Computers & Security 126 (2023) 103062

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Disarming visualization-based approaches in malware detection systems

Lara Saidia Fascí[a], Marco Fisichella[b,*], Gianluca Lax[a], Chenyi Qian[b]

[a] DIIES Dept., University of Reggio Calabria, Via Graziella, Loc. Feo di Vito, Reggio Calabria, 89122, Italy
[b] L3S Research Center of Leibniz University Hannover, Appelstrasse 9a, Hannover,30167, Germany

ARTICLE INFO

Article history:
Received 28 September 2022
Revised 30 November 2022
Accepted 10 December 2022
Available online 13 December 2022

Keywords:
Malware classification
Machine learning
Deep learning
GAN

ABSTRACT

Visualization-based approaches have recently been used in conjunction with signature-based techniques to detect variants of malware files. Indeed, it is sufficient to modify some byte of executable files to modify the signature and, thus, to elude a signature-based detector. In this paper, we design a GAN-based architecture that allows an attacker to generate variants of a malware in which the malware patterns found by visualization-based approaches are hidden, thus producing a new version of the malware that is not detected by both signature-based and visualization-based techniques. The experiments carried out on a well-known malware dataset show a success rate of 100% in generating new variants of malware files that are not detected from the state-of-the-art visualization-based technique.

1. Introduction

Malware detection has always been a hot topic and one of the main interests of cybersecurity experts. In recent years, the number of cyber attacks and, more importantly, the severity of these attacks have increased. Cyber criminals have become more ruthless than ever, with ransomware being one of the most dangerous threats on the current scene. In 2021, for example, the Irish Health Service Executive fell victim to a ransomware attack that forced the organization to shut down all of its IT systems (Mixon, 2021). In addition, a recent report by Claroty (2021) declares a 41% increase in industrial control systems (ICS) vulnerabilities discovered in the first half of 2021 compared to the previous six months. Some examples of exploitation of such vulnerabilities are the Brenntag and the Colonial Pipeline attacks, in which the two companies paid $4.4 million (75 bitcoin) to restore data stolen by a ransomware (Washington et al., 2022). These events show that there is no limit to what cyber attackers can do and malicious attacks can pose a threat not only to the security of systems, but also to their safety.

Malware detection techniques can be based on static or dynamic analysis. Currently, there is no consensus on which is the best technique. The static analysis of software does not involve the actual execution of the program. Static analysis can provide information such as opcode sequences (extracted by disassembling the binary file) and control flow graphs.

Dynamic analysis requires the program to be executed, often in a virtual environment. The dynamic analysis of code can help us learn whether APIs are being called, whether system calls are being made, whether instructions are being traced, whether registry changes are being made, or whether memory is being written (Damodaran et al., 2017).

Both static and dynamic analyses can fail because attackers have found different approaches to evade detection. For example, static malware analysis, which analyses the executable file, can be bypassed by well-disguised malware (Dube et al., 2012; Gibert et al., 2021). In dynamic analysis, on the other hand, a malware could bypass detection by simply changing its behaviour (García and DeCastro-García, 2021; Han et al., 2019). All these circumstances have led to the need to find new effective approaches to malware detection. These aspects are discussed in more detail in Section 2.1.

The field of computer science has become increasingly focused on machine learning, since machine-learning-based systems can be used in a wide range of applications, such as product recommendations, image recognition, natural language processing, clustering, and so on. Researchers have proposed the use of machine learning for malware classification and detection. One of the most promising approaches is visualization-based malware detection proposed by Nataraj et al. (2011). This approach converts binary files into images and assigns a label to each image depending on whether it is from malware or not. These images are used as a training set for a

* Corresponding author.
E-mail address: mfisichella@L3S.de (M. Fisichella).

https://doi.org/10.1016/j.cose.2022.103062

classification problem, and the generated prediction model is used to decide whether a file should be considered as malware or not.

An analysis of the performance of several visualization-based techniques in malware detection is provided by Prajapati and Stamp (2021). Their analyses find that the best accuracy was about 0.86 by using Multilayer Perceptron Neural Networks, about 0.90 using Convolutional Neural Networks and Recurrent Neural Networks, and about 0.92 by using transfer learning models (namely RestNet152 and VGG-19) (Prajapati and Stamp, 2021). A more recent study presented by Pinhero et al. (2021) proposes various fine-tuned Convolutional Neural Networks for visualization-based malware detection, and the measured accuracies are higher than 0.94. The high performance of visualization-based approaches in detecting malware highlights that designing a technique that can bypass their control opens up new attack threats.

In this work, we investigate this possibility and propose a new technique based on Generative Adversarial Networks to modify a malware so that it is not detected by the techniques designed by Pinhero et al. (2021). For the experimental evaluations, we develop a framework that implements our technique, we collect a large dataset of malware and goodware (i.e., benign software), and we show how a malware can be modified so that it is not detected yet preserving its malicious execution.

To highlight the significance of our contribution, we observe that the visualization-based technique can be used (1) alone or (2) together with another technique, say T (for example, T could be a signature-based technique). In the first case, our research is very relevant because an undetected malware is generated. Concerning the second case, we have two possibilities: (2a) T is able to detect a malware M. In this case, the visualization-based approach is not necessary. In the second case (2b), T does not detect M but the visualization-based approach does it. In this case, our technique can allow an attacker to modify the malware in such a way that the visualization-based approach is no longer able to detect it. Thus, in all cases (i.e., 1, 2a, and 2b), the visualization-based approach does not give an effective contribution in malware detection.

The remainder of the paper is arranged as follows. In Section 2, we discuss Malware detection and introduce the concept of Generative Adversarial Networks on which our strategy is based. Section 3 discusses related work. In Section 4, we present the architecture of our technique. Section 5 shows how our approach was implemented in the specific scenario of visualization-based malware detection systems. Section 6 presents the experiments carried out and a discussion on the validity of our proposal. Finally, we conclude this study and propose future directions in Section 7.

2. Preliminaries

In this section, we provide the background about two topics that are widely discussed in our paper, which are malware detection and GANs.

2.1. Malware detection

Malware analysis is a necessary step towards malware detection since it helps identifying the defining features of a malware (Tahir, 2018). Information such as byte code, opcode, API calls, file data, and registry data can be used to understand the purpose of a file and allow its classification as malware or benign file (Aboaoja et al., 2022a). There are three types of malware analysis:

- **Static Analysis**: this type of procedure consists in analyzing the file without executing it. Hence, only static information can be extracted, for example metadata (e.g. file name, type, and size), file headers, strings, MD5 checksums and hashes (N-able, 2021).
- **Dynamic Analysis**: dynamic or behavioral analysis is performed by executing the software in a controlled environment (e.g., virtual machines). Once the software is running, function calls and control flows are monitored as well as the instructions and parameters of functions. This type of analysis is more time and resource consuming since the environment used for the analysis needs to be purposely designed.
- **Hybrid Analysis**: it is simply a combination of the two techniques discussed above. Firstly, the executable is examined through static analysis (e.g. by checking the malware signature). Secondly, it is run in a virtual environment to check its behavior.

Malware detection is the process of discovery and identification of malicious activity performed by an executable file under investigation. There are many malware detection techniques but none of them can be considered 100% effective and undefeatable. Regardless, the three main approaches to malware detection are:

- **Signature-based malware detection**: each software has a sequence of bits called *signature* which constitute its digital footprint. Signature based detection uses these signatures to identify malware. For instance, signatures can be generated using frequency vectors (Wael et al., 2018; 2017) or values of opcodes (Khodamoradi et al., 2015). This type of malware detection is commonly used by antivirus software to extract the signature of the examined file and compare it with the signatures stored in a database of known threats.
- **Behavior-based malware detection**: the file under investigation is executed in a controlled environment (i.e. a sandbox) and its behavior is monitored. The behavior observed is then analyzed by the detection system in order to identify malicious activity. Examples of actions considered as potentially malicious are: any attempt to discover a sandbox environment; installing rootkits; deleting, altering, or adding system files; modifying other executable programs; downloading and installing unknown software; modifying the boot record or other initialization files to alter boot-up.
- **Heuristic-based malware detection**: this method of detection is carried out by generating generic rules that investigate data extracted (via dynamic or static analysis). The generated rules can be developed automatically using machine learning techniques, the YARA tool and other tools or manually based on the experience and knowledge of expert analysts (Aboaoja et al., 2022b).

Malware authors constantly look for ways to make malware detection fail its task. Some examples of such techniques are (Aboaoja et al., 2022b):

- *evasion techniques*: evasive malware are able to figure out whether they are being executed in a real machine or in a controlled environment. Hence, they can change their behavior based on this evaluation. One possible technique is the timing-based method in which a malware can go through extended sleep, schedule its own execution (e.g., on a specific date and time) or use stalling code. Furthermore, some evasive malware are able to perform their actions only after a set of user interactions occur;
- *obfuscation techniques*: the purpose is to modify or conceal the original code to make the executable difficult to analyze yet maintaining the functionality of the code;
- *zero-day attacks*: these attacks are hard to detect as they are carried out by malware which exploits vulnerabilities that are unknown by malware detector or to the vendor of the System under attack.

