model $\mathcal{M}_d$ in each dimension *d* defines an evaluation function $\mathcal{M}_d.\mathcal{F}$ indicating whether evidence *ev* is successfully collected in the given dimension *d*; *(ii)* a global evaluation function $\mathcal{F}'$ aggregates the result of $\mathcal{M}_d.\mathcal{F}$ in each dimension, finally resulting in a certificate award *iff* $\mathcal{F}' = \checkmark$.

We note that $\mathcal{F}'$ is defined by the CA according to the specific scenario. In our scenario, $\mathcal{F}' = \mathcal{M}_{d_d}.\mathcal{F} \vee \mathcal{M}_{d_p}.\mathcal{F} \vee \mathcal{M}_{d_m}.\mathcal{F}$, meaning that a certificate is awarded when the evidence is successfully collected in at least one dimension. We note that an independent certificate can also be awarded for each dimension according to $\mathcal{F}$, depending on the scenario.

Certification Models

We define one certification model for each property of interest: *(i) accuracy* (Sect. "Property Accuracy"), *(ii) privacy* (Sect. "Property Privacy"), and *(iii) robustness* (Sect. "Property Robustness").

Each certification model considers different dimensions according to the property.

Property Accuracy

Malware detectors *must* exhibit a high detection accuracy. We define a certification model $\mathcal{M}_{d_m} = \langle\, p, ToC, \mathcal{E}, \mathcal{F} \,\rangle$ considering dimension model (d_m) (Table 3(a)). Property accuracy is defined as *high detection accuracy*. The target of certification $\mathcal{M}.ToC$ is the trained malware detector.

Evidence collection model $\mathcal{M}_{d_m}.\mathcal{E}$ analyzes the required data to retrieve the corresponding metrics.

Formally, let ACC_j (AUC_j, resp.) be the accuracy (area under curve–AUC, resp.) retrieved from the j-th malware detector on a held-out test set. Evaluation function $\mathcal{M}_{d_m}.\mathcal{F}$ defines that $\mathcal{M}.p$ is supported by the j-th detector iff

$$ACC_j \geq t^{\text{acc}} \vee AUC_j \geq t^{\text{acc}} \tag{1}$$

In our case, $t^{\text{acc}} = 0.96$. We note that other quality metrics (e.g., recall) can be considered depending on the scenario.

Property Privacy

Malware detectors must examine the system in depth, from reading the content of all files to observing the behavior of all processes. Granting these permissions may be undesired (e.g., for internal policies or to reduce the attack surface). Thus, a detector *must* minimizes the data it collects and the permissions it requires. Property privacy models this need.

We define a certification model $\mathcal{M}_{d_d} = \langle\, p,\ ToC,\ \mathcal{E},\ \mathcal{F} \rangle$ considering dimension data (d_d) (Table 3(b)). Property privacy $\mathcal{M}_{d_d}.p$ is defined as the *minimization of the collected data and the access permissions necessary for their collection*. The target of certification $\mathcal{M}_{d_d}.ToC$ represents the input data used for training/inference and the permissions needed for their collection.

We model the data collected in terms of the *input space* $\mathcal{I}$ where malware can operate. $\mathcal{I}$ is composed of *(i)* `executable-file` denoting the executable file, *(ii)* `process-performance` denoting process-level performance metrics, *(iii)* `system-performance` denoting system-level performance metrics, and *(iv)* `syscall` denoting the observed system calls of each process. We note that additional data in $\mathcal{I}$ are omitted for brevity.

Let us denote with $\mathcal{I}_j$ the input space required by the j-th malware detector, and Inv the function taking as input a component `input`$\in \mathcal{I}$ of input space $\mathcal{I}$ and returning as output a qualitative score as follows. The qualitative score is 1 when `input` refers to system-level data, 2 to process-level data; it is increased by 1 if the detector needs administrator-level permissions to collect `input`.

Evidence collection model $\mathcal{M}_{d_d}.\mathcal{E}$ analyzes the required data to retrieve the corresponding scores.

Evaluation function $\mathcal{M}_{d_d}.\mathcal{F}$ defines that $\mathcal{M}_{d_d}.p$ is supported by the j-th detector iff

$$\sum_{\texttt{input}_i \in \mathcal{I}_j} \text{Inv}(\texttt{input}_i) \leq t^{\text{pr}} \tag{2}$$

In other words, property privacy is supported if the sum of the qualitative scores is below threshold t^{pr}. In our case, t^{pr} =2.

Property Robustness

Malware detectors *must* identify malware that actively attempts to escape classification by exploiting the vulnerabilities of the detectors, possibly caused by the peculiarities of ML [46]. We consider empirically proven robustness (Sect. "Non-Functional Properties") and focus on ML-specific *evasion attacks*, perturbing a data point at inference time by adding an imperceptible perturbation such that the predicted label changes from "malware" to "benign". We define two certification models $\mathcal{M}_{d_p}$ and $\mathcal{M}_{d_m}$ for dimensions process (d_p) and model (d_m), respectively, as follows.

Dimension process d_p defines a certification model $\mathcal{M}_{d_p} = \langle\, p, ToC, \mathcal{E}, \mathcal{F} \,\rangle$ (Table 3(c)). Property robustness $\mathcal{M}_{d_p}.p$ is defined as *input-dependent* or *input-independent robustness with strengthening technique adversarial training* in Fig. 1. Adversarial training adds evasion data points with the correct label to the training set, such that the trained ML model learns how to spot the imperceptible perturbations of an evasion attack [46]. The target of certification $\mathcal{M}_{d_p}.ToC$ represents the training process.

Evidence collection model $\mathcal{M}_{d_p}.\mathcal{E}$ collects evidence from the training process ($\mathcal{M}_{d_p}.ToC$).

