Deceiving AI-based Malware Detection through Polymorphic Attacks

Catalano C.[a,a], Chezzi A.[b,a], Angelelli M.[c,a] and Tommasi F.[d,a]

[a] *Department of Innovation Engineering, University of Salento, Lecce, Italy*

ARTICLE INFO

Keywords:
Malware Detection
Convolutional Neural Network CNN
Polymorphic malware
Adversarial Machine Learning
Adversarial Training

ABSTRACT

Malware detection is one of the most important tasks in cybersecurity. Recently, increasing interest in Convolutional Neural Networks (CNN) and Machine Learning algorithms, which are widely used in image analysis and predictive modelling, led to their use in static malware classification and to the application of these powerful tools in computer industry and industrial internet of things.

Many studies claim that the static malware detection approach, under well-defined conditions, can deliver fast and accurate malware classification results with relatively little human effort once the framework is implemented, relying solely on the binary content of the file. This becomes evident if we compare static malware detection to other techniques of dynamic nature. The focus of our research is to highlight strengths and weaknesses of CNNs used for static malware detection, starting from images obtained from byte-wise conversion of binary executable files to pixel images to critically analyze the assumptions underlying the performance of this type of technique.

1. Introduction

Malware may have dramatic effects on both personal and public digital assets, so the detection of malware is an area of major concern both for the research community and for private and public entities.

For this reason threats and vulnerabilities in the Industrial Internet of Things (IIoT) context deserve particular consideration where cybersecurity for Cyber-Physical Systems (CPS) and for networked Industrial Control Systems (nICS) become increasingly jeopardized by malware attacks. The malware menace in such critical industrial environments raises not only a demand of business impact assessment [1, 2] due to cyber-attacks but also specific need of cybersecurity models, frameworks, architectures [3] which are able to detect malware threats on the IIoT and promptly neutralize them.

1.1. Preliminary notions

Due to the relevance of malware detection techniques as a counter-response, it is necessary to understand their strengths and limitations, in order to avoid their improper adoption without appropriate knowledge of their modes of action. This work attention focuses on the analysis of the limits and opportunities offered by neural networks algorithms in the improvement of malware detection systems and provides new useful perspectives to find better practices in order to build models and tools which comply to the modern IIoT cybersecurity policies and needs.

For the sake of clarity, let us recall here the definitions of the most important concepts addressed in this work.

Malware Malware, also known as malicious code and malicious software, refers to a program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system, or otherwise annoying or disrupting the victim. Malware has become the most significant external threat to most systems, causing widespread damage [4].

Malware detection A malware detector is a program designed to detect malicious programs or code [5]. In general, a malware detector can be modeled as a function defined as follows [6]:

$$D(p) = \begin{cases} \texttt{malicious} & \text{if } p \text{ contains malicious code;} \\ \texttt{benign} & \text{if } p \text{ does not contain malicious code;} \\ \texttt{undecidable} & \text{if } D \text{ fails to determine } p; \end{cases} \tag{1}$$

where D is the decision function that can check whether an application or program p is either a benign or a malicious one.

From the above notation it can be seen that there are cases for which D cannot decide whether a program is harmful or not, perhaps because the malware has been created too recently or, more generally, because it does not provide enough elements to output an accurate detection result.

Malware analysis is the process of understanding the behavior and purpose of a suspicious file or piece of code, and it is one of the first steps towards malware detection. There are two fundamental approaches to malware analysis, namely, Static and Dynamic [7, 8, 9]:

- Static (or Code) analysis is a process to analyze a malware binary without actually running the code. It consists of reverse-engineering the malware's internals by loading the executable into a disassembler and looking at the program instructions, in order to discover what the program does [10, 11, 12, 13].
- Dynamic (or Behavioural) analysis is performed through the observation of the behavior of the malware while it is actually running on a host system. This form of analysis is often performed in a sandbox environment

✉ christian.catalano@unisalento.it (C. C.); andrea.chezzi@studenti.unisalento.it (C. A.); mario.angelelli@unisalento.it (A. M.)

ORCID(s): 0000-0003-4038-2317 (C. C.); 0000-0002-9782-7834 (A. M.)

[10]. The malware may also be debugged while running, using a dedicated program referred to as a debugger. This approach lets one observe the behaviour and effects of the malware on the host system, step by step, while its instructions are being processed.

These two paradigms are not mutually exclusive, and malware analysis techniques may be conducted also through a hybrid approach.

To improve the existing malware detection techniques and to increase the efficiency in the detection process, machine learning and data mining methods may be combined with traditional detection methods. In this context, we will evaluate some of the recent advances in the use of CNN (Convolutional Neural Networks) algorithms for malware detection.

To this purpose, we have tested (section 5) the robustness of a malware detector based on a CNN (section 3) against polymorphic attacks (section 4). The research results support the hypothesis that the efficiency of malware detectors using neural networks follows from special assumptions on the way the malware is designed. It will be seen that when these assumptions are violated, as it happens for polymorphic attacks, CNNs may be deceived, leading to misclassification and drastic effects on performance.

The focus of the present paper is set on a particular type of malware: that presenting itself as (or inside) Windows PE or Linux ELF binary files. Several other classes of malware recently gaining increasing attention are excluded from the present analysis (e.g those based on Javascript code, see for example BitM [14] and MinerAlert [15]).

2. State of the art

In the scope of the present research attention will be devoted to the topics listed below:

- Static malware analysis techniques (defender side);
- Image-based analysis and CNN algorithms for malware detection and classification (defender side);
- Adversarial Machine Learning (attacker side);
- Polymorphic malware (attacker side);

2.1. Static malware analysis techniques

As previously mentioned, static analysis examines the file for signs of malicious intent, disassembling the software and deciding whether it contains malicious code. There are different tools that can be used to perform static analysis, such as debuggers, disassemblers, decompilers and source code analyzers. Methods that are used in static analysis include File Format Inspection, String Extraction, Fingerprinting, AV scanning, and Disassembly [10].

2.2. Image-based analysis and CNN algorithms for malware detection and classification

Different studies in static malware detection claim optimistic results regarding the use of CNN, where the classification is based on the transformation of malware binary samples into pixels constituting the images to be analyzed. The novel aspect of this approach resides in the shift of perspective in the analysis, moving from the binary code to graphical features extracted from it: in this way, it is expected that the CNN, once trained on a dataset of images derived from malware binary samples, can decide how to classify a piece of software (using the function D described in (1)) based on graphical features, rather than on the individual blocks of code.

Hereafter, we briefly report a survey of the literature that backs the potentialities of CNN technology for malware detection. Among the first research studies in this context, we mention [16] and [17], whose main idea is to transform the byte values (ranging in $[0, 255]$) of the binary code into grayscale images, with shades of gray from 0 to 255. Once the corresponding images have been obtained, they are used by the graphic pattern recognition algorithms. CNN in 2011 were not yet as popular as today, so in the first paper graphic pattern recognition algorithms SIFT/GIST were used for the classification tests, without relying on machine learning techniques. On the other hand, the second paper envisages TensorFlow to implement unsupervised machine learning algorithms such as kNN and SVM.

Since then, several studies have adopted CNN as a support for classification of malware converted to images [18, 19, 20, 21, 22, 23, 24, 25]. The proposed techniques vary in the type of algorithms supported, in the setting of the hyper-parameters, in the type of malware detected, and in the conversion techniques (binary-to-image) used.