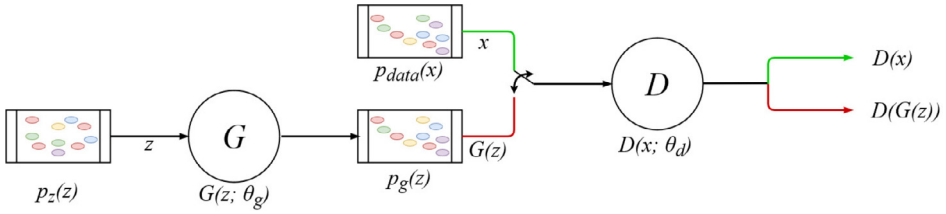

Fig. 1. Architecture of a GAN.

2.2. Generative adversarial networks

A Generative Adversarial Network (GAN) is a model category that specializes in complex tasks and can be applied to cybersecurity, privacy, modelling, simulation and natural language processing. A further benefit of Generative Adversarial Networks is that they can overcome the difficulty of training machine learning models without data. The GANs are capable of generating artificial adversarial samples to support training.

The concept of GAN was first introduced by Goodfellow et al. (2014) as a framework consisting of two models: a *generative model G* and a *discriminative model D*. The purpose of the former is to determine an estimated probability distribution that matches an input real data distribution (Cai et al., 2021); the latter is used to evaluate the probability that a sample comes from the real data distribution or from the distribution of the generative model.

In the original framework of GAN, the models used for the generator and discriminator are multilayer perceptrons. These neural networks are trained as follows:

- The training procedure of the generative model aims to maximize the probability of fooling the discriminative model, which is equivalent to minimizing the probability that the discriminator performs its task correctly;
- Training the discriminative model aims to maximize the probability of distinguishing real data from generated samples.

The generator network can be represented as in Fig. 1, where **z** is a vector of latent variables, while **x** is a vector of observed variables. Typically, each element of **x** depends on each element of **z**.

Let G be a differentiable function and let θ_g be the parameters of the multilayer perceptron of the generative model. These parameters can be learned by gradient descent. p_g represents the distribution of the generator over the data. Then, a noise prior probability distribution is defined for the input noise variables, namely $p_z(z)$. The mapping to the data space is denoted by $G(z;\theta_g)$. Let $p_{data}(x)$ be the real data distribution.

The mapping of the multilayer perceptron used for the discriminator is $D(x;\theta_d)$ and the output is a single scalar (Goodfellow et al., 2014). $D(x)$ is the probability that the sample is real.

2.2.1. Binary cross entropy

A binary classification model (Wikipedia Foundation, 2022) is a machine learning model that is tasked with classifying the elements of a set into two classes. This type of models needs to predict the label $\hat{y}$ for a given sample. Assuming that the actual label for a single sample x is y, the binary cross entropy function, often used as a loss function for binary classifiers, is defined as follows:

$$\mathcal{L}(\hat{y}, y) = y \ \log \ \hat{y} + (1-y) \ \log(1-\hat{y})$$

In the case of GANs, the discriminator behaves similarly to a binary classifier. Suppose that the label for a single data point x, which comes from the real data distribution $p_{data}(x)$, is $y = 1$. The label predicted by the discriminator is denoted by $\hat{y} = D(x)$. The loss function for a sample from the real data distribution is then:

$$\mathcal{L}\big(D(x), 1\big) = \log\big(D(x)\big)$$

An assumption is that the label for a single data point $G(z)$ coming from the generator's data distribution $p_g(z)$ is $y = 0$. The prediction for the generated data can be written as $\hat{y} = D(G(z))$. The loss function for an adverse sample is then:

$$\mathcal{L}\Big(D\big(G(z)\big), 0\Big) = \log\Big(1 - D\big(G(z)\big)\Big)$$

The goal of the discriminator is to distinguish between true and false samples (i.e., to classify the samples correctly). This goal implies that $\log(D(x))$ and $\log(1 - D(G(z)))$ are maximized, so that the loss function takes a practically infinite value when the predictions are incorrect. This is equivalent to:

$$\max\{\log\big(D(x)\big) \ + \ \log\big(1 - D(G(z))\big)\}$$

The generator is trained to outsmart the discriminator. With reference to the notation used so far, this means that the goal of the generator is to obtain $D(G(z)) = 1$. The implication is that the goal of the generator, which is opposite to that of the discriminator, is

$$\min\{\log\big(D(x)\big) + \log\big(1 - D(G(z))\big)\}$$

2.2.2. Value function in GANs

The formulation used so far was written considering only one data point from the distributions. To consider the whole data set, the formulation needs to be changed. The required change is to consider the expected value for each quantity in the formula. Thereupon, the cost functions $J^{(D)}$ and $J^{(G)}$, which apply to the discriminator and generator, respectively, can be defined as follows:

$$J^{(D)} = -\tfrac{1}{2}\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] - \tfrac{1}{2}\mathbb{E}_{z\sim p_z(z)}[\log D(1 - D(G(z)))]$$
$$J^{(G)} = -J^{(D)}$$

Finally, we can say that a GAN performs a mini-max two-player game (Goodfellow et al., 2014) between its two models to obtain effective generation of realistic data. This game can be formally represented by the following *value function*:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)} [\log D(1 - D(G(z)))]$$

The training algorithm for GANs was described by Goodfellow et al. (2014). In the training algorithm, the authors proposed that the discriminator should be updated more frequently than the generator. Therefore, the discriminator is updated k times every time the generator is updated.

Thus, for each training iteration, a mini-batch of m noise samples is obtained from the noise prior $p_g(z)$ for each of the k steps. Then, a mini-batch of m samples of training data is obtained from the data generating distribution $p_{data}(x)$. The discriminator is used to make predictions for these mini-batches and its weights are updated by ascending its stochastic gradient. After the k steps required to update the discriminator, a mini-sample is taken from

the noise prior $p_g(z)$, the generator is applied to the mini-sample samples, and the gradient descent is used to update the generator.

3. Related work

In this section, we survey visualization-based techniques for malware detection. The visualization-based approach for malware detection was proposed by Nataraj et al. (2011). In the proposed technique, each executable malware is read in its binary form as a vector of 8-bit unsigned integers. This vector is then organized into a 2-D array that can be visualized as a grayscale image. Since executable files vary in size, only the width of the image is fixed, while the height can vary. The obtained images were then used to classify the samples analyzed as belonging to a specific malware family.

Since this proposal, this topic has been the subject of many research works. For instance, IMCFN (Image-based Malware Classification using Fine-tuned Convolutional Neural Network Architecture) is the framework that implements the method proposed by Vasan et al. (2020a). This method converts the raw malware binaries into color images that are used by the fine-tuned CNN architecture to detect and identify malware families. In another paper, after assessing that both researchers and malware authors have proved that malware scanners are limited and easily evaded by simple obfuscation techniques, Vasan et al. (2020b) propose a novel ensemble convolutional neural network for image based detection of both packed and unpacked malware (IMCEC).

Recently, the use of the visualization-based approach has been proposed for Android malware as well. Bijitha and Nath (2022) have proposed a comprehensive study on the topic in which they analyze the effectiveness of different techniques. Geremias et al. (2022) consider the problem of malware android applications being present in official app stores and propose a new multi-view image-based Android malware detection system (that uses Deep Learning), implemented in three steps. Firstly, apps are evaluated according to several feature sets in a multi-view setting. Secondly, extracted feature sets are converted to an image format while maintaining the main components of the data distribution, useful to perform the classification task. Thirdly, built images are jointly represented in a single shot, each in a predefined image channel, enabling the application of deep learning architectures. Similarly, Yadav et al. (2022) have proposed a Deep Learning-based two-stage framework that detects Android malware and classifies variants using image-based malware representations of the Android DEX files. Mercaldo and Santone (2020) have also proposed the use of the visualization-based approach to perform malware detection in mobile environment and to identify the malware family for a malware and the variant inside the family.

Some research proposals have discussed the possibility of using GANs to perform attacks on malware detection systems (as done in our paper).

Lin et al. (2018) propose a framework called IDSGAN that can perform blackbox attacks against a machine learning based intrusion detection system of unknown structure. IDSGAN's generative model takes malicious traffic records and generates new adversarial patterns that can be used to evade detection. For this purpose, only some parts of the original traffic samples can be modified, namely the nonfunctional ones. The discriminator performs traffic sample classification and learns to mimic the behavior of the attacked intrusion detection system.

Hu and Tan (2017) propose a GAN-based framework to generate adversarial patterns that are able to evade detection by a machine learning based detection model with unknown parameters, namely a ML based blackbox detection system. The proposed algorithm is called *MalGAN* and can be considered offensive, which means that the authors take the perspective of an attacker. The only thing the attacker knows about the detection system to attack is the type of features it uses. However, the attacker can submit samples to the Blackbox Detector and retrieves the result of the analysis performed. MalGAN consists of a sample generator, namely the generative part of GAN, and a Substitute Detector, the discriminative model of GAN. These two elements are trained to allow bypassing the actual detector. Kawai et al. (2019) identify and address some issues of the MalGAN architecture and propose a new framework to improve MalGAN.