Evaluation function $\mathcal{M}_{d_p}.\mathcal{F}$ defines that $\mathcal{M}_{d_p}.p$ is supported iff at least 0.01% of adversarial training-created data points with label "malware" are added to the training set. We note that the percentage of adversarial training/created data points is taken from Grosse et al. [21].

Dimension model d_m defines a certification model $\mathcal{M}_{d_m} = \langle\, p, ToC, \mathcal{E}, \mathcal{F} \,\rangle$ (Table 3(d)). The target of certification $\mathcal{M}_{d_m}.ToC$ represents the ML model. Depending on the detector, property robustness can be supported at different levels in the hierarchy in Fig. 1, varying also $\mathcal{M}_{d_m}.\mathcal{E}$ and $\mathcal{M}_{d_m}.\mathcal{F}$.

Input-dependent detector. Property robustness $\mathcal{M}_{d_m}.p$ is defined as *empirical, input-dependent robustness* in Fig. 1. It refers to the need to control the entire system and its processes to execute an effective perturbation. Evidence collection model $\mathcal{M}_{d_m}.\mathcal{E}$ analyzes the ML model

(e.g., software artifacts) retrieving the type of data the ML model receives as input. Evaluation function $\mathcal{M}_{d_m}.\mathcal{F}$ defines that $\mathcal{M}_{d_p}.p$ is supported iff the only way to successfully perturb a data point is to have complete access to the victim system. We note that this scenario is unrealistic. Let us assume that a malware can obtain control of the victim system and *then* execute the perturbation allowing itself to evade classification. A malware detector would be able to catch the malware *before* the latter can hide itself with the perturbation.

Input-independent detector. Property robustness $\mathcal{M}_{d_m}.p$ is defined as *empirical, input-independent robustness* in Fig. 1. Formally, let *(i)* $\{p_i\}$ be a sequence of data points labeled as "malware"; *(ii)* $\mathcal{A}$ be a function crafting evasion data points, which takes as input the sequence $\{p_i\}$ of data points and returns as output a sequence $\{\widetilde{p_i}\}$ of perturbed data points; $y(p_i)$ be the predicted label for data point p_i. Evidence collection model $\mathcal{M}_{d_m}.\mathcal{E}$ exercises the ML model, sending evasion data points according to $\mathcal{A}$ and retrieving the predicted label. Evaluation function $\mathcal{M}_{d_m}.\mathcal{F}$ defines that $\mathcal{M}_{d_m}.p$ is supported if

$$\frac{|\{\widetilde{p_i} \mid y(\widetilde{p_i}) = \text{“benign”}\}|}{|\{\widetilde{p_i}\}|} \leq t^{\mathrm{r}} \tag{3}$$

In other words, property robustness is supported if the number of evasion data points that evade the classifier is below the threshold t^{r}. In our case, $t^{\mathrm{r}} = 0.1$, which means that at most 10% of the evasion data points can evade classification. We note that a tighter threshold can be fixed according to the scenario.

Certification Results

Additional Malware Detectors

We present the two detectors in literature, which have been certified in our experimental evaluation according to the certification models in Sect. "A Certification Scheme for Malware Detectors". Together with ours in Sect. "Lightweight Malware Detection", these three detectors are a good approximation of the entire domain of malware detection.

Static Malware Detector

Static detector DS considers *MalConv*, a convolutional neural network presented in 2017 by Raff et al. [41]. It is the first approach to fully exploit the power of deep learning, as it takes as input the *executable file as is*, without any preprocessing. We refer to the implementation by Anderson et al. [8] which is publicly available.

Input. Each data point is the executable file to analyze. The file size is fixed to 1 MB. Larger files are truncated, and smaller files are padded with a special value.

ML model. DS implements a convolutional neural network structured as follows. The first layer is an embedding layer mapping each byte to a 8-dimensional vector. The next two layers implement a convolution followed by a pooling layer. The last layer is a fully connected layer.

Training. DS is trained on the executable files at the basis of dataset *EMBER*, a dataset containing features of more than 1 million of benign and malign software [8].

Hybrid Malware Detector

Hybrid detector (DH) considers the solution presented by Rosenberg et al. [44]. It works with both dynamic (i.e., *n*-grams of observed system calls) and static (i.e., strings found in the executable file) features.

Input. Each data point contains: *(i)* a one-hot encoded vector where each *i*-th feature represents the presence of a system call (formally, Windows API call) in a fixed-size sequence of system calls; *(ii)* a one-hot encoded vector where each *i*-th feature represents the presence or absence in the executable file of the *i*-th string among the top-20,000 most frequent strings.

ML model. DH implements a custom, two-branch architecture. The first branch consists of an LSTM layer taking as input the sequence of system calls. The second branch consists of two fully connected layers, taking as input the strings. The output of the two branches is flattened and taken as input by the last fully connected layer.

Training. DH is trained on a dataset of 54,000 data points generated by executing different benign and malign software. Each software is run in a sandbox for 2 min and the corresponding system calls are retrieved. The system calls are divided into sliding windows (step 1), each including a *n*-gram with n=140 and the top-20,000 most frequent strings extracted from executable files. Each data point includes a sliding window and a label "benign"/"malign" for the software retrieved using the online service *VirusTotal* (https://www.virustotal.com/).

