I. J. Computer Network and Information Security, **2025, 1, 1-16**
Published Online on February 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2025.01.01

Enhancing Adversarial Examples for Evading Malware Detection Systems: A Memetic Algorithm Approach

Khadoudja Ghanem*
University Constantine 2, Abdelhamid Mehri, Faculty of New Technologies of Information and Communication, Department of Computer Science, 25000, Algéria
E-mail: khadoudja.ghanem@univ-constantine2.dz
ORCID iD: https://orcid.org/0000-0003-4401-4554
*Corresponding Author

Ziad Kherbache
University Constantine 2, Abdelhamid Mehri, Faculty of New Technologies of Information and Communication, Department of Computer Science, 25000, Algéria
E-mail: ziad.kherbache@univ-constantine2.dz

Omar Ourdighi
University Constantine 2, Abdelhamid Mehri, Faculty of New Technologies of Information and Communication, Department of Computer Science, 25000, Algéria
E-mail: omar.ourdighi@univ-constantine2.dz

Received: 26 August 2023; Revised: 25 October 2023; Accepted: 27 December 2023; Published: 08 February 2025

Abstract: Malware detection using Machine Learning techniques has gained popularity due to their high accuracy. However, ML models are susceptible to Adversarial Examples, specifically crafted samples intended to deceive the detectors. This paper presents a novel method for generating evasive AEs by augmenting existing malware with a new section at the end of the PE file, populated with binary data using memetic algorithms. Our method hybridizes global search and local search techniques to achieve optimized results. The Malconv Model, a well-known state-of-the-art deep learning model designed explicitly for detecting malicious PE files, was used to assess the evasion rates. Out of 100 tested samples, 98 successfully evaded the MalConv model. Additionally, we investigated the simultaneous evasion of multiple detectors, observing evasion rates of 35% and 44% against KNN and Decision Tree machine learning detectors, respectively. Furthermore, evasion rates of 26% and 10% were achieved against Kaspersky and ESET commercial detectors. In order to prove the efficiency of our memetic algorithm in generating evasive adversarial examples, we compared it to the most used evolutionary-based attack: the genetic algorithm. Our method demonstrated significantly superior performance while utilizing fewer generations and a smaller population size.

Index Terms: Malware Detection, Adversarial Examples, Memetic Algorithms, Genetic Algorithm, Malconv Model, Machine Learning.

1. Introduction

In the context of malware detection, adversarial examples (AEs) can be used to create malicious code that evades detection engines, thus posing a significant threat to cyber security. AEs are specifically crafted inputs designed to fool machine learning (ML) models by exploiting vulnerabilities in their decision-making process[1].

While ML-based classifiers have shown improved efficacy in detecting malware, adversaries have proposed countermeasures to bypass detection, necessitating the development of effective countermeasures in response[2]. Generating AEs through adversarial attacks is considered a powerful technique for evading the detection of carefully perturbed malware samples. Malware can be modified by different types of manipulation functions, the existing literature indicates that few researchers have analyzed the specific features that adversaries can modify in malwares to generate adversarial samples [3], this highlights a research gap in understanding the feature frequency and modification

techniques that attackers can employ to create adversarial malwares. Furthermore, the generation of adversarial malwares that preserve functionality and the original maliciousness semantics of the malwares and can evade various state-of-the-art detection techniques is another challenge that needs to be addressed [4].

Few surveys focusing on the adversarial attacks in the context of malware [5,6] are presented in the literature. In [7] authors provide a comprehensive review on the state of-the-art research efforts of adversarial attacks against Windows PE malware detection as well as corresponding defenses to increase the robustness of existing PE malware detection solutions.

The Portable Executable (PE) file format is used by Microsoft Windows operating systems for executables, object code, DLLs (dynamic-link libraries), FON (font) files, and others. Malware developers often use the PE format to distribute their malicious software, while security researchers use the same format to analyze and detect it [8,9]. There are many possible and simple solutions to perform manipulations on PE files [10], but, there are few special cases in which it is possible to directly perform changes to the executable without compromising its functionality.

The focus of this study will be on generating AEs with the aim of evading detection in the context of malware analysis. When we have limited knowledge of the targeted model, the use of genetic algorithms as an optimizer in adversarial attacks has shown promising results[11]. However, premature convergence is an inherent characteristic of such classical genetic algorithms that makes them incapable of searching numerous solutions of the problem domain. A memetic algorithm is an extension of the traditional genetic algorithm. It uses a local search technique to reduce the likelihood of the premature convergence.