Concerning the accuracy of visualization-based techniques, several studies have been carried out. Prajapati and Stamp (2021) compared a wide variety of deep learning techniques, including multilayer perceptrons (MLP), convolutional neural networks (CNN), long short-term memory (LSTM), and gated recurrent units (GRU). The best accuracy measured is about 0.92.

A higher accuracy is measured for the technique proposed by Pinhero et al. (2021) (called *Pinhero* in the following). Therein, three types of conversions from binary file to images are defined. Each executable file is read in its binary form. Then the binary contents are read in groups of 8 bits each, byte by byte, and mapped to a decimal value in the range [0,255]. The resulting decimal values are stored in a 1-D vector. Depending on the type of image, the conversion is completed as follows:

- For *grayscale images*, each pair of consecutive values of the vector is used to identify the corresponding pixel value in a color map. Each identified value is used to create a 1-D pixel array. The image width is 512px, while the height is variable as it depends on the file size;
- For *RGB images*, each pair of consecutive values of the vector is used to identify three corresponding pixel values in 3 different color maps (one for each channel, namely red, green and blue). Each identified value is then used to create a 1-D pixel array;
- For *Markov images*, a frequency table is created by considering the frequency of occurrence of byte b_i followed by b_{i+1} and b_i followed by b_k where $0 \leq k \leq 255$. Then, the probability that b_i is followed by b_{i+1} is calculated. This quantity is given as *transition probability matrix* $TM[i][j]$, where MP is the maximum likelihood calculated from the transition probability matrix. Finally, the image is created, where each pixel is calculated according to the following equation:

$$P = \left(TM[i][j] * \left(\frac{255}{MP}\right)\right) \ mod \ 256$$

Their study showed that the best results were obtained for 256x256 Markov images.

In our study, we show that the *Pinhero* technique is not infallible and that it is possible to generate malware that are not detected from this technique. Moreover, we design a system to generate such malware and experimentally show that our proposal is effective.

Overview of the proposal Visualization-based malware detection systems can be used alone or together with other techniques to improve the overall accuracy in detecting malware. The goal of our study is to obfuscate a malware in such a way that visualization-based approaches are unable to correctly classify such a malware. In this section, we present the architecture of the system designed for this purpose, which consists of a *Substitute Detector*, a *Generator*, in addition to a *Blackbox Detector*.

In this architecture, the Blackbox Detector models the visualization-based system to be attacked and can be viewed as external anti-virus software that accepts executable files as input and provides a binary classification into benign or malicious file as output. Since it is based on visualization, the internal structure of the

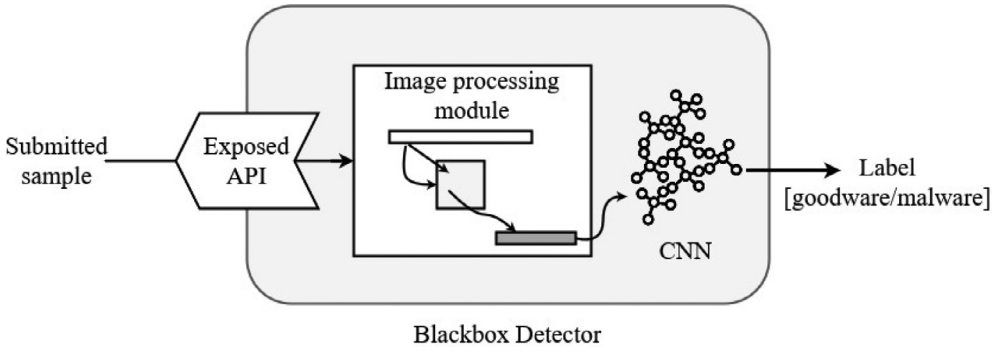

Fig. 2. Blackbox Detector.

Blackbox Detector, shown schematically in Fig. 2, includes a module for converting executables into images and a neural network for detection. New executables are submitted to the Blackbox Detector and converted into an image (there are several strategies to convert an executable into an image; Section 3 described the most commonly used approaches). Then, the processed image sample is used as input to a convolutional neural network that makes predictions for benign and malicious files.

4.1. Substitute detector

The Substitute Detector is a model trained to mimic the behavior of the Blackbox Detector, which is useful for the attacker to test new malware patterns. This is necessary because the output of the Blackbox Detector is only a boolean value (malware or not). By a *whitebox* that works like the Blackbox Detector, the attacker can determine to what extent a sample is close to being classified as malware or not, and this feedback is useful to generate a new sample.

The Substitute Detector consists of an executable-to-image conversion module and a neural network used for detection. The training of the Substitute Detector is shown schematically in Fig. 3. A set of executable files is used to feed the Blackbox Detector, which converts the binary files into images and returns a binary label indicating whether a file is malware. Then, the same set of files, along with the labels assigned by the Blackbox Detector, is sent to the Substitute Detector so that it can learn to emulate the Blackbox Detector in classifying the adversarial samples.

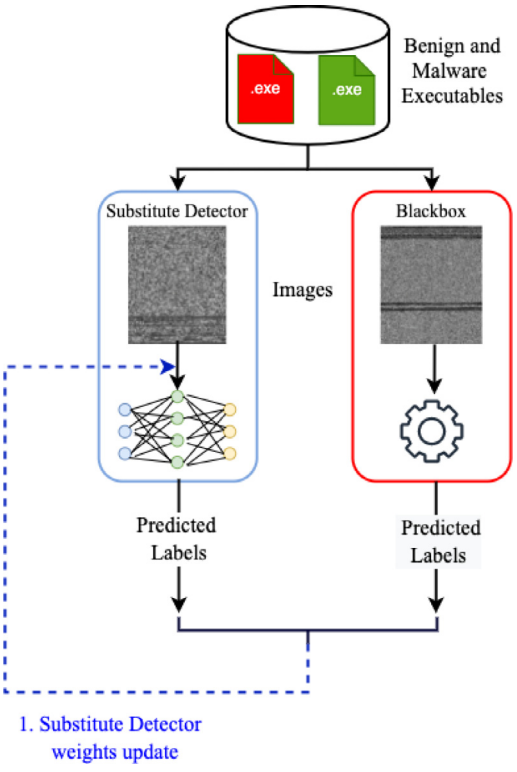

Fig. 3. Substitute Detector training.

4.2. Generator

The Generator is a generative network and its main purpose is to produce benign or malicious samples that are not correctly classified by the Blackbox Detector. The training of the Generator is shown schematically in Fig. 4.

The Generator has two inputs: the random noise values used to create the generated samples and the executable samples that are converted into images (the process of generating an image from an executable is shown in Section 5.4). The Generator then creates new samples that are output as images and transmitted to the second stage of the Substitute Detector (i.e., they are the input to the neural network). The Substitute Detector classifies the generated adversarial images as malware or goodware. The labels generated by the Substitute Detector are used along with the gradient information to train the Generator.

Obviously, the generated noise should be such that the correct execution of the executable file is not compromised. There are several possibilities to modify an executable file yet keeping the usual execution result:

1. by appending the noise at the end of the executable file, because the execution will ignore the bits after the *end-of-file* marker;
2. by including debug information, which is not useful for the execution;
3. by including the noise in a commented section of the code;
4. by including the noise in a section of an always-false test. The idea is to inject into the malware an IF statement in which the test is always false (e.g., `if` $x = x + 1$) and the execute sequence is a (suitably) long block of *NOPs* (not operations). The subsequent address references should be shifted to consider the space taken by the injected IF: however, in case the IF statement is injected as the last instruction of the executable file, no shift has to be done. Obviously, the block of NOPs is used to inject the noise. In a more complex attack, the IF condition could be something like `if` $a < 0$, where a is a positive variable.

It is worth noting that to implement these strategies, the attacker runs the compiler in unoptimized mode (i.e., debug mode or by setting suitable code generation options). Consequently, the first

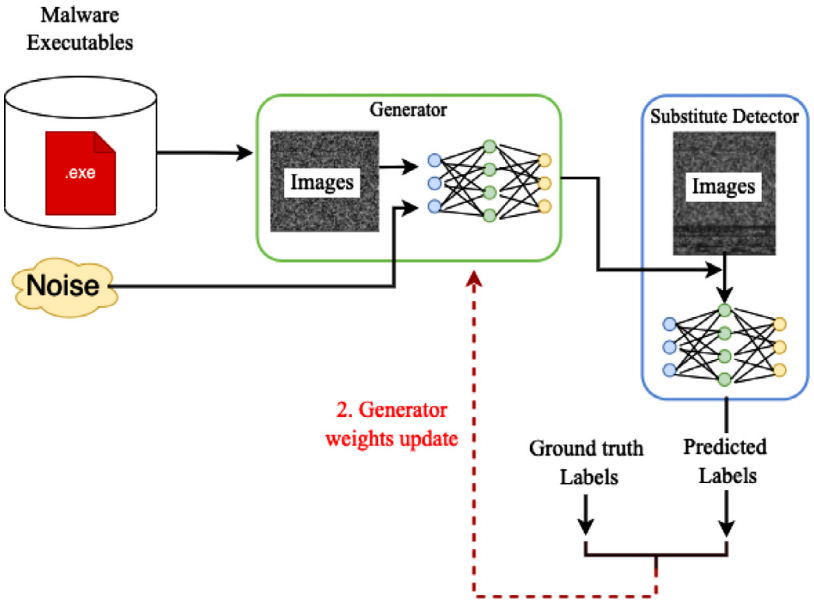

Fig. 4. Generator training.

three techniques are easy to be detected, and the last one should be used in a real attack. In the experiments described in Section 6, we applied the first strategy to show that it is effective even if it is less powerful than the others because the noise can injected only at the end of the file (in contrast, the other techniques allow for the injection into any part of the file).