All these works have contributed to define the base for our analysis, providing a class of well-defined approaches that we can test.

2.3. Adversarial Machine Learning

A typical example of an adversarial attack is the exposition of inaccurate or misrepresentative data to a machine-learning model with during its training, or the introduction of maliciously designed data to deceive an already trained model into making errors.

The term *adversary* is used in the field of computer security to describe agents (humans or machines) that attempt to penetrate or corrupt a computer network or a program. Adversaries can use a variety of attack methods to disrupt a machine learning model, either during the training phase (called a *poisoning attack*), or after the classifier has already been trained (an *"evasion" attack*). There are a large variety of different adversarial attacks that can be used against machine learning systems. Many of these work on both deep learning systems as well as traditional machine learning models such as SVMs and linear regression [39, 40, 41].

As a high-level instance of these attack, we discuss an adversarial example, namely, a specially crafted input that is designed to look *normal* to humans, but leads a machine learning model to misclassification. Often, an appropriate form of *noise* is used to elicit the misclassifications. Adversarial examples, which we can also call *antagonistic examples*, are a real problem for any machine learning technique:

through changes that are imperceptible to the human eye, it is possible to generate examples that can confuse any classifier with very high probability, regardless of its accuracy during training.

One of the most interesting methods developed to mislead a CNN consists in finding and applying *Universal Adversarial Perturbations* to image samples that need to be classified [26, 27, 28].

This unintended behaviour of neural networks under adversarial examples can be exploited by attackers to create malicious software specifically designed to evade cybersecurity defenses based on neural network classifiers, not to mention other critical consequences in the most diverse areas of use of CNN in real life, where the classification and recognition of images is a crucial point for the safety and security of the users [29, 30, 31, 32].

2.4. Polymorphic malware

Polymorphic malware [33, 34, 35] is a type of malware that constantly changes its identifiable features in order to evade detection [36]. Polymorphic malware uses encryption techniques, but in a more complex way than other malware does: with encryption, the main body of the code (also referred to as its payload) appears meaningless, and its functionalities are restored through decryption. For the malicious code, a decryption function is added to the code, so that, when the code is executed, this function reads the payload and decrypts it before its execution. Encryption alone is not polymorphism: in order to gain polymorphic behaviour, the encrypter/decrypter pair is mutated with each copy of the code. This allows different versions of the code, all of which work the same way, with no *a priori* bound on the number of encrypted versions they can generate. For this reason, antiviruses can hardly detect polymorphic malware [37].

3. CNN malware detector

The implementation of our malware detector is based on design choices that are in line with most of the specifications found in the previously cited works.

3.1. Architecture overview

The hereby illustrated malware detector is basically a Convolutional Neural Network algorithm that solves a 2-class binary image classification problem, where the two classes are identified with the labels “malware” and “goodware”. Here, the term “goodware” refers to any harmless program or executable that works without any intentional (e.g. non-random) malicious effect.

The configuration is meant to prevent the occurrence of false negatives during the prediction phase, adopting a preventive and more cautious perspective.

In Figure1, the architecture of the malware detector is illustrated.

The first phase is preprocessing, which converts binary executables into pixel images.

Once the image samples are obtained, they are ready to be fed as input into the trained CNN architecture, which will return the classification scores used to determine the class of the examined sample. In the figure 1 below, an example shows what happens when a malware executable is correctly detected.

3.2. Main properties

The proposed malware detector has the following key properties:

- It uses static analysis: the executable files are examined without running them, the useful information that drives the detector's decision is extracted directly from the binary file content.
- Signature detection: the detector bases its evaluation on the experience gained in the machine learning training phase, where it learnt specific features and signatures of the existing malware from a given dataset.
- Similarities with anomaly detection: given a dataset, this detector learns not only the signatures of the malware samples, but also the features of the harmless goodware, and makes its prediction accordingly. This behaviour is analogous to several anomaly detection methods.
- Uses images and not flat byte strings as input.

To avoid any misunderstanding, it is worth noting how the present work fully acknowledges the novelty and the usefulness of CNNs in the context of malware detection. Far from denying it, the focus of the present work is the assessment of the existence of a wide class of malware that, through some of the countless way to obfuscate its code might be able to escape the CNNs' detection mechanisms. In short, while CNNs are a certainly a strong aid in the struggle against malware, one cannot rely exclusively on them in such endeavour.

3.3. Implementation

For this work, we have chosen three among the most popular CNN architectures, namely: VGG16, ResNet50v2, and InceptionV3, and they have been implemented using Keras and Tensorflow frameworks for Python 3.8.6 (64 bit). The machine dedicated to training features a GTX 1080 GPU (8 GB VRAM) and an Intel Core i7 2600 CPU with 32 GB of RAM.

3.4. Dataset and training

The Linux (ELF files) and Windows (Win PE format) malware executable samples have been downloaded and selected from VirusShare.com, which holds a repository of malware samples [38]. On the other hand, the goodware samples collection for both platforms have been gathered by dumping applications and regular program executables from installed OS environments. At this point, our dataset samples are binary files:

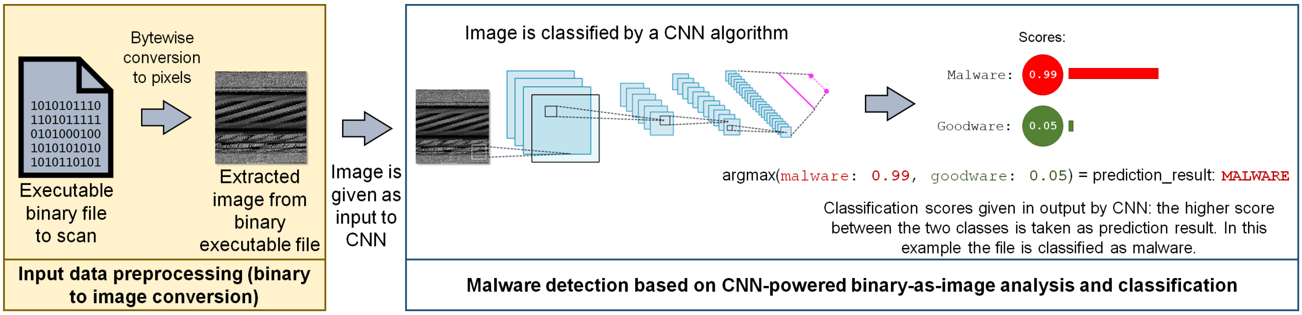
Bytewise conversion to pixels
1010101110
1101011111
0101000100
1010101010
1010110101
Executable binary file to scan
Extracted image from binary executable file
Input data preprocessing (binary to image conversion)
Image is given as input to CNN
Image is classified by a CNN algorithm
Scores:
Malware:
0.99
Goodware:
0.05
argmax(malware: 0.99, goodware: 0.05) = prediction_result: MALWARE
Classification scores given in output by CNN: the higher score between the two classes is taken as prediction result. In this example the file is classified as malware.
Malware detection based on CNN-powered binary-as-image analysis and classification

Figure 1: CNN malware detector architecture overview

- 8674 samples for Linux (4544 malware and 4130 goodware)
- 15439 samples for Windows (7493 malware and 7946 goodware)

In order to generate a dataset that can be used by a CNN learning algorithm, we need to convert the binary executable samples into images, more precisely grayscale pixel images: for this purpose, each byte from the binary file, which can assume only unsigned values between 0 and 255, is mapped to a pixel of a specific tone of gray, labelled from value 0 (black) to value 255 (white).