Results

We present the results of the execution of the certification models in Sect. "A Certification Scheme for Malware Detectors" against malware detectors in this article. We note that evidence on the behavior of DS [8, 41] and

Table 3 Certification models

(a) Property accuracy—dimension d_m

Appl.	Property p	Target *ToC*	Evid. coll. $\mathcal{E}$	Eval. func. $\mathcal{F}$
DS [8, 41]	High detection accuracy	*MalConv* model in [8]	Retrieve metrics from test set	Accuracy or AUC are bounded by t^{acc} (Eq. (1))
DD [11]	High detection accuracy	LSTM model in [11]	Retrieve metrics from test set	Accuracy or AUC are bounded by t^{acc} (Eq. (1))
DH [44]	High detection accuracy	Two-branch model in [44]	Retrieve metrics from test set	Accuracy or AUC are bounded by t^{acc} (Eq. (1))

(b) Property privacy—dimension d_d

Appl.	Property p	Target *ToC*	Evid. coll. $\mathcal{E}$	Eval. func. $\mathcal{F}$
DS [8, 41]	Minimization of the collected data and access permissions for their collection	Dataset EMBER [8] of malware/benign executable files	Dataset structure analysis	Portion of entire input space covered by the used data is bounded by t^{pr} (Eq. (2))
DD [11]	Minimization of the collected data and access permissions for their collection	Real and synthetic dataset of system-level performance metrics [11]	Dataset structure analysis	Portion of entire input space covered by the used data is bounded by t^{pr} (Eq. (2))
DH [44]	Minimization of the collected data and access permissions for their collection	Dataset of malware/benign executable files and observed system calls as n-grams [44]	Dataset structure analysis	Portion of entire input space covered by the used data is bounded by t^{pr} (Eq. (2))

(c) Property robustness—dimension d_p

Appl.	Property p	Target *ToC*	Evid. coll. $\mathcal{E}$	Eval. func. $\mathcal{F}$
DS [8, 41]	Empirical, input-independent robustness with strengthening technique adversarial training	Training process in [8]	Inspect configuration of the training process	Usage of adversarial training with at least 0.01% evasion data points with a given ϵ
DD [11]	Empirical, input-dependent robustness with strengthening technique adversarial training	Training process in [11]	Inspect configuration of the training process	Adversarial training with at least 0.01% evasion data points with a given $\epsilon \in [0.05]$
DH [44]	Empirical, input-independent robustness with strengthening technique adversarial training	Training process in [44]	Inspect configuration of the training process	Adversarial training with at least 0.01% evasion data points with $\epsilon \in [0.0005, 0.0049]$

(d) Property robustness—dimension d_m

Appl.	Property p	Target *ToC*	Evid. coll. $\mathcal{E}$	Eval. func. $\mathcal{F}$
DS [8, 41]	Empirical, input-independent robustness with strengthening technique adversarial training	*MalConv* model in [8]	Perturb the *DOS header* of 60 malware according to $\mathcal{A}$ in a functionality-preserving manner	The number of misclassified data points crafted according to $\mathcal{A}$ is bounded by t^{r} (Eq. (3))
DD [11]	Empirical, input-dependent robustness with strengthening technique adversarial training	LSTM model in [11]	Analysis of the ML model	The only way to successfully perturb a data point is to have complete access to the victim system
DH [44]	Empirical, input-independent robustness with strengthening technique adversarial training	Two-branch model in [44]	Perturb the test set of 36,000 data points according to $\mathcal{A}$, adding system calls (with $\epsilon \in [0.0005, 0.0049]$) and strings in a functionality-preserving manner	The number of misclassified data points crafted according to $\mathcal{A}$ is bounded by t^{r} (Eq. (3))

DH [44] refers to data and results provided in the corresponding publications. The evidence on the behavior of our malware detector DD in Sect. "Lightweight Malware Detection" refers to data collected from the detector in operation. We executed detector DD on an Apple MacBook Pro with 10 CPUs Apple M1 Pro, 32 GBs of RAM, operating system macOS Ventura, Python v3.11.6, and ML libraries *Keras* v2.13.0, *scikit-learn* v1.1.3 [40], *Tensorflow* v2.15.0, *Tensorflow-Metal* v1.1.0, and *Adversarial Robustness Toolbox* v1.16.0 [37]. All artifacts are available at https://doi.org/10.13130/RD_UNIMI/5VTJCC.

Section "Accuracy Evaluation" and Table 5(a) present our results for property accuracy; Sect. "Privacy Evaluation" and Table 5(b) present our results for property privacy; Sect. "Robustness Evaluation" and Table 5(c)–(d) present our results for property robustness.

Accuracy Evaluation

All detectors support $\mathcal{M}_{d_m}.p$ in the dimension model (d_m).

DS and DD achieved the best results in terms of AUC: 0.9981 and 0.9975, respectively. DD also reported ACC = 0.9975. DH achieved slightly lower values in terms of ACC: 0.9694. We note that ACC is slightly higher than ACC retrieved when only dynamic (0.9248) or static features (0.9619) are considered. Table 4 reports additional classification metrics for DD. In particular, precision = 0.9977, that is, DD identifies a malware in almost all cases; recall = 0.9973, that is, virtually all malware are detected by DD; specificity = 0.9976, that is, DD identifies a benign software in almost all cases.

Therefore, the output of the evaluation function $\mathcal{M}_{d_m}.\mathcal{F}$ is ✓ for the three detectors.

Finally, $\mathcal{F}'$ aggregates the output of $\mathcal{M}_{d_m}.\mathcal{F}$ (✓ for DS, DD, and DH). According to $\mathcal{F}' = \mathcal{M}_{d_d}.\mathcal{F} \vee \mathcal{M}_{d_p}.\mathcal{F} \vee \mathcal{M}_{d_m}.\mathcal{F}$ in Sect. "Certification in a Nutshell", the output is ✓ for all detectors. Certificates $\mathcal{C}_{DS}$, $\mathcal{C}_{DD}$, and $\mathcal{C}_{DH}$ are awarded to DS, DD, and DH, respectively. Each certificate is defined as $\langle \mathcal{M}_{d_m}, \{\text{AUC, ACC}\} \rangle$, where $\mathcal{M}_{d_m}$ is the certification model defined for each detector and {AUC, ACC} is the collected evidence.