4.3. Overview of the entire architecture

In the previous sections, we presented the individual components of the architecture. In this section, we report the complete framework implementing the proposed methodology, which is schematized in Fig. 5.

In this figure, we show how *Blackbox Detector, Substitute Detector*, and *Generator* interact with each other when they are trained in parallel to be time efficient. Indeed, in the serialized version of the approach so far described, first we train the substitute detector and, then, we start the training of the Generator. However, this methodology requires more time than the parallel modality that is described in the following.

When the parallel training is used, the input of the system can be (i) either a sample extracted by the dataset of benign and malware executable files (as shown in Fig. 3) or (ii) a sample produced by the *Generator*. In the latter case, the produced sample is an image, which is both sent to the *Substitute Detector* (as shown in Fig. 4) and converted into an exe file and sent to the *Blackbox Detector*. Now, if the label predicted by the *Substitute Detector* is equal to the label assigned by the *Blackbox Detector*, then the *Generator* updates the weights, else the *Generator* does not use this sample for its training. In contrast, the training of the *Substitute Detector* occurs in all the cases.

5. Implementation

As presented in Section 1, we deploy our system to mitigate the state-of-the-art in visualization-based malware detection, that is the *Pinhero* technique. In this section, we describe the implementation of the proposed system, which uses Python and the artificial intelligence software library Tensorflow. The project code is available at Code Repository (2022).

5.1. Dataset

The dataset used in our experiments consists of both malware and goodware. The malware samples are from the *MalImg* dataset (Nataraj et al., 2020; Nataraj et al., 2011). As for goodware, no datasets and no direct reliable sources of safe software were found. Therefore, we collected *.exe* files both by web scraping on different platforms (DriverPack Solution, 2022; Filehippo, 2022; Major Geeks, 2022; Portable Freeware, 2022; Softonic, 2022) and by using executables of two virtual machines right after installing the 32-bit version of Windows 8 and 10, respectively. To ensure that the collected *.exe* files are not malware, we scanned each file with VirusTotal software, as done by Pinhero et al. (2021). In total, we collected about 2000 samples, which are available at Repository (2022).

5.2. Blackbox detector

As no implementation of the *Pinhero* technique is publicly available, we implemented the Blackbox Detector using the method defined by Pinhero et al. (2021). More specifically, we implemented the M11 model, which performed best when trained with the same parameters. As defined in that paper, the convolutional neural network used for classifying the processed images is VGG3 with dropout and batch normalization from the VGG (Visual Geometry Group) architecture. VGG is an object recognition architecture based on CNN but deeper than most conventional CNN architectures, as its fully connected layers can count up to 138M parameters. The VGG acronym is usually followed by a number indicating the number of layers used in the network. Both the model used by Pinhero et al. (2021) and that used in our project have VGG3 as a baseline.

The strategy used by Pinhero et al. (2021) to convert a binary file to an image is schematized in Fig. 6. First, a 256x256 matrix $\mathcal{C}$ of random values between 0 and 255, said *color map* is generated. Then, the content of the executable binary is read byte by byte, and for each pair of consecutive elements of the binary data array, respectively of decimal value x and y, an element of the color map is identified as $\mathcal{C}[x][y]$. This element is then stored in a 1-D pixel array that is used to create a grayscale image.

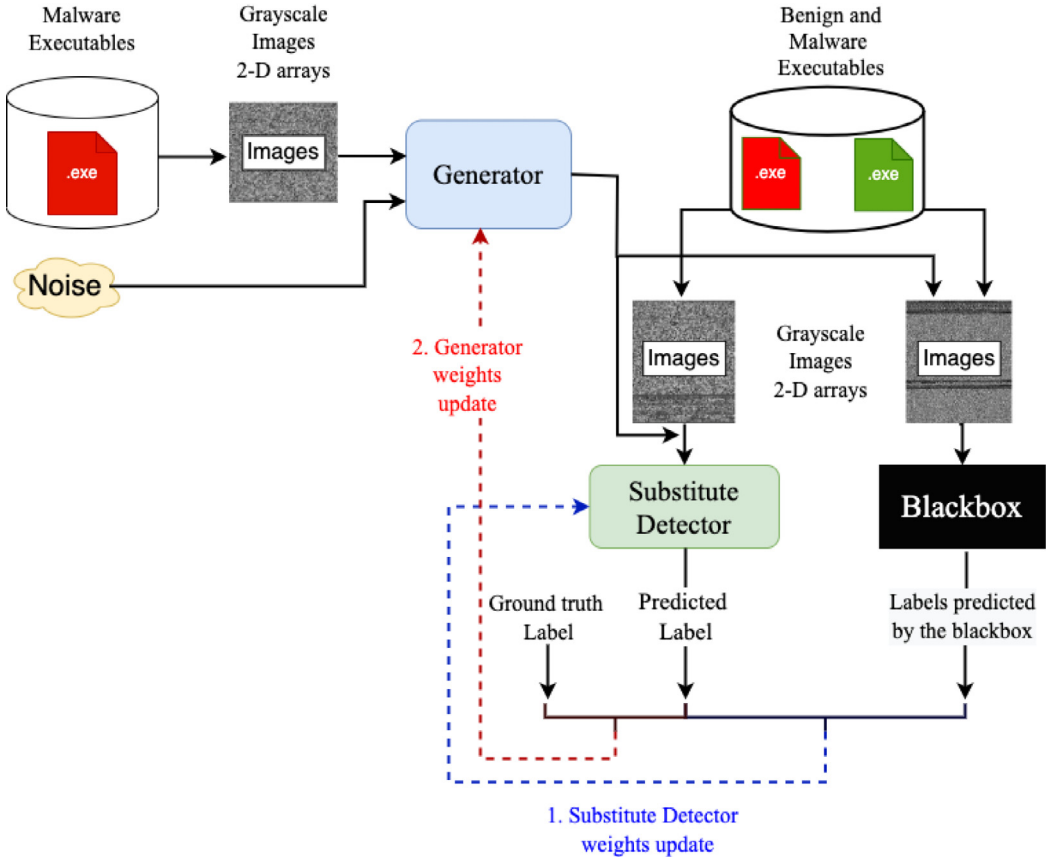

Fig. 5. Proposed architecture.

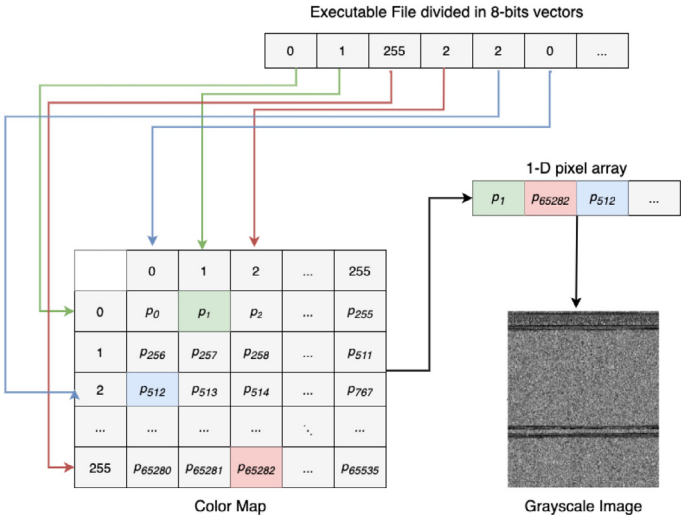

Fig. 6. Input of the Blackbox Detector: from binary data to grayscale image with color map.

We trained the model with the dataset described in Section 5.1 and measured the obtained loss, accuracy, and F-score, which are commonly used to verify the performance of malware classifiers and detectors. The results measured versus the number of epochs are reported in Fig. 7, and show that the performances of the Blackbox Detector are comparable to the results presented by Pinhero et al. (2021).

5.3. Substitute detector

Similar to the Blackbox Detector, the Substitute Detector is a convolutional neural network designed for binary classification. Figure 8 shows a representation of its architecture.

Batch normalization and strided convolutions are also used in addition to ReLU layers. Dense layers are used later in the dense part of the network. The decision for the number of neurons in the dense part was dictated by the results obtained after trying different combinations. The model is sequential. The first layer is a resizing layer to handle inputs of different sizes. The input size chosen is (256, 256) and the interpolation strategy is "bilinear". Note that the Substitute Detector does not use a color map to process the input images, since it is assumed that the conversion algorithm is an aspect hidden to the attacker.