For the training task, we need to split the dataset into training data and testing data. For both Windows and Linux, datasets are split into three subsets: the training set (70%), the validation set (15%), and the test set (15%).

The splitting for the two datasets is shown in Figure 2.

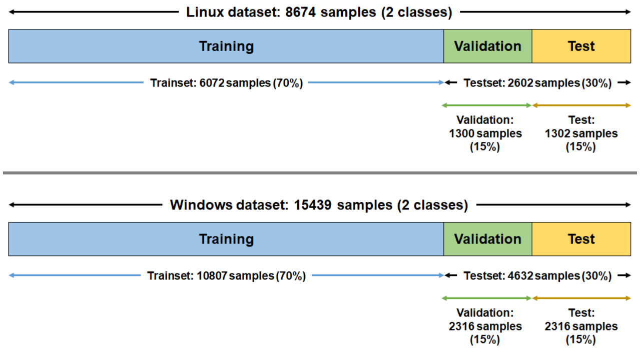
Linux dataset: 8674 samples (2 classes)
Training
Validation
Test
Trainset: 6072 samples (70%)
Testset: 2602 samples (30%)
Validation: 1300 samples (15%)
Test: 1302 samples (15%)
Windows dataset: 15439 samples (2 classes)
Training
Validation
Test
Trainset: 10807 samples (70%)
Testset: 4632 samples (30%)
Validation: 2316 samples (15%)
Test: 2316 samples (15%)

Figure 2: Dataset splitting

3.5. Preliminary test results on regular malware

The results assessing the prediction performance of CNN models trained on the samples in the test sets are shown in Tables 1 and 2.

Regarding Linux platform dataset, the results show very good malware classification and detection performance for all the three presented CNN architectures.

Each of the tested CNN reaches an accuracy above 99% on the test set samples, indicating an outstanding capability of generalization of the prediction on malware detection. In this case, the best CNN is found to be VGG16, with its accuracy of 99.77%, closely followed by the other two.

On Windows platform dataset, the results show lower, but still appreciable malware classification and detection performances for all the three tested CNN architectures. Each of the CNN reaches an accuracy above 86% on the test set samples. In this case, the best CNN is ResNet50v2, with its accuracy of 89.25%. The other two, VGG16 and Inception V3, offer similar performance in terms of accuracy, which is slightly less than 87%.

In both platform cases, the CNNs generate interesting results, confirming the claims of the literature papers [18, 19, 20, 21, 22, 23, 24, 25], presented earlier in the previous section, that malware detection and classification based on CNN-powered static image analysis represents a promising and reliable method in this field of cybersecurity.

4. Polymorphic attack

Although the results obtained so far in terms of CNN classification performance are remarkable, we have to check them on a new attack method that exploits CNNs' sensitivity to small perturbations in the datasets. The method we propose may break malware detectors as they have been implemented following the scientific literature, bringing to light the limits of CNN technology in terms of effects on the prediction results of the three CNN already trained and tested in the previous section. We can define this new method as an adversarial machine learning attack based on the design of malware in the form of polymorphic binary executables.

4.1. Proposed idea

The idea behind the attack is based on the difference between the extraction of meaningful graphic features from CNN architectures and the functionalities of the original malware's binary: this difference is exploited looking at alterations of graphic features derived from the binary-to-image conversion while preserving the functionalities of a malware. This type of alterations is enabled by a polymorphic engine and we will focus on its effects compared to the traditional signatures used by the standard antivirus and malware detection systems.

	CNN	Classes	# testset samples		Precision	Recall	F1-score	Accuracy
Linux dataset	VGG16	*Malware*	682	1302	99.71%	99.85%	99.78%	99.77%
		Goodware	629		99.84%	99.68%	99.76%	
	ResNet50v2	*Malware*	682	1302	99.85%	99.41%	99.63%	99.62%
		Goodware	620		99.36%	99.84%	99.60%	
	InceptionV3	*Malware*	682	1302	99.85%	99.97%	99.41%	99.39%
		Goodware	620		98.88%	99.84%	99.36%	

Table 1
Results of trained CNN models (Linux)

	CNN	Classes	# testset samples		Precision	Recall	F1-score	Accuracy
Windows dataset	VGG16	*Malware*	1124	2316	85.64%	87.54%	86.58%	86.83%
		Goodware	1192		99.84%	99.68%	99.76%	
	ResNet50v2	*Malware*	1124	2316	92.35%	84.88%	88.46%	**89.25%**
		Goodware	1192		86.75%	93.37%	89.94%	
	InceptionV3	*Malware*	1124	2316	93.32%	78.29%	85.15%	86.74%
		Goodware	1192		82.23%	94.71%	88.03%	

Table 2
Results of trained CNN models (Windows)

Our approach is to trigger a modification of scores for both the two classes (malware and goodware) of the classifiers, i.e. to lower the classification score for "malware" label and to raise the "goodware" score as well. More precisely, our objective is to invert the classification behavior of CNN-based malware detectors, so that a polymorphic malware makes the CNN recognizes it as a goodware, although it is definitely not, hence tearing apart their effectiveness at this task. This is shown in Figure 3 This line of attack is possible by leveraging polymorphism: we manipulate or "spike" the original malware samples by inserting typical goodware graphic features into them, still retaining unchanged the original code maliciousness. This step is expected to increase the goodware class score. This practice is also accompanied by layers of classic binary encryption that have the purpose of preventing the detection of graphic malware signatures and, as a consequence, lowering the "malware" class score of the examined samples. When tested, CNNs analyze this "spiked" malware, so they detect goodware graphic features misclassify it as harmless goodware samples.

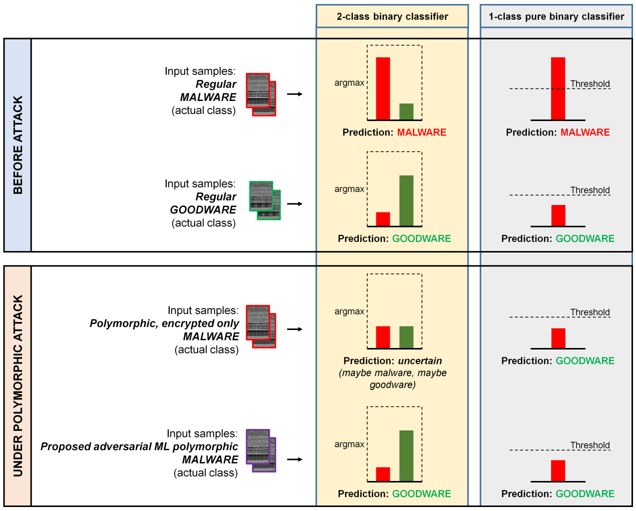
2-class binary classifier
1-class pure binary classifier
BEFORE ATTACK
Input samples: Regular MALWARE (actual class)
argmax
Prediction: MALWARE
Threshold
Prediction: MALWARE
Input samples: Regular GOODWARE (actual class)
argmax
Prediction: GOODWARE
Threshold
Prediction: GOODWARE
UNDER POLYMORPHIC ATTACK
Input samples: Polymorphic, encrypted only MALWARE (actual class)
argmax
Prediction: uncertain (maybe malware, maybe goodware)
Threshold
Prediction: GOODWARE
Input samples: Proposed adversarial ML polymorphic MALWARE (actual class)
argmax
Prediction: GOODWARE
Threshold
Prediction: GOODWARE

Figure 3: Different working scenarios of binary classifiers.