Privacy Evaluation

Detectors DS and DD support $\mathcal{M}_{d_d}.p$ in the dimension data (d_d). They analyze the executable file (`executable-file` with score= 1) and system-level performance metrics (`system-performance` with score= 1), respectively. Evidence collection was successfull: both scores equal 1, hence below the threshold t^{pr} in $\mathcal{M}_{d_d}.\mathcal{F}$. The output of evaluation function $\mathcal{M}_{d_d}.\mathcal{F}$ is ✓ for the two detectors.

Detector DH does not support $\mathcal{M}_{d_d}.p$. DH needs to *(i)* monitor running processes to collect the system call n-grams (`syscall` with score= 2), which requires administrator-level permissions (score increased by 1); and *(ii)* analyze executable files (`executable-file` with score= 1)

Evidence collection was unsuccessfull: the sum of the scores is 4, hence above t^{pr}. The output of evaluation function $\mathcal{M}_{d_d}.\mathcal{F}$ is ✗.

Finally, $\mathcal{F}'$ aggregates the output of $\mathcal{M}_{d_d}.\mathcal{F}$ (✓ for DS and DD, ✗ for DH). According to $\mathcal{F}' = \mathcal{M}_{d_d}.\mathcal{F} \vee \mathcal{M}_{d_p}.\mathcal{F} \vee \mathcal{M}_{d_m}.\mathcal{F}$ in Sect. "Certification in a Nutshell", the output is ✓ for the first two detectors, ✗ for the last one. Certificates $\mathcal{C}_{DS}$, $\mathcal{C}_{DD}$ are awarded to DS and DD respectively. Each certificate is defined as $\langle \mathcal{M}_{d_d}, \{\text{score}= 1\} \rangle$, where $\mathcal{M}_{d_d}$ is the certification model defined for each detector and {score= 1} is the evidence collected.

Robustness Evaluation

All detectors DS, DD, and DH do not support $\mathcal{M}_{d_p}.p$ in the dimension process (d_p). They do not use adversarial training or any other strengthening techniques. The evidence collection was unsuccessful, and the result of the evaluation function $\mathcal{M}_{d_p}.\mathcal{F}$ is ✗ for all detectors.

Detectors DS and DH do not support $\mathcal{M}_{d_m}.p$ in the dimension model (d_m). For what concerns DS, the evasion attack implemented in $\mathcal{A}$ perturbs the section *DOS header* of the malware executable files [17]. The attack preserves the malware functionality, because the target section is ignored by the operating system but strongly influences classification. Evidence collection was unsuccessful: the ratio of misclassified malware data points was $52/60 = 0.87$ [17], hence above the threshold t^{r} in $\mathcal{M}_{d_m}.\mathcal{F}$. For what concerns DH, the evasion attack implemented in $\mathcal{A}$ perturbs both static (strings in the executable file) and dynamic (observed system calls as n-gram) features. The first feature is perturbed by adding new strings without changing the functionalities of the executable file [21, 44]. The second feature is perturbed similarly, adding system calls to the executable file without changing the overall functionality [44]. Evidence collection was unsuccessful: the ratio of misclassified malware data points was 0.82 [44], therefore above t^{r}. Finally, DD supports $\mathcal{M}_{d_m}.p$ in the dimension model (d_m), since collected evidence (see Sect. "Lightweight Malware Detection") supports the claim that $\mathcal{M}_{d_m}.p$ is input-dependent.

Our experiments also collected evidence mounting an evasion attack against post-processed data points used in DD. The attack implemented in $\mathcal{A}$ perturbs the extracted features (i.e., system-level performance metrics) using *fast gradient sign method* (FGSM) [20]. According to FGSM, features are perturbed maximizing the ML model loss; ϵ

bounds the largest perturbation applicable to a feature. For example, $\epsilon = 0.7$ means that the value of any features changes of ± 0.7 at most. Recalling that the feature values range in our case in [0, 1], ϵ varies in $\{0.01, 0.1\}$ step 0.01 and $\{0.2, 0.9\}$ step 0.1, to maximize diversity. Figure 7 shows the ratio of misclassified data points varying ϵ. We can observe that the ratio of misclassified malware data points in the worst case of $\epsilon \in \{0.09, 0.1, 0.2\}$ was 1.

Therefore, the output of the evaluation function $\mathcal{M}_{d_m}.\mathcal{F}$ is ✗ for DS and DH is ✗, ✓ for DD.

Finally, $\mathcal{F}'$ aggregates the output of $\mathcal{M}_{d_p}.\mathcal{F}$ (✗) and $\mathcal{M}_{d_m}.\mathcal{F}$ (✗ for DS and DH, ✓for DD). According to $\mathcal{F}' = \mathcal{M}_{d_d}.\mathcal{F} \vee \mathcal{M}_{d_p}.\mathcal{F} \vee \mathcal{M}_{d_m}.\mathcal{F}$ in Sect. "Certification in a Nutshell", the output is ✗ for DS and DH and a certificate cannot be released. On the contrary, the output is ✓ for DD and a certificate is released. The certificate is defined as $\langle \{ \mathcal{M}_{d_p}, \mathcal{M}_{d_m} \}, \textit{inspection results} \rangle$, where $\{ \mathcal{M}_{d_p}, \mathcal{M}_{d_m} \}$ are the certification models defined for DD and *inspection results* is the collected evidence.

Discussion

Four main findings emerge from the analysis in this paper.