The process of generating an image from an executable file is shown in Fig. 9. The file is read as a 1-D binary bit string vector. The string is split into 8-bit vectors and converted to decimal values, resulting in a single 1-D pixel array vector. Each entry corresponds to a pixel value from 0 to 255. The resulting 1-D pixel array is reshaped to a 2-D matrix and displayed as a grayscale image. As done by Pinhero et al. (2021), the width of the image is 512 pixels and the height is variable as it depends directly on the file size.

After resizing, the input is processed by the convolutional layers. Finally, the dense part of the network further processes the outputs of the previous layers and finally, only one neuron is used in the last layer to make the final prediction.

The ground truth labels used in calculating the losses are the labels resulting from the predictions of the Blackbox Detector for both adversarial malware and goodware files.

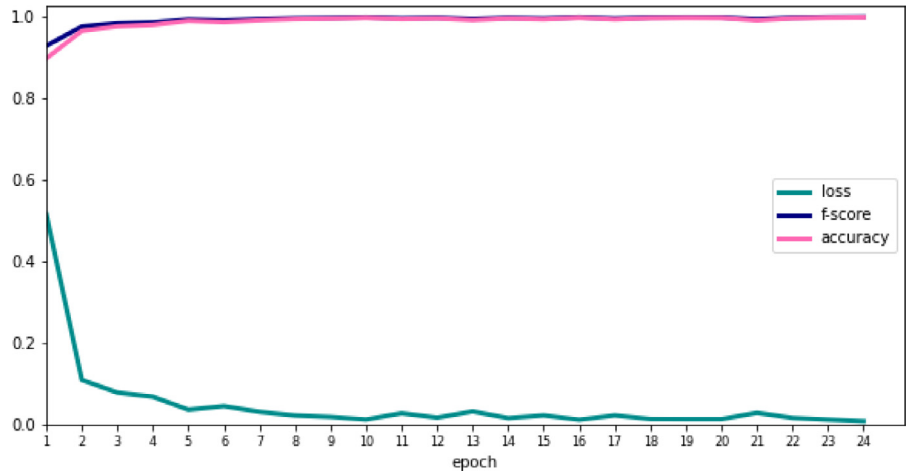

Fig. 7. Results of the Blackbox Detector training.

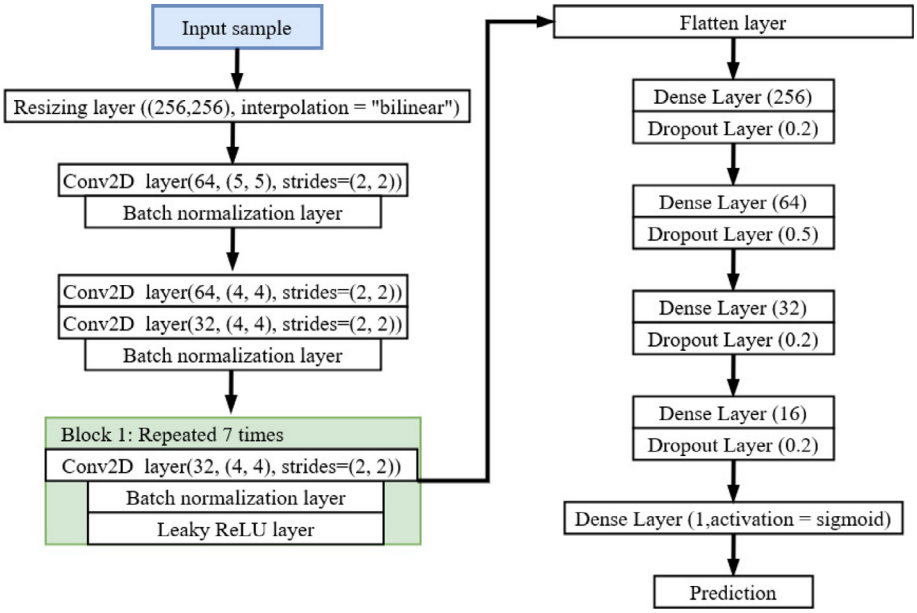

Fig. 8. Substitute Detector architecture.

We trained the model of the Substitute Detector and measured the obtained loss, accuracy, and F-score versus the number of epochs. The results of this training are reported in Fig. 10, and show that, after few epochs, the predictions of the Substitute Detector are almost the same as those of the Blackbox Detector, which is the desired goal.

5.4. Generator

Using the Generator, an attacker can create variants of the same malware, one or more of which can bypass the existing detection system. As done by Kawai et al. (2019), a single input sample is considered at a time until the generated malware pattern is classified as benign. The structure of the Generator is shown in Fig. 11.

The Generator has two inputs: the input noise and the original malware sample. The input noise is of the form [batch_size, 100], while the malware sample is converted to a grayscale image (the process of generating an image from an executable file is shown in Fig. 9). To simplify the presentation, the input sizes are not shown and only one example is shown in the figure. However, the real system was trained with data batches containing 64 samples each. Also, the representation was designed to highlight the processing sections by grouping the relevant layers. The architecture used for the Generator is complex for two reasons: first, the generated samples are very large and adding features to the input noise is not straightforward. Secondly, the use of simple upsampling layers has meant that convolutional layers have also had to be incorporated into the network.

The input noise is fed into a dense layer of [128*16*1] neurons. The output of this layer is the input to a batch normalization layer, followed by a leaky ReLU. The batch normalization layers are used to make the network more stable and overall faster. They take the inputs and perform recentering and rescaling. After these initial layers, a reshaping layer is used to reshape the input into the form (16,128,1).

Then, batches of upsampling, 2D convolutions, batch normalization, and leaky ReLU layers are alternated with different kernel sizes. Finally, a Conv2dTranspose layer is used as the last layer for the noise processing section. After this layer, the output values are in the range [-1, 1]. These values are obtained when the function *tanh* is used as the activation for the last Conv2D transpose layer. These values are normalized in the range [0, 255] and the output is

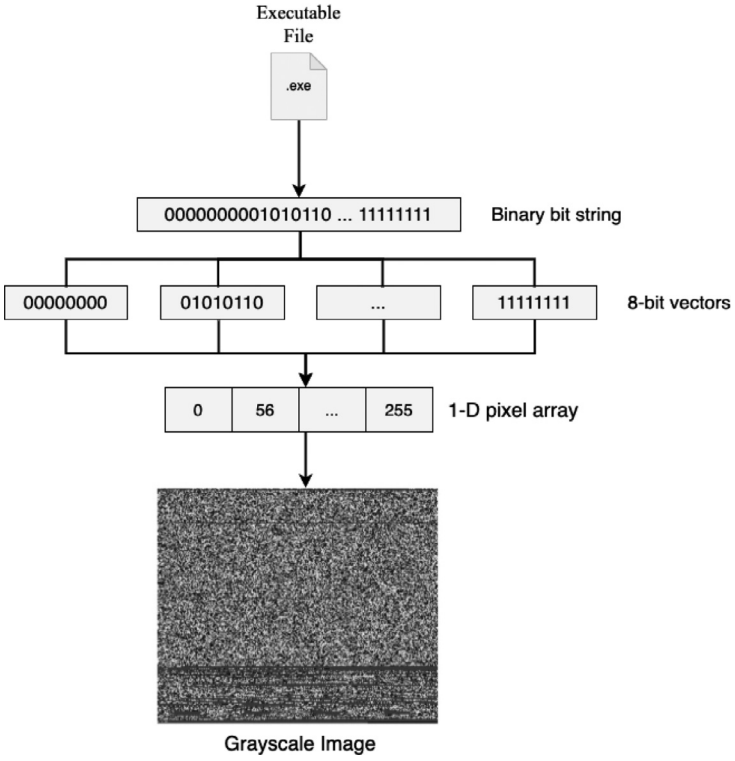

Fig. 9. Input of the Substitute Detector and Generator: from binary data to grayscale image.

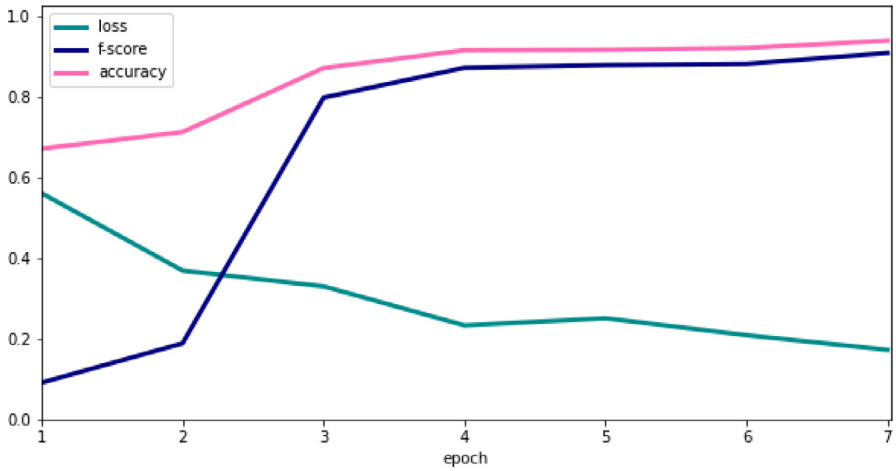

Fig. 10. Results of the Substitute Detector testing.

concatenated with the malware sample, which is the other input to the network. The loss for the Generator is calculated based on the predictions made by the Substitute Detector for new samples. The loss function chosen for the combined model is the binary cross entropy.