4.2. The bricks

The proposed polymorphic malware is composed of three main blocks (Figure 4): an extractor (decrypter), a goodware camouflage mask, and the encrypted malware binary code.

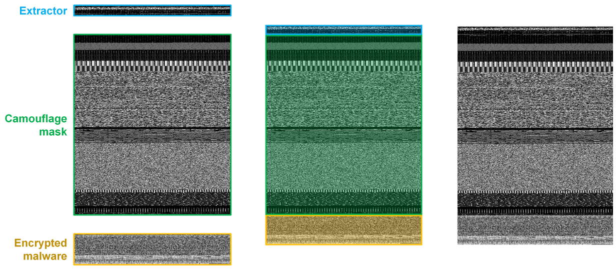

Figure 4: Polymorphic malware architecture

Extractor The extractor code is implemented in C, which allows to get a very lightweight and efficient binary once compiled. Its job is to reverse the operation of malware encryption done by the polymorphic engine: once the polymorphic malware has misled the CNN malware detector, the extractor decrypts the malware binary and runs it in the targeted system.

Camouflage mask The goodware samples that serve as camouflage masks are picked from the goodware samples of the respective Linux and Windows test sets.

Encrypted malware The malware samples are taken by the malware executables in both the Windows and Linux training sets. The choice of the samples from the training set gives the worst case scenario for the attacker side and the best one for the CNN detector: in fact, it is assumed that malware samples that belong to the training set on which the models have been trained are easier to be correctly detected by the classifiers themselves.

Implementation details In brief, the C extractor considers, at compile time, a set of descriptive parameters of the mask and the encrypted malware blocks, e.g. sizes in bytes, delimitation marker strings, padding sizes, and optional switches. The parameters in the C source code file need to be tuned appropriately: this task is automated by the polymorphic engine script written in bash. Once launched, the extractor performs the following sequence of tasks: it reads itself byte-by-byte, until it finds the first of two predetermined marker strings that indicates the offset where the goodware block begins. Then, it reads the bytes constituting the goodware block, stores them into an array, and stops when the number of bytes read equals the size of the goodware block.

Subsequently, the second marker is searched and an analogous procedure is done in order to read and store the sequence of bytes of the encrypted malware block into a second array. Having both blocks been stored, the decryption is carried out through bytewise subtraction mod 256 of the two arrays.

Once the malware has been decrypted, it is written on a file and executed on the fly on the target system. In order to retrieve the right offset of the blocks inside the polymorphic sample, the use of strings as markers becomes necessary due to the difficulty to determine the blocks exact size a priori. Optional steps may be performed during extraction, such as XOR decryption or other logical techniques.

4.3. The polymorphic nature of this approach

The polymorphic nature of this kind of approach is given by the possibility to pick and use any of the wide range of available goodware for a given target OS platform. This allows keeping the encrypted malware, under polymorphic disguise, reusable for several attacks. Indeed, even if a polymorphic malware is detected and added to database or the training set of the known malware samples, one can choose a new available goodware program, and use it as a camouflage mask: in this way, a new polymorphic malware is produced with the same functionality but with unknown signatures This characteristic is enhanced by the XOR encryption layer, which can increase the differences between different generations of polymorphic malware, enhancing polymorphism.

5. CNN under polymorphic attack

Polymorphic malware samples are created by stacking together the three aforementioned blocks. The process of stacking and binding correctly each of the blocks is carried out by our polymorphic engine, which is a shell script written for this purpose. Figure 5 shows the process of creating malware samples through the polymorphic malware engine

Now we evaluate the effects and test the robustness of the three CNN trained models against polymorphic malware. Figure 6 shows how the test of the attack is carried out in term of choice of samples from the dataset. The same procedure is used for both Linux and Windows datasets.

The previously trained CNN models are now attacked and the results are considered to assess the impact of the polymorphic malware.

6. Attack results

Regarding Linux platform dataset, the results show dramatic consequences caused by the polymorphic samples on classification detection performance for all the three CNN architectures under examination. None of the tested CNN has been able to correctly classify one single polymorphic malware sample: all samples have been misclassified, resulting in an evasion rate of 100% for all the models (Tables 3 and 4).

On Windows platform dataset, the results show basically the same situation as for Linux samples: also in this case, almost all of the polymorphic samples have been misclassified. In the best case represented by ResNet50v2 model, the lowest evasion rate exceeds 93% (Tables 3 and 4).

We note that the Linux dataset where the CNN performed better under the original malware's attack coincides with the dataset where the worst results are attained under the polymorphic attack. This behavior suggests that the best results of CNN models in normal conditions, i.e. with regular samples, may be less robust under *ad hoc* polymorphic malware attacks: a possible interpretation of this phenomenon relies on a stronger dependence on the assumption of "faithful" representation of malware's functionalities, and better results on the regular samples may be a hint of a stronger alignment of CNN models with the standard encoding of malware. This could lead to an unbalanced (worst) performance on malware encoded in non-standard forms, such as those obtained through the polymorphic engine.

7. Countermeasures

We now address the possibility to use adversarial training as a viable method to mitigate the consequences of polymorphic malware attacks involving CNN classifiers.

7.1. Adversarial training

The adversarial training is one of the most effective approaches to defend deep learning models against adversarial examples, since it aims to improve the inherent robustness of the models.

The initial idea of adversarial training is first brought to light by [Szegedy et al., 2014], where neural networks are trained on a mixture of adversarial examples and clean examples. Goodfellow et al. (2015) went further and proposed FGSM to produce adversarial examples during training.