F1 *Data representation can positively influence detection quality.* Our results show that the high detection performance achieved by DS, DD, and DH, can lie in the way data are represented and features extracted. Static detector DS was a pioneer in deep learning-based malware detection, showing that a high AUC (0.9981, the highest among the approaches considered in this paper) can be achieved without manual feature extraction. However, Anderson et al. [8] showed that shallow learning can be better: a *LightGBM* model achieved AUC= 0.9991, with no fine-tuning but carefully extracted features on the same dataset of DS. Our dynamic approach (DD) sets a new bar for dynamic, lightweight malware detection (ACC= 0.9975). Other approaches achieved lower results with simpler ML models and data representations. For instance, Milosevic et al. [34] considered a larger set of process-level features related to the behavior of individual Android apps (e.g., total CPU usage, number of page faults), modeled as individual samples rather than as time-series. A logistic regression achieved ACC = 0.86 in the best case. Abdelsalam et al. [2] considered set of features similar to the ones in DD, but retrieved at process-level and in individual samples. A Convolutional Neural Network (CNN) achieved ACC $\approx$ 0.97 in the base case. Virtually the same set of features was considered in [1] for anomaly detection, achieving accuracy $\geq$ 0.9 using k-means-based clustering. Finally, when considering hybrid malware detection, the highest accuracy (0.9694) was achieved when static and dynamic features were jointly considered, as discussed in Sect. "Accuracy Evaluation".

Table 4 Additional metrics for DD

Metric	Value
AUC	0.9975
Accuracy	0.9975
Recall	0.9973
Precision	0.9977
F1	0.9975
Specificity	0.9976

F2 *Data preparation increases detection quality more than in-depth data collection.* DS and DD, both relying on easily accessible data and thus supporting property privacy, achieved the highest detection quality. By contrast, DH requires more data and higher permissions for data collection, not supporting property privacy. This result suggests that malware can be detected with high quality by favoring data preparation over in-depth data collection.

F3 *Malware detectors do not support real-world adversarial environments.* The lack of support of property robustness means that the considered detectors cannot safely operate in an adversarial environment. For what concerns DS, an attacker can purposefully modify a legacy portion of the executable file that does not affect the functionality of the malware to mislead DS. This scenario also applies to DH, since both system calls and strings are perturbed in a functionality-preserving manner. Instead, attacks against DD can either perturb *(i)* collected data points or *(ii)* the malware executable file. While attack *(i)* assumes full control of the system and is then inapplicable, attack *(ii)* is challenging, because the attacker should modify the malware executable file to affect system-wide performance metrics and escape classification. Recalling Sect. "Property Robustness", both these scenarios introduce input-dependent robustness (i.e., "by design"). The survey by Ling et al. [29] discusses the issue of real-world evasion attacks in malware detection.

F4 *Certification models give precise information on the conditions under which the properties have been evaluated.* Understanding the precise conditions under

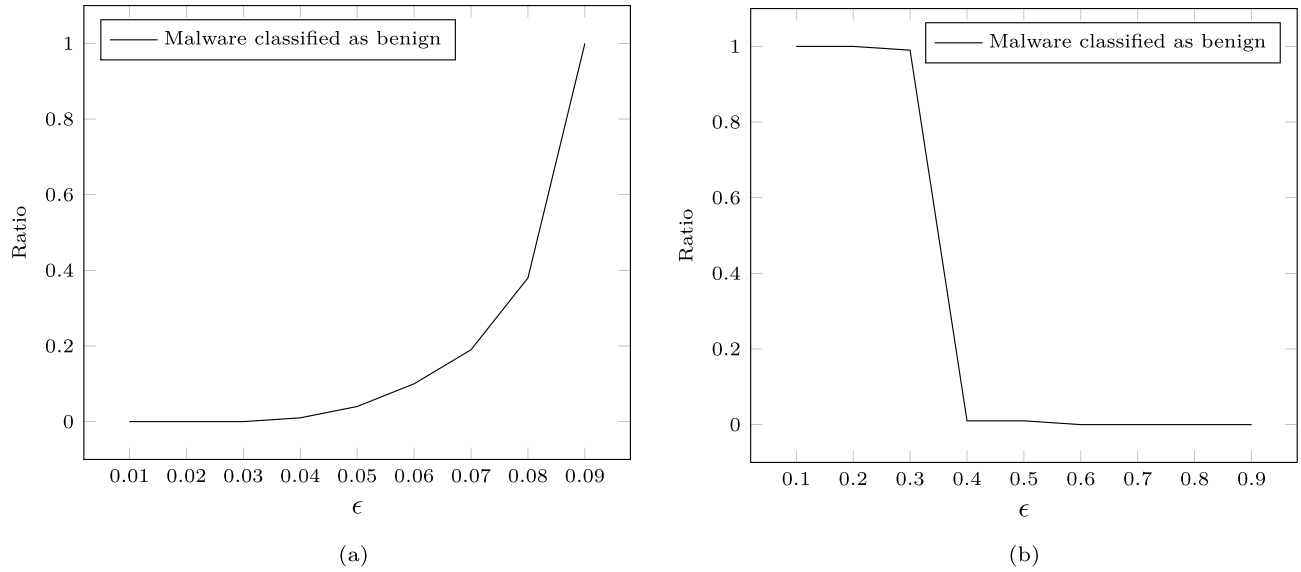

Fig. 7 Ratio of malware data points classified as benign in DD, out of 100 perturbed malware data points, with ϵ 0.9 (**a**) and $\epsilon \geq 0.9$ (**b**)

which the properties have been evaluated is fundamental for sound decision-making. According to the retrieved certification results, users may opt for a mathematically proven robust malware detector (e.g., [24, 44, 45]). Following F3, users can choose DD knowing that evasion attacks against it might be difficult in practice. Finally, users willing to give full access to their system might also choose DH.