Finally, without changing the architecture and input layer, the Generator can be trained to work with samples of similar size that are conveniently filled with random padding to match the input size.

6. Experimental results

In this section, we describe and discuss the experiments carried out to validate our study. The hardware used to perform the experiments consists of a cluster of 7 servers, each with CPU Intel E5-2620v4, 16 cores, 128 GB RAM, and 4 GTX 1080Ti GPUs.

6.1. Training the GAN

The training of the proposed GAN was performed on both a dataset of about 200 benign samples and 1500 malware samples. It was found that training with a larger dataset leads to greater variability in the results obtained, but also requires a longer training time. Considering that the main goal is to generate one malware, using a smaller dataset is more favorable for an attacker, since a larger number of benign and malicious samples would also increase the number of requests to the Blackbox Detector to obtain labels, which could lead to security warnings in the case of real systems. This aspect is discussed in more detail in Section 6.4.

Figure 12 shows some of the generated patterns across different training epochs for a malicious sample from the MalImg dataset. By comparing the first and second subfigures, it is evident the noise introduced at the end of the file (which is missing in the first

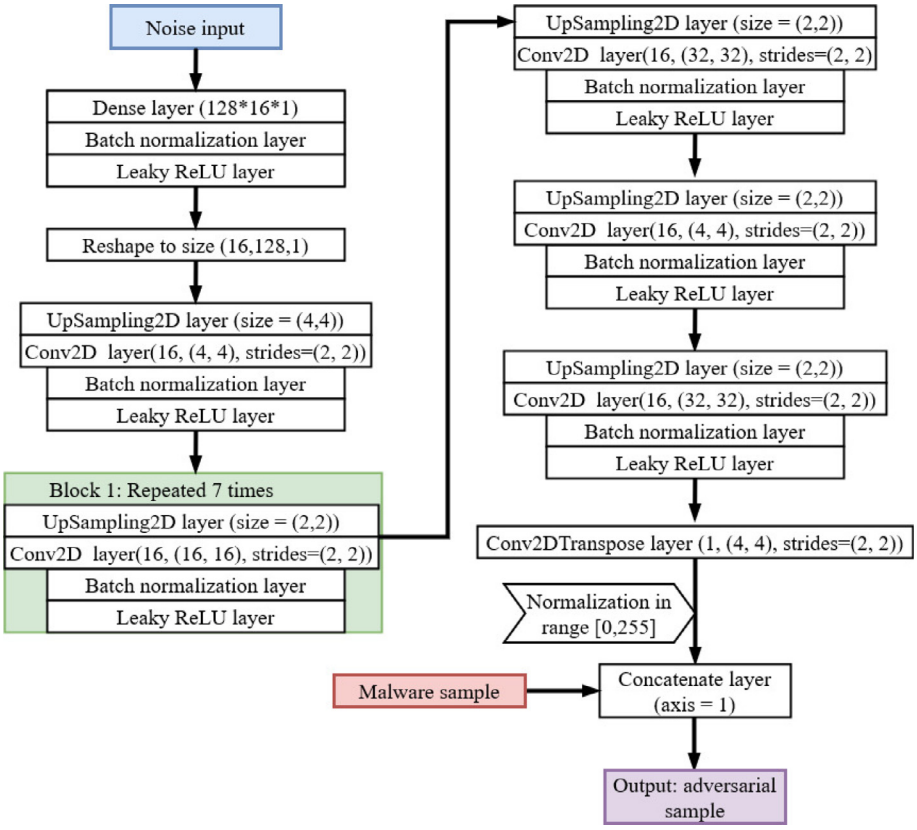

Fig. 11. Generator architecture.

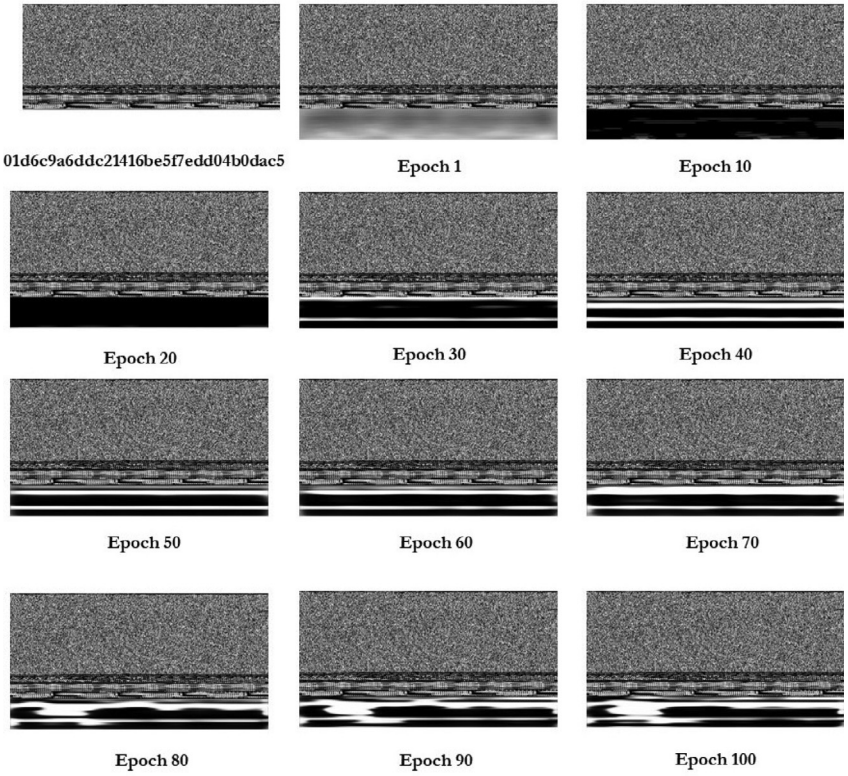

Fig. 12. Generated adversarial samples over training epochs for malware sample '01d6c9a6ddc21416be5f7edd04b0dac5'.

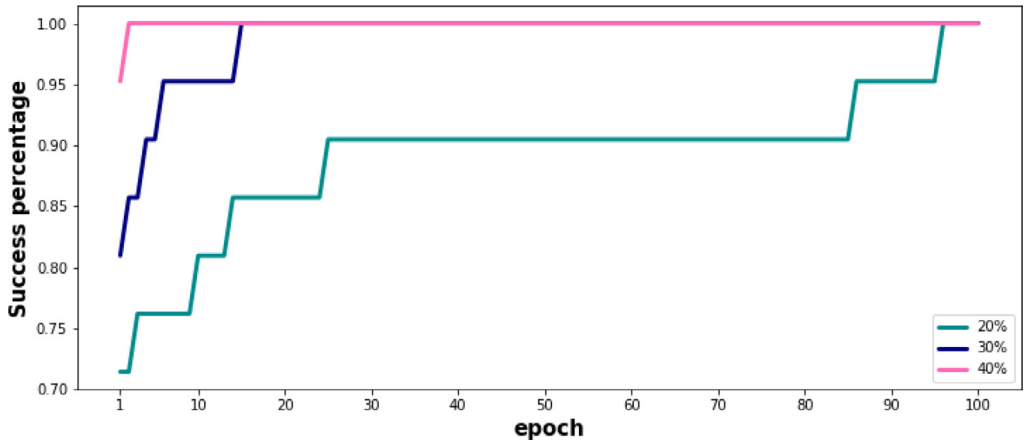

Fig. 13. Success ratio versus epochs for different noise percentages.

Table 1
Tested hyperparameters.

	Hyperparameters	Tested Values		
GAN	img_width	128	256	**512**
	batch_size	32	128	**64**
	epochs	25	50	**100**
Substitute Detector	learning rate	1e-4	1e-3	**2e-3**
Generator	learning rate	1e-4	1e-3	**1e-3**

subfigure and has a size of 30%). Observe that the first part of all subfigures is not modified by the generator, as the binary content of the sample must be preserved in order for the executable to remain as such. The noise is initially random and epoch after epoch produces the pattern that eludes the visualization-based approach.

6.2. Hyperparameters

Training GANs can be challenging, so the choice of hyperparameters was based mainly on empirical assessments. Several experiments were conducted by varying the hyperparameters and, as done by Prajapati and Stamp (2021), we report in Table 1 the most interesting settings. The initial attempts to train the framework were performed with a learning rate of 1e-4 for both the Generator and Substitute Detector. These values resulted in slow convergence and in some cases the GAN did not converge at all. The next step was to increase the learning rate. The third column shows the chosen values. In this case, it was observed that the Generator outperformed the Substitute Detector rather quickly, leading to instability in the training. Finally, the values in the last column were used.