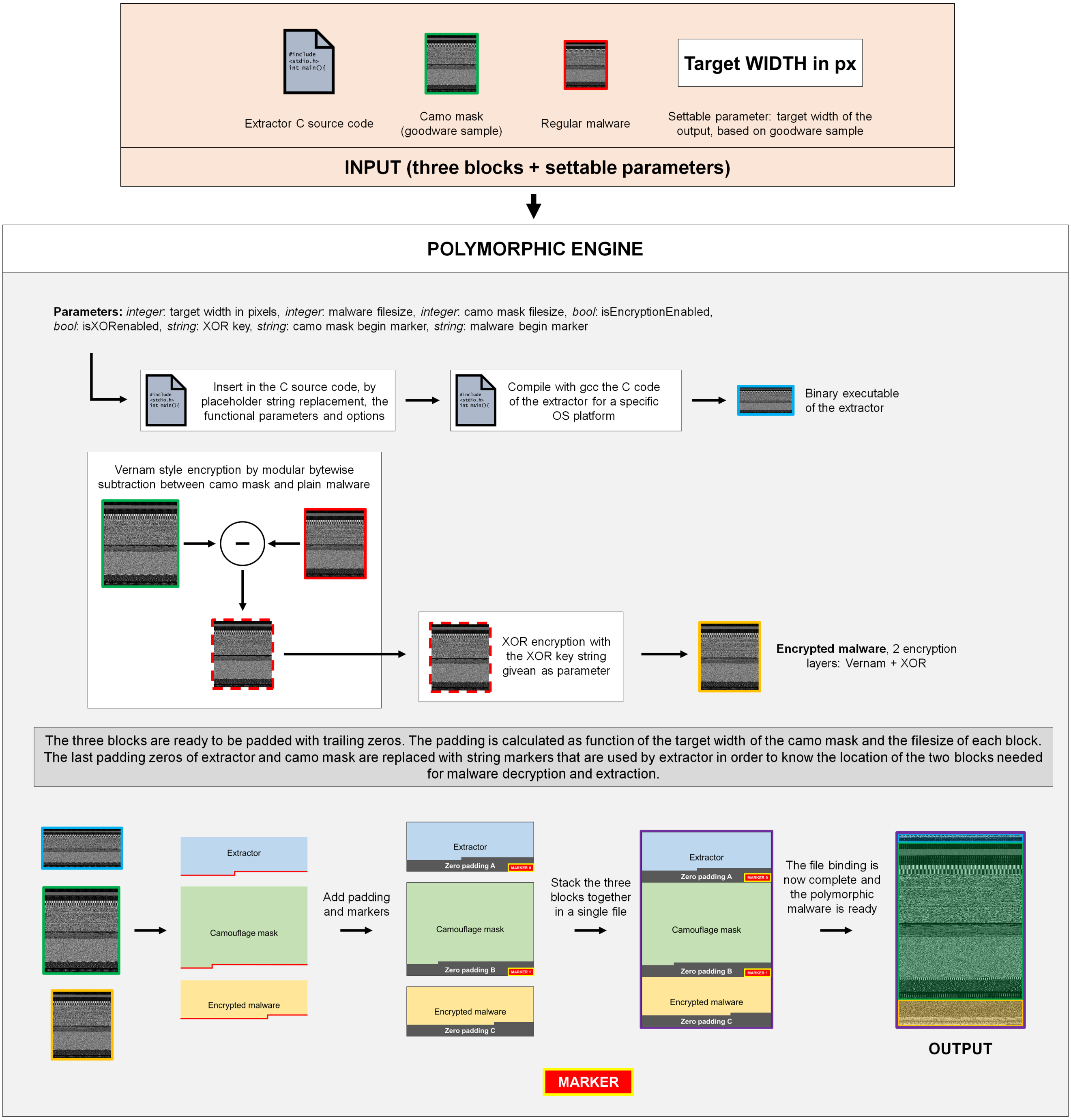
Extractor C source code
Camo mask (goodware sample)
Regular malware
Target WIDTH in px
Settable parameter: target width of the output, based on goodware sample
INPUT (three blocks + settable parameters)
POLYMORPHIC ENGINE
Parameters: integer: target width in pixels, integer: malware filesize, integer: camo mask filesize, bool: isEncryptionEnabled, bool: isXORenabled, string: XOR key, string: camo mask begin marker, string: malware begin marker
Insert in the C source code, by placeholder string replacement, the functional parameters and options
Compile with gcc the C code of the extractor for a specific OS platform
Binary executable of the extractor
Vernam style encryption by modular bytewise subtraction between camo mask and plain malware
−
XOR encryption with the XOR key string givean as parameter
Encrypted malware, 2 encryption layers: Vernam + XOR
The three blocks are ready to be padded with trailing zeros. The padding is calculated as function of the target width of the camo mask and the filesize of each block. The last padding zeros of extractor and camo mask are replaced with string markers that are used by extractor in order to know the location of the two blocks needed for malware decryption and extraction.
Extractor
Camouflage mask
Encrypted malware
Add padding and markers
Extractor
Zero padding A
MARKER 0
Camouflage mask
Zero padding B
MARKER 1
Encrypted malware
Zero padding C
Stack the three blocks together in a single file
Extractor
Zero padding A
MARKER 0
Camouflage mask
Zero padding B
MARKER 1
Encrypted malware
Zero padding C
The file binding is now complete and the polymorphic malware is ready
OUTPUT
MARKER

Figure 5: Polymorphic malware creation process

7.2. CNN hardening attempt by using adversarial training technique

A straightforward strategy for training an adversarially robust model is to create and, subsequently incorporate adversarial examples into the training process. In other words, knowing that "standard" training may generate models that are prone to adversarial examples, we also train on a limited number of adversarial examples.

A robust classifier is one that correctly labels adversarially perturbed images, whichare derived from polymorphic malware in our case.

7.3. Second attack

The adversarial samples of the first generation of polymorphic malware, which have produced the results discussed above, are inserted in both Linux and Windows train sets. Then, the same datasets are augmented with the first generation polymorphic malware samples, and the CNN models are trained on these extended datasets. After

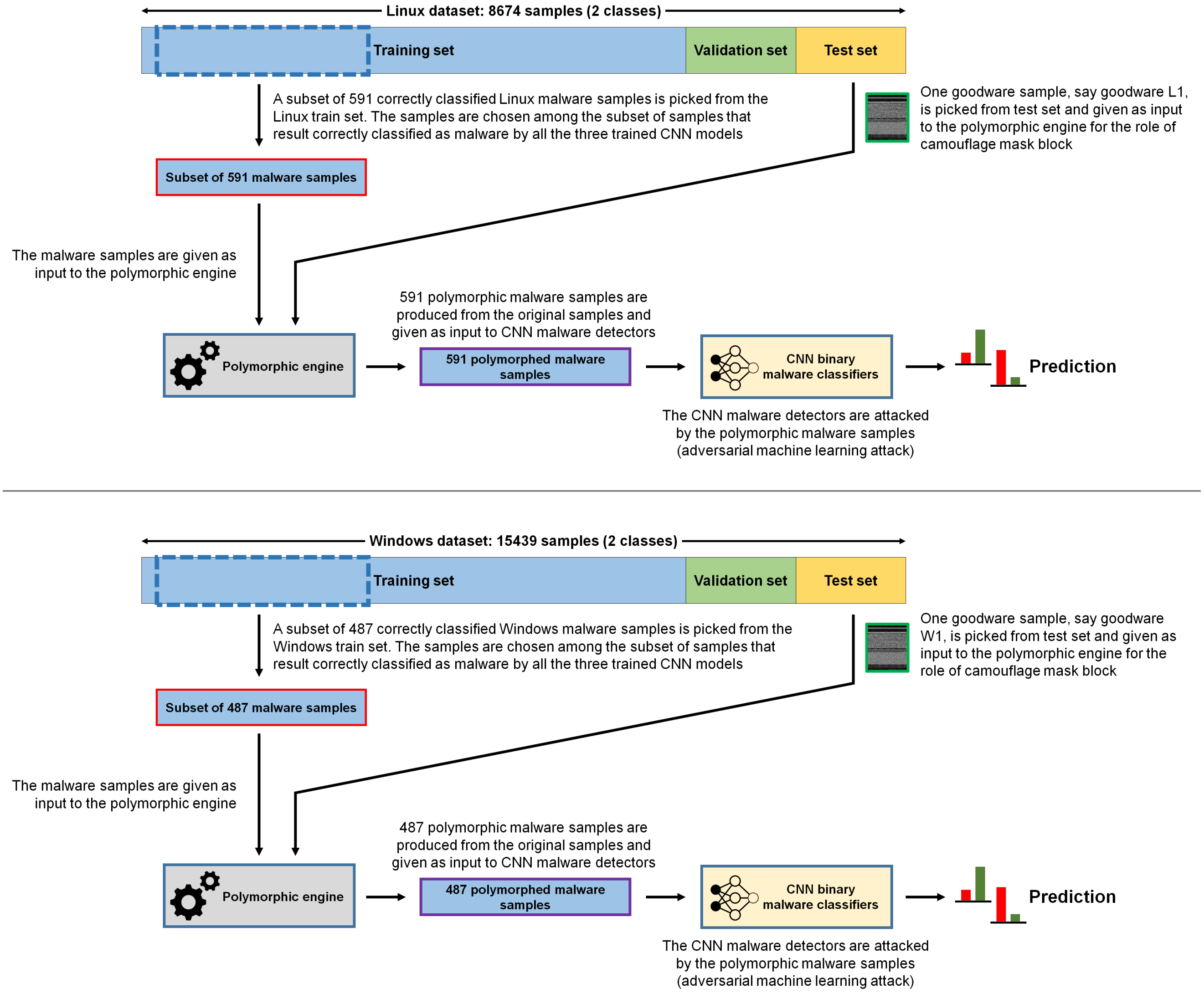
Linux dataset: 8674 samples (2 classes)
Training set
Validation set
Test set
A subset of 591 correctly classified Linux malware samples is picked from the Linux train set. The samples are chosen among the subset of samples that result correctly classified as malware by all the three trained CNN models
One goodware sample, say goodware L1, is picked from test set and given as input to the polymorphic engine for the role of camouflage mask block
Subset of 591 malware samples
The malware samples are given as input to the polymorphic engine
591 polymorphic malware samples are produced from the original samples and given as input to CNN malware detectors
Polymorphic engine
591 polymorphed malware samples
CNN binary malware classifiers
Prediction
The CNN malware detectors are attacked by the polymorphic malware samples (adversarial machine learning attack)
Windows dataset: 15439 samples (2 classes)
Training set
Validation set
Test set
A subset of 487 correctly classified Windows malware samples is picked from the Windows train set. The samples are chosen among the subset of samples that result correctly classified as malware by all the three trained CNN models
One goodware sample, say goodware W1, is picked from test set and given as input to the polymorphic engine for the role of camouflage mask block
Subset of 487 malware samples
The malware samples are given as input to the polymorphic engine
487 polymorphic malware samples are produced from the original samples and given as input to CNN malware detectors
Polymorphic engine
487 polymorphed malware samples
CNN binary malware classifiers
Prediction
The CNN malware detectors are attacked by the polymorphic malware samples (adversarial machine learning attack)