Table 5 Certification results

(a) Property accuracy—dimension d_m		
Appl.	Collected evidence	Result
DS[8, 41]	AUC = 0.99821, ACC =-	✓
DD[11]	AUC = 0.9975, ACC = 0.9975	✓
DH[44]	AUC =-, ACC = 0.9694	✓
(b) Property privacy—dimension d_d		
Appl.	Collected evidence	Result
DS[8, 41]	Input space= { `executable-file` (1)}	✓
DD[11]	Input space= { `system-performance` (1)}	✓
DH[44]	Input space= { `executable-file` (1), `syscall` (2+requires admin)}	✗
(c) Property robustness—dimension d_p		
Appl.	Collected evidence	Result
DS[8, 41]	Adv. training not used	✗
DD[11]	Adv. training not used	✓
DH[44]	Adv. training not used	✗
(d) Property robustness—dimension d_m		
Appl.	Collected evidence	Result
DS[8, 41]	87% perturbed malware misclassified	✗
DD[11]	The ML model requires data whose perturbation necessitates complete control of the victim system	✗
DH[44]	82% perturbed malware misclassified	✗

- means that such data are not available; numbers between brackets in Table 5(b) indicate the privacy score according to Sect. "Property Privacy"

From the above findings, we conclude that in an adversarial environment, *using simpler ML models with carefully selected features can lead to better results* in malware detection performance and privacy. The latter can also facilitate the usage of robustness techniques due to the simplicity of both the training process and the model (e.g., low training time). We finally conclude that *certification is fundamental to reliably evaluate and distribute ML-based applications* following AI Act prescriptions.

Related Work

We extend the discussion in Sect. "Motivations", providing a complete overview of ML-based malware detectors classified according to the type of analysis (i.e., *static*, *dynamic*, and *hybrid*) and the considered features.

Static analysis. Static analysis approaches consider features extracted from executable files. Frequently, API/system calls and assembly instructions are considered. For example, Hardy et al. [22] focused on Windows API calls. Each data point represents an executable file, whose features are the one-hot encoded API calls found in the file. The accuracy recovered according to a stacked Autoencoder is ≈ 0.97. Kan et al. [26] considered the instructions found in the assembly code recovered from the executable file. Similar instructions are grouped to reduce the dimensionality of the input space. The accuracy retrieved according to a CNN (convolutional neural network) is ≈ 0.99 at most.

Control-flow graphs can also be extracted from the executable file. For example, Ma et al. [31] focused on Android malware. Three sets of features are extracted from the control-flow of each app. The first set is the invoked Android API calls, fed to a decision tree; the second set is the number of times each API is invoked, fed to a deep neural network (DNN); the third set is the ordered sequence of invoked APIs, fed to an LSTM model. The output of the three classifiers is combined by using soft voting, achieving F1-score ≈ 0.99. Herath et al. [23] fully exploited control-flow graphs. Nodes in the graph represent individual code blocks, edges the execution flow, and attributes data on the block operations. The graph is fed to a graph-native ML model (Deep Graph CNN), achieving recall 1 at most.

Non-traditional features have also been proposed. For instance, Kolter et al. [27] converted executable files into a hexadecimal representation and retrieved sequences of four-bytes n-grams. The top-500 most informative n-grams are selected for training and fed to different shallow learning models. The AUC recovered (area under curve) is ≈ 0.99 at most, according to the AdaBoost decision tree.

Based on the seminal work of Nataraj et al. [36], image representations have been used. Each data point represents the bytes of the executable files as pixels in a gray-scale image. Texture-based features are then extracted and classified using k-nearest neighbors (kNN). The retrieved accuracy in distinguishing between malware families is ≈ 0.99. Kalash et al. [25] and Ahmed et al. [3] fed this gray-scale representation to a CNN, achieving accuracy ≥ 0.97 in the aforementioned task. Yan et al. [50] used three sets of static features retrieved from executable files. The first set is a gray-scale image representation, fed to a CNN; the second are the assembly instructions sequences, fed to an LSTM model; the third are the characteristics of the executable file itself. The output of two classifiers and the third feature set is stacked on a logistic regression model, achieving accuracy ≈ 0.99. Darwaish et al. [16] represented static features as an RGB image, using a specific pre-processing that separates benign and suspicious features into different channels. Images are fed to CNN achieving accuracy ≈ 0.99. The proposed approach also exhibits high empirical robustness.

Dynamic analysis. Dynamic analysis approaches consider features extracted from the system and its processes. Rieck et al. [43] introduced q-grams. They are a compact representation of observed system calls and their parameters, retrieved over q-sized sliding windows. q-grams are then one-hot encoded, and their dimensionality is reduced to facilitate (incremental) comparison. The retrieved (modified) F1-score is ≈ 0.99 using a custom distance-based classifier. Zemmari et al. [39] focused on Android malware. Each data point is a vector of the most discriminant system calls of an app. Each system call is represented according to its frequency. The AUC retrieved according to shallow learning models such as random and rotation forests is 1 at most. Dai et al. [15] considered three features sets referred to each process. The first set contains the sequence of observed system calls, preprocessed using natural language processing techniques, the second set contains the values of hardware performance counters. These two features sets are fed to a gated recurrent unit (GRU) network. The third set contains the gray-scale image representation of the process memory dump. The output of the two classifiers is combined using soft voting, achieving accuracy ≈ 0.97. Abdelsalam et al. [2] focused on process-level information to detect an infected VM in the cloud. Each data point corresponds to a VM at a given time instant, and is represented as a two-dimensional matrix. Each row refers to a process, each column to process data such as percentage of CPU usage, number of context switches, number of opened file descriptors. The accuracy retrieved according to a CNN is ≈ 0.97 at most.