6.3. Validation

To validate our approach, we applied our proposal to the set of malware files not used in the training phase to generate a new variant of the malware that is not detected by the *Pinhero* technique. We measured the ratio of malware files for which at least a variant has been generated versus the number of epochs. We repeated this experiment using three different sizes of *input noise*: 20%, 30%, and 40% of the input file. Specifically, the input noise is the amount of bits that the Generator creates to be added to the file: obviously, the size of the new file is obtained by summing the size of the initial malware and the size of the noise. The results of these experiments are shown in Fig. 13. We observe and highlight that our proposal was able to generate a new malware variant for all of the malware files given in input. As expected, the larger the input noise size, the faster the generation of the malware variant.

Now, we consider the time needed to generate malware variants: Fig. 14 shows the number of hours needed to perform 100 epochs of training for the GAN using different file sizes and input noise percentages. We observe that less than one hour is sufficient for small files (less than 50 kilobytes), whereas 5 h and a half are needed for large files (more than 500 kilobytes) and 40% of input noise. The measured times are feasible and acceptable for an attack in the considered context.

6.4. Discussion

The idea of a machine learning model being retrained to protect itself from adversarial attacks is not uncommon. Although the possibility of introducing such a mechanism is not mentioned by Pinhero et al. (2021), this aspect can be evaluated in our study. The Blackbox Detector could be retrained to improve its ability to detect malware created by the Generator based on additional adversarial samples collected. This capability exists only after an attack has been detected and the adversarial samples have been identified. The detected adversarial samples can then be used for retraining. At this point, however, the attackers could also perform the Generator and Substitute Detector training against the newly trained Blackbox Detector, as is common with GANs.

Specifically, there are two main reasons that have prevented us from pursuing this approach:

1. retraining the Blackbox Detector (i.e., the model implementing the technique presented by Pinhero et al., 2021) and consequently the Substitute Detector would change the associated model. On the one hand, this means that both detectors would improve the discrimination of adversarial samples. On the other hand, this change could also lead to worse detection of other malware families. In practice, this would not allow a fair comparison with the state of the art;
2. retraining the discriminatory part of the system as a defensive countermeasure does not necessarily mean that the attacker will eventually be defeated. The attacker could easily continue to train the Generator to produce new variants that are not detected. In this approach, improvements on one side lead to improvements on the other.

Moreover, adversarial training is not always easy to implement. For example, Qiu et al. (2019) addresses the fact that it is unrealistic to include all unknown attack patterns in adversarial training, which leads to a limitation of adversarial training. Furthermore, it was highlighted by Ding et al. (2019) that the adversarial training can be sensitive to the distribution of the training data

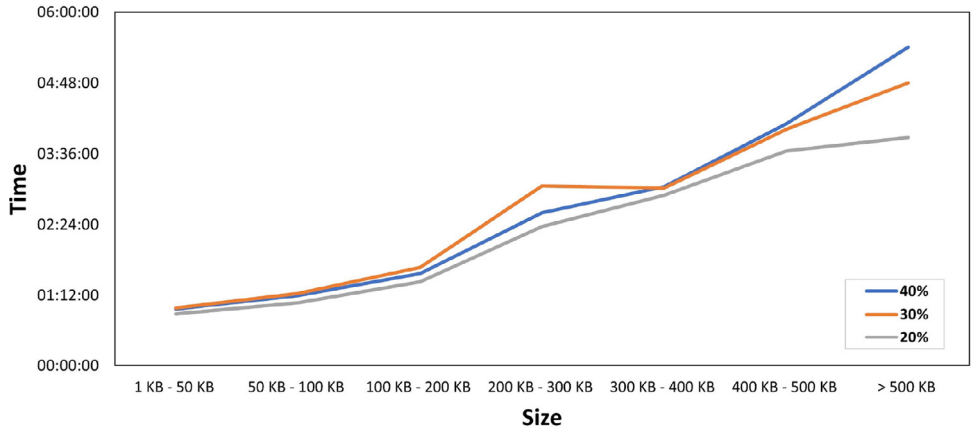

Fig. 14. Running time for different file sizes and input noise percentages.

and that this type of training suffers from generalization problems (Song et al., 2018).

It is important to note that the chosen training strategy is different from a conventional GAN. The first difference is that the Substitute Detector receives malware and benign samples with the labels predicted by the Blackbox Detector. The second difference is that the combined model (Generator plus Substitute Detector) is trained on malicious samples with ground truth labels corresponding to the *benign* class to learn how to defeat the Blackbox Detector. This difference in training strategy causes the Generator to converge to a solution because, once the Generator successfully generates a malicious pattern that bypasses detection, the predicted labels passed from the Blackbox Detector to the Substitute Detector are the same ones that would be used to identify benign samples. Therefore, the Substitute Detector loses the ability to detect malicious patterns. It can be inferred that unless the Blackbox Detector learns to discriminate malicious samples (e.g., by re-training), training will converge to a solution. Note that the loss of the Generator decreases over epochs and its accuracy increases, resulting in most samples being mislabeled by the Substitute Detector and eventually by the Blackbox Detector. This means that the attack can be considered successful even if only a single file escapes detection.

A major advantage for attackers using this framework is that they have different batches of malware patterns available when they store the generated patterns across different epochs and consider those that have been mislabeled by the Blackbox Detector. In this way, the same malware with its many variants can be used for multiple attacks to avoid detection. As our experiments show, our proposal was able to generate at least one new malware variant for all malware files included in our publicly available dataset.

However, the limitation of our proposal is the training time with a larger dataset and larger files. Our experiments have shown that with our settings, one hour is needed to generate an undetected malicious variant for a small file (less than 50 kilobytes), while five and a half hours are needed for a large file (more than 500 kilobytes).

Finally, in a real scenario, the Generator tries to cheat the Blackbox Detector on every attempt, but the Blackbox Detector should be queried to make predictions. It can be assumed that a detector that provides a public API has a request control system. In this case, the attacker should also avoid being identified as someone who floods requests. There are several ways to accomplish this. It might be possible to time the request so that it does not occur too frequently. This approach has the disadvantage of being time consuming. But again, it all depends on the scenario and how urgent the attackers need to get into the system protected by the Blackbox Detector. Another option would be to find a way to mask the attacking machine with different hosts.

7. Conclusion

Our research aimed to prove that machine learning based malware detection systems relying on a visualization approach are still not suitable as a standalone malware detection solution. This point was proven by implementing a GAN-based malware obfuscation system. We implemented a framework that has proven successful in creating malicious patterns that can evade detection. Moreover, the GAN implemented in this system is not a conventional GAN, as the Substitute Detector is controlled by the Blackbox Detector. The contribution of this work compared to previous GAN-based attack systems with similar architecture is that the proposed model is able to generate samples that can be converted back to the original executables yet preserving the expected file execution. Moreover, this study provides the dataset used in the experiments to allow reproducibility and to be used in other studies for comparisons.

Future developments of the system could include solutions to improve training stability. A common proposal in the literature is to use the Wasserstein loss function (Arjovsky et al., 2017) to improve the stability of the training to avoid common error modes. In addition, training the Blackbox Detector could include using different image conversion techniques and evaluating the impact of the different techniques.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Lara Saidia Fascí: Conceptualization, Methodology, Software, Investigation, Writing – original draft. **Marco Fisichella:** Supervision, Conceptualization, Validation, Resources, Writing – original draft, Writing – review & editing, Visualization. **Gianluca Lax:** Supervision, Conceptualization, Validation, Resources, Writing – original draft, Writing – review & editing, Visualization. **Chenyi Qian:** Conceptualization, Methodology, Software, Investigation.

Data availability

Data will be made available on request.

Acknowledgement

This work was supported in part by the research project "SoBigData++" funded by the European Commission under the Horizon 2020 program with grant agreement number 871042.

References

Aboaoja, F.A., Zainal, A., Ghaleb, F.A., Al-rimy, B.A.S., Eisa, T.A.E., Elnour, A.A.H., 2022. Malware detection issues, challenges, and future directions: a survey. Appl. Sci. 12 (17). doi:10.3390/app12178482.

Aboaoja, F.A., Zainal, A., Ghaleb, F.A., Al-rimy, B.A.S., Eisa, T.A.E., Elnour, A.A.H., 2022. Malware detection issues, challenges, and future directions: a survey. Appl. Sci. 12 (17), 8482.

Arjovsky, M., Chintala, S., Bottou, L., 2017. Wasserstein generative adversarial networks. In: International Conference on Machine Learning. PMLR, pp. 214–223.

Bijitha, C., Nath, H.V., 2022. On the effectiveness of image processing based malware detection techniques. Cybern. Syst. 53 (7), 615–640.

Cai, Z., Xiong, Z., Xu, H., Wang, P., Li, W., Pan, Y., 2021. Generative adversarial networks: a survey towards private and secure applications. arXiv preprint arXiv:2106.03785.

Claroty, 2021. Security researchers reveal staggering magnitude of ICS vulnerabilities in 2021 as cyber attacks on critical infrastructure increase. https://www.prnewswire.com/news-releases/.