Figure 6: CNN attack

Table 3

Attack results for polymorphic malware samples designed for Linux

CNN	Classes	# testset samples		Mean Class. Margin	Well Classified	Misclassified	Evasion rate
VGG16	*Malware*	591	591	-0.999995	0	591	100%
	Goodware	0		NA	0	0	NA
ResNet50v2	*Malware*	591	591	-0.999816	0	591	100%
	Goodware	0		NA	0	0	NA
InceptionV3	*Malware*	591	591	-0.996308	0	591	100%
	Goodware	0		NA	0	0	NA

Table 4

Attack results for polymorphic malware samples designed for Windows

CNN	Classes	# testset samples		Mean Class. Margin	Well Classified	Misclassified	Evasion rate
VGG16	*Malware*	487	487	-0.717704	2	485	99.59%
	Goodware	0		NA	0	0	NA
ResNet50v2	*Malware*	487	487	-0.827743	32	455	93.42%
	Goodware	0		NA	0	0	NA
InceptionV3	*Malware*	487	487	-0.712381	12	475	97.53%
	Goodware	0		NA	0	0	NA

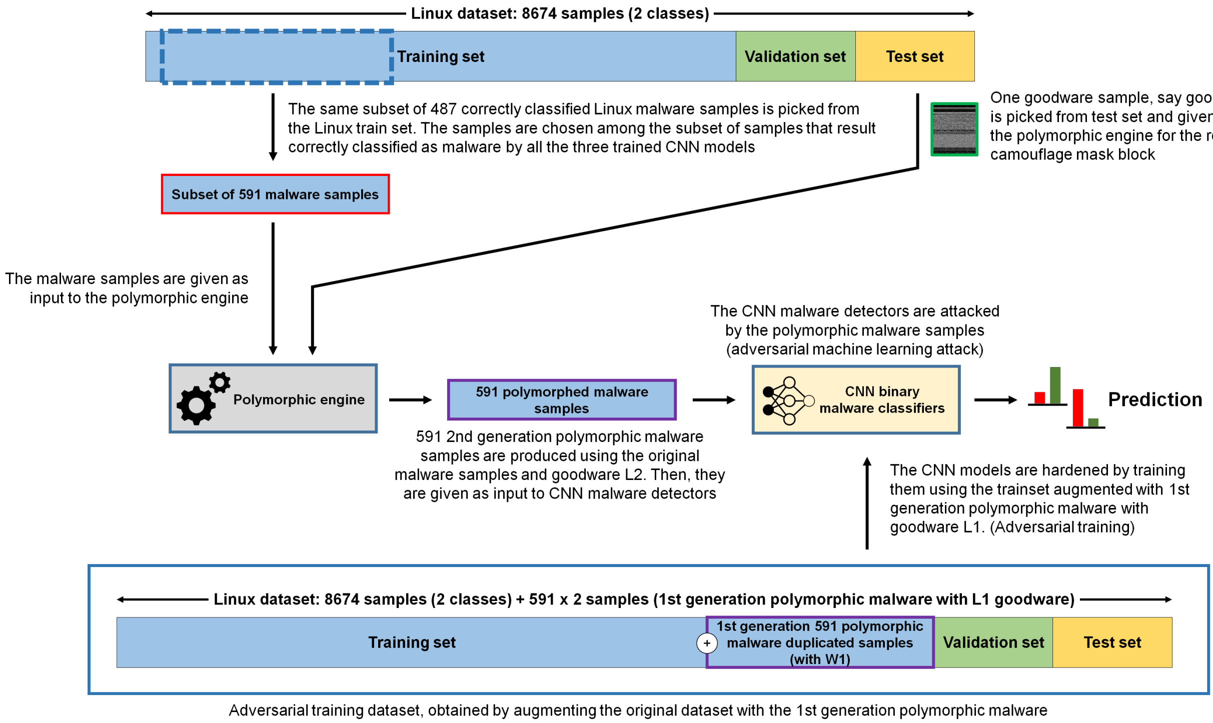
Linux dataset: 8674 samples (2 classes)
Training set
Validation set
Test set
The same subset of 487 correctly classified Linux malware samples is picked from the Linux train set. The samples are chosen among the subset of samples that result correctly classified as malware by all the three trained CNN models
One goodware sample, say goodware ''L2'', is picked from test set and given as input to the polymorphic engine for the role of camouflage mask block
Subset of 591 malware samples
The malware samples are given as input to the polymorphic engine
The CNN malware detectors are attacked by the polymorphic malware samples (adversarial machine learning attack)
Polymorphic engine
591 polymorphed malware samples
CNN binary malware classifiers
Prediction
591 2nd generation polymorphic malware samples are produced using the original malware samples and goodware L2. Then, they are given as input to CNN malware detectors
The CNN models are hardened by training them using the trainset augmented with 1st generation polymorphic malware with goodware L1. (Adversarial training)
Linux dataset: 8674 samples (2 classes) + 591 x 2 samples (1st generation polymorphic malware with L1 goodware)
Training set
+
1st generation 591 polymorphic malware duplicated samples (with W1)
Validation set
Test set
Adversarial training dataset, obtained by augmenting the original dataset with the 1st generation polymorphic malware

Figure 7: Diagram of CNN adversarial training and 2nd generation attack.

adversarial training, a new attack against the adversarial-trained models is perpetrated through the second generation malware. The objective is to evaluate to what extent this adversarial training countermeasure is able to harden CNN malware classification models against polymorphic malware of different subsequent generations.