Finally, there are other features and representations. For example, Fang et al. [47] proposed a peculiar black-white image representation of the observed system calls. Each data point refers to the observed system calls of Android apps, transformed into images. A CNN achieved F1-score ≈ 0.98 at most. Busch et al. [13] focused on network traffic, represented using a graph. It encodes network flow data, from

endpoints to packet-level data. Graphs are fed into a Graph NN, achieving recall ≈ 0.99.

Hybrid analysis. Hybrid analysis approaches consider both static and dynamic analyses. For example, Lu et al. [30] focused on Android malware. Static features refer to app data such as file entropy, permissions, and intents, fed into a deep belief network. Dynamic features refer to the sequence of invoked Android API calls fed into a GRU network. The output of the two classifiers is stacked on a NN, achieving precision ≈ 0.97. Miller et al. [33] considered Windows malware. Static features refer to the content (e.g., imports, *packer*, etc.) and metadata (e.g., operating system version) of the executable file. Dynamic features refer to the *n*-grams of Windows API calls, paths of accessed files, requested IP addresses, to name but a few. The training dataset is labeled according to existing anti-malware tools and, upon dubious match, human experts. The detection rate retrieved according to logistic regression is 0.89.

Conclusions

Real-world malware detection is an urgent need that has been investigated by the research community over the last decades. The approach in this paper started from the requirements in the AI Act and defined a lightweight malware detector that supports non-functional properties beyond vanilla accuracy, including privacy and robustness. Our detector relies on a limited amount of data that can be easily collected with low permissions without affecting the ability to distinguish legitimate behavior from malware. We discussed the importance of advancing detector verification to the next step, and introduced an ML certification scheme supporting the verification of the detector behavior according to a large set of non-functional properties. Finally, we certified and compared the proposed approach with two malware detectors in the state of the art, showing that privacy and robustness can be supported with low impact on detector accuracy.

Funding Open access funding provided by Università degli Studi di Milano within the CRUI-CARE Agreement. The work was partially supported by project "BA-PHERD—Big Data Analytics Pipeline for the Identification of Heterogeneous Extracellular non-coding RNA as a Disease Biomarkers", funded by the European Union—NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.1: "Fondo Bando PRIN 2022" (CUP G53D23002910006); project MUSA—Multilayered Urban Sustainability Action-project, funded by the European Union—NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.5: Strengthening of research structures and creation of R&D "innovation ecosystems", set up of "territorial leaders in R&D" (CUP G43C22001370007, Code ECS00000037); project SERICS (PE00000014) under the NRRP MUR program funded by the EU-NextGenerationEU; program "piano sostegno alla ricerca" PSR and the PSR–GSA–Linea 6; project ReGAInS, funded by the Italian University and Research Ministry, within the Excellence Departments program (law 232/2016). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

Code Availability All artifacts are available at https://doi.org/10.13130/RD_UNIMI/5VTJCC.

References

1. Abdelsalam M, Krishnan R, Sandhu R. Clustering-based IaaS cloud monitoring. In: Proc. of IEEE CLOUD 2017, Honolulu. 2017.
2. Abdelsalam M, Krishnan R, Sandhu R. Online malware detection in cloud auto-scaling systems using shallow convolutional neural networks. In: Proc. of DBSec 2019, Charleston. 2019.
3. Ahmed I, Anisetti M, Ahmad A, et al. A multilayer deep learning approach for malware classification in 5g-enabled iiot. IEEE TII. 2023;19:2.
4. Alhashmi N, Almoosa N, Gianini G. Path asymmetry reconstruction via deep learning. In: Proc. of IEEE MELECON 2022, Palermo. 2022.
5. Almazrouei E, Gianini G, Mio C, et al. Using autoencoders for radio signal denoising. In: Proc. of ACM Q2SWinet 2019, Miami Beach. 2019.
6. Almazrouei E, Gianini G, Almoosa N, et al. What can machine learning do for radio spectrum management? In: Proc. of ACM Q2SWinet 2020, Alicante. 2020.
7. Almazrouei E, Gianini G, Almoosa N, et al. Robust computationally-efficient wireless emitter classification using autoencoders and convolutional neural networks. Sensors. 2021;21(7):2414.
8. Anderson HS, Roth P. EMBER: an open dataset for training static PE malware machine learning models (2018). arXiv:1804.04637.
9. Anisetti M, Ardagna CA, Bena N. Multi-dimensional certification of modern distributed systems. IEEE TSC. 2023;16(3):1999–2012.
10. Anisetti M, Ardagna CA, Bena N, et al. Rethinking certification for trustworthy machine-learning-based applications. IEEE Internet Comput. 2023;27(6).