Code Repository, 2022. Project code. https://github.com/Lara-F/Disarming-Visualization-based-Approaches-in-Malware-Detection-Systems.

Damodaran, A., Troia, F.D., Visaggio, C.A., Austin, T.H., Stamp, M., 2017. A comparison of static, dynamic, and hybrid analysis for malware detection. J. Comput. Virol. Hacking Tech. 13 (1), 1–12.

Ding, G.W., Lui, K.Y.C., Jin, X., Wang, L., Huang, R., 2019. On the sensitivity of adversarial robustness to input data distributions. ICLR (Poster).

DriverPack Solution, 2022. Driverpack. https://driverpack.io/en.

Dube, T., Raines, R., Peterson, G., Bauer, K., Grimaila, M., Rogers, S., 2012. Malware target recognition via static heuristics. Comput. Secur. 31 (1), 137–147. doi:10.1016/j.cose.2011.09.002.

Filehippo, 2022. Software that matters. https://filehippo.com/.

García, D.E., DeCastro-García, N., 2021. Optimal feature configuration for dynamic malware detection. Comput. Secur. 105, 102250. doi:10.1016/j.cose.2021.102250.

Geremias, J., Viegas, E.K., Santin, A.O., Britto, A., Horchulhack, P., 2022. Towards multi-view android malware detection through image-based deep learning. In: 2022 International Wireless Communications and Mobile Computing (IWCMC). IEEE, pp. 572–577.

Gibert, D., Mateu, C., Planes, J., Marques-Silva, J., 2021. Auditing static machine learning anti-malware tools against metamorphic attacks. Comput. Secur. 102, 102159. doi:10.1016/j.cose.2020.102159.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. Adv. Neural Inf. Process. Syst. 27.

Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., Mao, L., 2019. MalDAE: detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. Comput. Secur. 83, 208–233.

Hu, W., Tan, Y., 2017. Generating adversarial malware examples for black-box attacks based on GAN. arXiv preprint arXiv:1702.05983.

Kawai, M., Ota, K., Dong, M., 2019. Improved MalGAN: Avoiding malware detector by leaning cleanware features. In: 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC). IEEE, pp. 040–045.

Khodamoradi, P., Fazlali, M., Mardukhi, F., Nosrati, M., 2015. Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In: 2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS), pp. 1–6. doi:10.1109/CADS.2015.7377792.

Lin, Z., Shi, Y., Xue, Z., 2018. IDSGAN: generative adversarial networks for attack generation against intrusion detection. arXiv preprint arXiv:1809.02077.

Major Geeks, 2022. It's all geek to me. https://www.majorgeeks.com/.

Mercaldo, F., Santone, A., 2020. Deep learning for image-based mobile malware detection. J. Comput. Virol. Hacking Tech. 16 (2), 157–171.

Mixon, E., 2021. Top 10 ransomware attacks of 2021 (so far). https://www.blumira.com/ransomware-attacks-2021/.

N-able, 2021. Malware analysis steps and techniques - n-able. https://www.n-able.com/blog/malware-analysis-steps:text=What%20is%20static%20malware%20analysis,without%20even%20viewing%20the%20code.

Nataraj L., Karthikeyan S., Jacob G., Manjunath, B. S., 2020. MalImg dataset. https://www.dropbox.com/s/ep8qjakfwh1rzk4/malimg_dataset.zip?dl=0.

Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S., 2011. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security, pp. 1–7.

Pinhero, A., Anupama, M.L., Vinod, P., Visaggio, C.A., Aneesh, N., Abhijith, S., Anantha Krishnan, S., 2021. Malware detection employed by visualization and deep neural network. Comput. Secur. 105, 1–30.

Portable Freeware, 2022. The portable freeware collection. https://www.portablefreeware.com/.

Prajapati, P., Stamp, M., 2021. An empirical analysis of image-based learning techniques for malware classification. In: Malware Analysis Using Artificial Intelligence and Deep Learning. Springer, pp. 411–435.

Qiu, S., Liu, Q., Zhou, S., Wu, C., 2019. Review of artificial intelligence adversarial attack and defense technologies. Appl. Sci. 9 (5), 909.

Repository, 2022. Goodware dataset. https://mega.nz/file/tHYzDSIL#k2YjAWHKn-TQ0qJE6_iTuDRWK0BzbVdQT0_4LGSX4uTY (password: Ben1gN@D$!?).

Softonic, 2022. App news and reviews, best software downloads and discovery. https://en.softonic.com/.

Song, C., He, K., Wang, L., Hopcroft, J. E., 2018. Improving the generalization of adversarial training with domain adaptation. arXiv preprint arXiv:1810.00740.

Tahir, R., 2018. A study on malware and malware detection techniques. Int. J. Educ. Manage.Eng. 8 (2), 20.

Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., Zheng, Q., 2020. IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture. Comput. Netw. 171, 107138.

Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q., 2020. Image-based malware classification using ensemble of CNN architectures (IMCEC). Comput. Secur. 92, 101748.

Wael, D., Sayed, S.G., AbdelBaki, N., 2018. Enhanced approach to detect malicious vbscript files based on data mining techniques. Procedia Comput. Sci. 141, 552–558. doi:10.1016/j.procs.2018.10.127. The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018)/The 8th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2018)/Affiliated Workshops, https://www.sciencedirect.com/science/article/pii/S1877050918317757

Wael, D., Shosha, A., Sayed, S.G., 2017. Malicious vbscript detection algorithm based on data-mining techniques. In: 2017 Intl Conf on Advanced Control Circuits Systems (ACCS) Systems & 2017 Intl Conf on New Paradigms in Electronics & Information Technology (PEIT), pp. 112–116. doi:10.1109/ACCS-PEIT.2017.8303028.

Washington, C., Yarbrough, P., Parker, S., Islam, R., Patamsetti, V.V., Abiona, O., 2022. Information assurance technique for mitigation of data breaches in the human service sector. Int. J. Commun. Netw. Syst. Sci. 15 (2), 15–30.

Wikipedia Foundation, 2022. Binary classification — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Binary_classification.

Yadav, P., Menon, N., Ravi, V., Vishvanathan, S., Pham, T.D., 2022. A two-stage deep learning framework for image-based android malware detection and variant classification. Comput. Intell..

Lara Saidia Fascí got her Master Degree in Computer Science at Universitá Mediterranea di Reggio Calabria (Italy) with the thesis on "Implementation of a GAN-based malware generation framework to attack machine-learning-based detection systems".

Dr. Marco Fisichella has more than 14 years of experience in Artificial Intelligence, both in industry and academia. From April 2021, he led at the L3S Research Centre of Leibniz University in Hannover, Germany, the research group with a focus on artificial intelligence and intelligent systems. Within the scope of the activities at L3S, new methods of artificial intelligence are to be explored both in the field of basic research and in application-oriented research, especially for the application fields of mobility, intelligent production and personalised medicine. From 2015 to March 2021, he was Head of Data at Otto Group, where he led the department responsible for implementing algorithms against online fraud. Marco led various AI and data science projects for corporate clients (e.g. Otto, Vodafone, DriveNow, Bonprix, etc.). His research interests include artificial intelligence, machine learning, data mining, information retrieval, generative models, event detection, clustering methods based on statistical approaches, near duplicate detection.

Prof. Dr. Gianluca Lax (Member, IEEE) received the PhD and Habilitation degrees in computer science from the University of Calabria, Rende, Italy, in 2005 and 2018, respectively. He is currently an Associate Professor of computer science with the Mediterranea University of Reggio Calabria, Reggio Calabria, Italy. He has authored more than 120 articles published in leading international journals and conference proceedings. His research interests include information security and social network analysis.

Chenyi Qian got her Master Degree in Computer Science at Leibniz university of Hannover (Germany) with the thesis on "Adversarial Attacks Based on GAN to Avoid Malware Detection."

Update

Computers & Security
Volume 144, Issue , September 2024, Page

DOI: https://doi.org/10.1016/j.cose.2024.103934

Computers & Security 144 (2024) 103934

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Corrigendum to "Disarming visualization-based approaches in malware detection systems" [Computers & Security Volume 126, March 2023, 103062]

Lara Saidia Fascí [a], Marco Fisichella [b,*], Gianluca Lax [a], Chenyi Qian [b]

[a] DIIES Dept., University of Reggio Calabria, Via Graziella, Loc. Feo di Vito, Reggio Calabria, 89122, Italy
[b] L3S Research Center of Leibniz University Hannover, Appelstrasse 9a, Hannover,30167, Germany

The authors noticed that the link where they stored the open distributed dataset was not permanent and wanted to give other researchers the possibilities to research continuously on the dataset used.

The authors propose to add the following current link for the permanent access of their dataset:

https://data.mendeley.com/datasets/cv3v9szdn7/3.

The authors would like to apologise for any inconvenience caused.

DOI of original article: https://doi.org/10.1016/j.cose.2022.103062.

* Corresponding author.
E-mail address: mfisichella@L3S.de (M. Fisichella).

https://doi.org/10.1016/j.cose.2024.103934

Available online 7 June 2024