Polymorphism vs CNN adversarial training CNN models, trained on adversarial train set, are tested for classification on the second generation polymorphic samples. From the results, we can also get an insight of how much adversarial trained CNN models can effectively automatically learn to achieve some level of robustness against malware polymorphism technique.

Figure 7 gives an illustration of the second generation polymorphic malware attack to CNN models trained with first generation polymorphic malware samples.

8. Results after hardening

After CNN models hardening using adversarial training techniques, some improvements under specific circumstances are observed in the classification performance results.

For the Linux samples (Table 5), we can see that the InceptionV3 model benefited from the adversarial training countermeasure the most. In fact, its evasion rate dropped of more than 30% starting from 100% down to less than 69%. The other two architectures had little or no improvements at all, with an evasion rate steady to 100% for VGG16 and above 95% in the case of ResNet50v2.

For the Windows samples (6), the results show something different, at least for two out of three architectures. After hardening, VGG16 shows an evasion rate drop of about 90%, going down to the only 9% of misclassified polymorphic malware samples. InceptionV3 shows significant classification improvements over polymorphic samples, with an evasion rate down to less than 32% and a reduction of more than 67% of the original value. On the other hand, ResNet50v2 is the only tested architecture that did not get any improvement after adversarial training, with an evasion rate over 99%.

These results confirm the goodness of CNN-based classification under the assumption of a "standard" malware encoding, in line with the discussion of the previous section, here generated by first-generation polymorphic attacks and mitigated by adversarial training. In fact, the polymorphic adversarial attacks exhibit the best malicious effectiveness against those models (either unhardened and hardened CNN) that performed better under the original malware samples.

Despite the positive effects of the adversarial training, the issues that polymorphism poses on CNN-based malware detection are far from solved. In fact, even after the adversarial training and in almost any tested architecture, the evasion rate on polymorphic malware is still too high to consider CNN-powered malware detectors reliable, without essential and effective countermeasures and architectural adjustments.

Furthermore, the marked difference in classification performance between different architectures with the same

Table 5
Attack results for polymorphic malware samples designed for Linux.

CNN	Classes	# testset samples		Mean Class. Margin	Well Classified	Misclassified	Evasion rate
VGG16	*Malware*	591	591	-0.980454%	0	591	100%
	Goodware	0		NA	0	0	NA
ResNet50v2	*Malware*	591	591	-0.771529%	27	564	95.53%
	Goodware	0		NA	0	0	NA
InceptionV3	*Malware*	591	591	-0.266261%	184	407	68.87%
	Goodware	600		NA	0	0	NA

Table 6
Attack results for polymorphic malware samples designed for Windows.

Hardened CNN with adversarial training on 1st gen polymorphic attack (2nd gen attack)							
CNN	**Classes**	**# testset samples**		**Mean Class. Margin**	**Well Classified**	**Misclassified**	**Evasion rate**
VGG16	*Malware*	487	487	0.580322	443	44	9.03%
	Goodware	0		NA	0	0	NA
ResNet50v2	*Malware*	487	487	-0.925513	2	485	99.56%
	Goodware	0		NA	0	0	NA
InceptionV3	*Malware*	487	487	0.215359	332	155	31.83%
	Goodware	0		NA	0	0	NA

dataset suggests a severe unpredictability of the benefits resulting from the adversarial training, based on the operating conditions. This unpredictability and the lack of control over the performance of classifiers may arise among different architectures and furthermore, between different generations of polymorphic malware. This makes it difficult to find attributes and features for an objective evaluation of the reliability of the models.

9. Future work

Future works on CNN-powered malware detection should concentrate on the exploration of possible techniques capable of counteracting the effects of adversarial machine learning. This is critical in light of the ongoing exploitation of machine learning technologies for static malware analysis, since currently, unhardened CNN architectures are inadequate to address security issues such as the detection of polymorphic malware and adversarial samples.

Based on the work we have done, we can argue that the research on CNN and ML algorithms for static malware analysis and signature-based detection has to be integrated by other complementary approaches: indeed, the combination of ML and static analysis, provides several opportunities for relevant improvements, with special regard to adversarial training technique; on the other hand, it has to take into account the sensitivity of CNNs and the opportunity (for the attacker) to mislead them, with little effort. These issues represent a valid starting point for the design of better malware detection systems based on static analysis and signature detection.

10. Conclusions

So far, we have addressed some issues and challenges that malware poses to cybersecurity. To face these issues and to accomplish the task of building reliable malware detectors, we explored some possibility offered by machine learning technologies, such as CNN architectures, namely malware and goodware classifiers.

The focus of the present research is set on the effectiveness of CNNs in the field of malware detection, when malware files are preprocessed, statically transformed into image samples, and fed into CNN for classification. The experiments we conducted focused on 2-classes binary classification (malware and goodware) and three of the best CNN architectures available were considered. After collecting all the necessary samples of malware and harmless programs, two separate datasets were formed for two OS platforms, Linux and Windows. After the training phase, the models were evaluated on the samples in test sets and, then, the encouraging results that came from them were discussed.

Our main contribution focused on the problems involved by the use adversarial machine learning with CNN-based classifiers designed for malware detection. To this purpose a polymorphic engine capable of generating polymorphic malware aimed at attacking and deceiving CNN classifiers was designed and implemented experimentally. The proposed polymorphic malware design was able to deceive, evade a malware detector and ultimately to render it useless, in case no kind of hardening against adversarial polymorphic malware samples is performed.

It must be emphasized that the hardening approach here used assumes the defender's knowledge of the proposed polymorphic attack (i.e. the defender is aware of the set of adversarial samples designed and used by the attacker and knows which are the samples it must be able to block). In turn, this implies the analyzed scenario is the best for the defender (and consequently the worst for the attacker). In conclusion, the present evidence supports the statement that, even after hardening, a CNN-based malware detector, manifests significant malware misclassification issues against polymorphic samples.

After showing countermeasures are required, adversarial training on the considered CNN models was introduced, which led to a mitigation of the deleterious effects on the CNN classification performances caused by the polymorphic malware attack. Unfortunately, the high cost of generating strong adversarial examples makes standard adversarial training impractical on large-scale problems. Furthermore, this measure proves not sufficient to make these applications of CNN classifiers reasonably reliable for malware detection. In conclusion, the use of adversarial training to quash the described attacks deserves further study and it is a starting point for improving cybersecurity and malware detection in the computer industry and in the era of industrial internet of things.

References

[1] A. Corallo, M. Lazoi and M. Lezzi. (2020). Cybersecurity in the context of industry 4.0: A structured classification of critical assets and business impacts. *Computers in industry*, 114, 103165

[2] A. Corallo, M. Lazoi, M. Lezzi and P. Pontrandolfo. (2021). Cybersecurity Challenges for Manufacturing Systems 4.0: Assessment of the Business Impact Level. *IEEE Transactions on Engineering Management*.