11. Anisetti M, Ardagna CA, Bena N, et al. Lightweight behavior-based malware detection. In: Proc. of MEDES 2023, Heraklion. 2023.
12. Ardagna CA, Bena N. Non-functional certification of modern distributed systems: a research manifesto. In: Proc. of IEEE SSE 2023, Chicago. 2023.
13. Busch J, Kocheturov A, Tresp V, et al. NF-GNN: network flow graph neural networks for malware detection and classification. In: Proc. of ACM SSDBM 2021, Tampa. 2021.
14. Commission E. Artificial intelligence—questions and answers*. Tech. rep., European Commission. 2023. https://ec.europa.eu/commission/presscorner/api/files/document/print/en/qanda_21_1683/QANDA_21_1683_EN.pdf
15. Dai Y, Li H, Qian Y, et al. SMASH: a malware detection method based on multi-feature ensemble learning. IEEE Access. 2019;7:112588.
16. Darwaish A, Naït-Abdesselam F, Titouna C, et al. Robustness of Image-based android malware detection under adversarial attacks. In: Proc. of IEEE ICC 2021, Montreal. 2021.
17. Demetrio L, Biggio B, Lagorio G, et al. Explaining vulnerabilities of deep learning to adversarial malware binaries. In: Proc. of ITASEC 2019, Pisa. 2019.
18. European Union Agency for Cybersecurity. ENISA Threat Landscape 2022. Tech. rep. European Union Agency for Cybersecurity. 2022.
19. Gibert D, Zizzo G, Le Q. Certified robustness of static deep learning-based malware detectors against patch and append attacks. In: Proc. of ACM AISec 2023, Copenhagen. 2023.
20. Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. In: Proc. of ICLR 2015, San Diego. 2015.
21. Grosse K, Papernot N, Manoharan P, et al. Adversarial examples for malware detection. In: Proc. of ESORICS 2017, Oslo. 2017.
22. Hardy W, Chen L, Hou S, et al. DL4MD: a deep learning framework for intelligent malware detection. In: Proc. of DMIN 2016, Las Nevas. 2016.
23. Herath JD, Wakodikar PP, Yang P, et al. CFGExplainer: explaining graph neural network-based malware classification from control flow graphs. In: Proc. of 2022 IEEE/IFIP DSN, Baltimore. 2022.
24. Huang Z, Marchant NG, Lucas K, et al. Rs-del: edit distance robustness certificates for sequence classifiers via randomized deletion. In: Proc. of NeurIPS 2023, New Orleans. 2023.
25. Kalash M, Rochan M, Mohammed N, et al. Malware classification with deep convolutional neural networks. In: Proc. of IFIP NfTMS 2018, Paris. 2018.
26. Kan Z, Wang H, Xu G, et al. Towards light-weight deep learning based malware detection. In: Proc. of IEEE COMPSAC 2018, Tokyo. 2018.
27. Kolter JZ, Maloof MA. Learning to detect and classify malicious executables in the wild. JMLR 2006;7(12)
28. Lansing J, Benlian A, Sunyaev A. "Unblackboxing" Decision Makers' interpretations of IS certifications in the context of cloud service certifications. JAIS. 2018;19.
29. Ling X, Wu L, Zhang J, et al. Adversarial attacks against Windows PE malware detection: a survey of the state-of-the-art. In: COSE. 2023. p. 128.
30. Lu T, Du Y, Ouyang L, et al. Android malware detection based on a hybrid deep learning model. In: SCN 2020. 2020.
31. Ma Z, Ge H, Liu Y, et al. A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE Access. 2019;7:21235–45.
32. Malwarebytes. 2023 state of malware. Malwarebytes: Tech. rep. 2023.
33. Miller B, Kantchelian A, Tschantz MC, et al. Reviewer integration and performance measurement for malware detection. In: Proc. of DIMVA 2016, San Sebastiàn. 2016.
34. Milosevic J, Malek M, Ferrante A, et al. A friend or a foe? Detecting malware using memory and CPU features. In: Proc. of SECRYPT 2016, Lisbon. 2016.
35. Mio C, Gianini G. Signal reconstruction by means of embedding, clustering and AutoEncoder ensembles. In: Proc. of IEEE ISCC 2019, Barcelona. 2019.
36. Nataraj L, Karthikeyan S, Jacob G, et al. Malware images: visualization and automatic classification. In: Proc. of VizSec 2011, Pittsburgh. 2011.
37. Nicolae MI, Sinn M, Tran MN, et al. Adversarial robustness toolbox v1.2.0. 2018. arXiv:1807.01069.
38. NIST. Artificial intelligence risk management framework (ai rmf 1.0). Tech. rep., NIST. 2023.
39. Vinod P, Zemmari A, Conti M. A machine learning based approach to detect malicious android apps using discriminant system calls. In: FGCS 2019. p. 94.
40. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: machine learning in Python. In: JMLR 2011. p. 12.
41. Raff E, Barker J, Sylvester J, et al. Malware detection by eating a whole EXE (2017). arXiv:1710.09435.
42. Ramos IFF, Gianini G, Damiani E. Neuro-symbolic AI for sensor-based human performance prediction: system architectures and applications. In: Proc. of ESREL 2022, Dublin. 2022.
43. Rieck K, Trinius P, Willems C, et al. Automatic analysis of malware behavior using machine learning. JCS. 2011;19(4):639–68.
44. Rosenberg I, Shabtai A, Rokach L, et al. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In: Proc. of RAID 2018, Heraklion. 2018.
45. Saha S, Wang W, Kaya Y, et al. DRSM: de-randomized smoothing on malware classifier providing certified robustness. In: Proc. of ICLR 2024, Vienna. 2024.
46. Szegedy C, Zaremba W, Sutskever I, et al. Intriguing properties of neural networks. In: Proc. of ICLR 2014, Banff. 2014.
47. Wang F, Al Hamadi H, Damiani E. A visualized malware detection framework with CNN and conditional GAN. In: Proc. of IEEE Big Data 2022, Osaka. 2022.
48. Wang J, Chang X, Wang Y, et al. Lsgan-at: enhancing malware detector robustness against adversarial examples. Cybersecurity. 2021;4(1):1–15.
49. Yakimova Y, Ojamo J. Artificial intelligence act: Meps adopt landmark law. 2024. https://www.europarl.europa.eu/news/en/press-room/20240308IPR19015/artificial-intelligence-act-meps-adopt-landmark-law.
50. Yan J, Qi Y, Rao Q. Detecting malware with an ensemble method based on deep neural network. In: SCN 2018. 2018.
51. Yoon J, Jarrett D, van der Schaar M. Time-series generative adversarial networks. In: Proc. of NeurIPS 2019, Vancouver. 2019.