[3] M. Lezzi, M. Lazoi and A.Corallo. (2018). Cybersecurity for Industry 4.0 in the current literature: A reference framework. *Computers in Industry*, 103, 97-110.

[4] P. Mell, K .Kent and J. Nusbaum. *Guide to malware incident prevention and handling* (pp. 800-83). Gaithersburg, Maryland: US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

[5] Ö.A. Aslan and R. Samet. (2020). A comprehensive review on malware detection approaches. *IEEE Access, 8*, 6249-6271.

[6] R. Tahir. (2018). A Study on Malware and Malware Detection Techniques. *International Journal of Education and Management Engineering 8*(2), 20.

[7] R. Sihwail, K. Omar and K.A.Z. Ariffin. (2018). A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology, 8*(4-2), 1662.

[8] M Sikorski and A. Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. *no starch press*

[9] J Bermejo Higuera, C. Abad Aramburu, J.R. Bermejo Higuera, M.A. Sicilia Urban and J.A. Sicilia Montalvo, (2020). Systematic approach to malware analysis (SAMA). *Applied Sciences, 10*(4), 1360.

[10] Y. Supriya, G. Kumar, D. Sowjanya, D. Yadav and D.L. Kameshwari. Malware Detection Techniques: A Survey. In *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2020, pp. 25-30.

[11] T. Alsmadi and N. Alqudah. (2021, July). A Survey on malware detection techniques. In *2021 International Conference on Information Technology (ICIT)* (pp. 371-376). IEEE.

[12] B.M. Serinelli, A. Collen & N.A. Nijdam (2020). Training guidance with kdd cup 1999 and nsl-kdd data sets of anidinr: Anomaly-based network intrusion detection system. *Procedia Computer Science, 175*, 560-565.

[13] B.M. Serinelli, A. Collen & N.A. Nijdam (2021). On the analysis of open source datasets: validating IDS implementation for well-known and zero day attack detection. *Procedia Computer Science, 191*, 192-199.

[14] F. Tommasi, C. Catalano & I. Taurino (2022). Browser-in-the-Middle (BitM) attack. *International Journal of Information Security 21*(2). 179-189.

[15] F. Tommasi, C. Catalano, U. Corvaglia & I. Taurino. (2022) MinerAlert: an hybrid approach for web mining detection. *Journal of Computer Virology and Hacking Techniques,*. 1-14.

[16] L. Nataraj, S. Karthikeyan, G. Jacob and B.S. Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (pp. 1-7).

[17] J.S. Luo and D.C.T. Lo(2017, December). Binary malware image classification using machine learning with local binary pattern. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 4664-4667). IEEE.

[18] M. Krčál, O. Švec, M. Bálek and O. Jašek. (2018). Deep convolutional malware classifiers can learn from raw executables and labels only.

[19] V.S.P. Davuluru, B.N. Narayanan and E.J. Balster. (2019, July). Convolutional neural networks as classification tools and feature extractors for distinguishing malware programs. In *2019 IEEE National Aerospace and Electronics Conference (NAECON)* (pp. 273-278). IEEE.

[20] A. Bensaoud, N. Abudawaood and J. Kalita. (2020). Classifying malware images with convolutional neural network models. *International Journal of Network Security*, 22(6), 1022-1031.

[21] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro and C. Nicholas. (2017). Malware detection by eating a whole EXE (2017). *arXiv preprint* arXiv:1710.09435.

[22] M. Kumari, G. Hsieh and C.A. Okonkwo. (2017, December). Deep Learning Approach To Malware Multi-Class Classification Using Image Processing Techniques. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 13-18). IEEE.

[23] Z. Cui, F. Xue, X. Cai, Y. Cao, G.G. Wang, and J. Chen. (2018). Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7), 3187-3196.

[24] L. Chen, R. Sahita, J. Parikh and M. Marino. (2020). STAMINA: Scalable Deep Learning Approach for Malware Classification. Intel Labs Whitepaper, https://www.intel.com/content/www/us/en/artificial-intelligence/documents/stamina-deep-learningfor-malware-protection-whitepaper.

[25] J. Singh, D. Thakur, F. Ali, T. Gera and K.S. Kwak. (2020). Deep feature extraction and classification of android malware images. *Sensors*, 20(24), 7013.

[26] S.M Moosavi-Dezfooli, A. Fawzi, O. Fawzi and P. Frossard. (2017). Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1765-1773).

[27] R. Labaca-Castro, L. Muñoz-González, F. Pendlebury, G.D. Rodosek, F.Pierazzi and L.Cavallaro. (2021). Universal Adversarial Perturbations for Malware. *arXiv preprint* arXiv:2102.06747.

[28] L. Demetrio, B. Biggio, G. Lagorio, F. Roli and A. Armando. (2019). Explaining vulnerabilities of deep learning to adversarial malware binaries. *arXiv preprint* arXiv:1901.03583.

[29] I. Goodfellow, P. McDaniel & N. Papernot (25 June 2018). Making machine learning robust against adversarial inputs. *Communications of the ACM, 61*(7), 56-66.

[30] A. Madry, A. Makelov, L. Schmidt, D. Tsipras & A. Vladu. (2017). Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv preprint arXiv:1706.06083*.

[31] N. Carlini & D. Wagner (2017, May). Towards Evaluating the Robustness of Neural Networks. In *2017 ieee symposium on security and privacy (sp)* (pp. 39-57). Ieee.

[32] K.D. Gupta, D. Dasgupta & Z. Akhtar (2021). Determining Sequence of Image Processing Technique (IPT) to Detect Adversarial Attacks. *SN Computer Science, 2*(5), 1-20.

[33] P. Dwivedi and H. Sharan. (2021). Analysis and Detection of Evolutionary Malware: *International Journal of Computer Applications, 975*, 8887.

[34] B.B. Rad, M. Masrom and S. Ibrahim. (2012). Camouflage in malware: from encryption to metamorphism. *International Journal of Computer Science and Network Security, 12*(8), 74-83.

[35] A. Sharma and S.K. Sahay. Evolution and detection of polymorphic and metamorphic malwares: A survey. *arXiv preprint arXiv:1406.7061*.

[36] B.B. Rad, M. Masrom & S Ibrahim (2012). Camouflage In Malware: From Encryption To Metamorphism. *International Journal of Computer Science and Network Security, 12*(8), 74-83.

[37] D. Lin & M. Stamp (2011). Hunting for Undetectable Metamorphic Viruses. *Journal in Computer Virology 7*(3), 201-214.

[38] VirusShare.com - Research Papers and Publication that have cited VirusShare https://virusshare.com/research

[39] M. Kianpour and S.F. Wen. (2019, September). Timing attacks on machine learning: State of the art. In *Proceedings of SAI Intelligent Systems Conference* (pp. 111-125). Springer, Cham.

[40] A. Kurakin, I. Goodfellow and S. Bengio, Adversarial machine learning at scale. *arXiv preprint* arXiv:1611.01236.

[41] BB. iggio, I. Corona, B. Nelson, B.I. Rubinstein, D. Maiorca, G. Fumera, ... & F. Roli (2014). Security evaluation of support vector machines in adversarial environments. *In Support Vector Machines Applications* (pp. 105-153). Springer, Cham.