The novelty of the proposed solution in this paper is that it is the first work that hybridize global search (GA)with local search(MA:Hill Climbing) to generate successful AE. Indeed, all proposed methods in the category Evolutionary-based attack use only genetic algorithm, a global search method. The proposed solution in this paper leverage memetic algorithms, and the MalConv model to explore the effectiveness of these techniques in creating adversarial malware samples.

The main contributions in the current paper are:

- Exploring the potential of memetic algorithms in modifying malware to evade detection and generate successful evasive AE.
- Evaluating the effectiveness of the MalConv model, a state-of-the-art architecture, in detecting and classifying adversarial malware samples.
- Comparing the performance of memetic algorithms and genetic algorithms in generating adversarial examples to evade ML detectors (KNN, DT), and commercial Anti virus engines (Kaspersky, ESET).
- Investigating the impact of adversarial learning on enhancing the detection mechanisms employed by security systems within the context of malware analysis.

The paper is structured as follows: Section 2 presents the related works and the basics of used materials. Section 3 describes the mechanisms of the proposed method for generating executable adversarial malware samples. The results of the conducted experiments are reported in Section 4. Finally, the conclusion and future works are provided in section 5.

2. Related Works

Numerous research papers have addressed the generation of AEs to bypass malware detection systems. These studies delve into the inherent vulnerabilities of malicious code detection engines and propose improved methods using artificial intelligence (AI) to generate adversarial samples that can evade detection [8,12-16]. These studies highlight the significance of the problem and the need to develop robust defenses against adversarial attacks on malware detection models.

Many Attack strategies are proposed in the literature[7], gradient-based attack, Reinforcement Learning based attack, randomized attack, evolutionary-based attack and Generative adversarial attack are some examples of these strategies. In this study, we aim to find an optimized attack, thus, we focus on recent evolutionary-based attacks.

In [17], the authors propose AIMED a method to minimize malware scores. The approach consists of eight main components, including the Manipulation Box, which injects byte-level perturbation such as Padding, Section Injection, API Injection, and Header Fields into the malware sample to create the population of a genetic algorithm based attack. The approach is tested against four malware classifiers, with Kaspersky chosen for its good performance, and the impact of evasive mutations among different black box classifiers is also compared. The approach has a fast convergence and the cross evasion rate is high(82%). The problem with this method is that 76% of input files are unmodifiable,thus it will be interresting “to gain a better understanding about how to convert unmodifiable files into modifiable ones”[17].

Similarly, in [18] authors propose MDEA to retrain the MalConv model. To generate the adversarial malware samples, MDEA adjusts 10 kinds of format -preserving manipulations as the action space and employs a genetic algorithm to optimize different action sequences by selecting manipulations from the action space until the generated adversarial malware bypasses the target malware detectors. MDEA limits each manipulation with a parameter set to

make the adversarially trained models converge within an acceptable time. However, the generated adversarial malware samples by MDEA are not tested for functionality.

In [19], authors introduced GAMMA a black-box attack framework, where, the generation approaches of adversarial malware are limited to two types of functionality-preserving manipulations: section injection and padding. Specifically, benign contents are extracted from the goodware as adversarial payloads to inject either into some newly created sections (section injection) or at the end of the file (padding). Gamma optimizes the probability of evading detection and penalizes the size of the injected adversarial payload as a regularization term. It employs genetic algorithm to bypass the malware detector with few queries as well as small adversarial payloads.

In [20], authors optimize code caves in malware binaries to evade ML detectors. By dynamically introducing unused blocks in malware binaries while preserving their original functionality, the authors generate AEs using artificial neural networks. Genetic algorithms are employed to determine the content to place in code caves for achieving misclassification. Evaluation of the proposed model is conducted in a black-box setting using MalConv architecture, it achieves a result of 97.99% successful evasion rate from 2k tested malware samples. Additionally, the transferability of the proposal is successfully tested on commercial Anti virus engines available at VirusTotal, demonstrating a reduction in the detection rate for the crafted AEs.

The authors in [21] have explored the susceptibility of deep network-based malware detection methods to evasion attacks, they propose a gradient-based attack that can evade Malconv architecture designed for this purpose by making small changes to the input data. They have demonstrated that their adversarial malware binaries can evade the targeted network with a 60% success rate, even when less than 1% of their bytes are modified, while maintaining their intrusive functionality. They used 200 malware samples, were the smaller input file size is 106k., but they found that appending bytes to the end of the file reduces the effectiveness of the gradient-based approach.

Genetic programming has also been introduced in the mobile domain. In [22], authors present a method that evolves automatically variants of malwares from the ones in the wild by using genetic programming.

In [23] the authors propose a testing framework for learning-based Android malware detection systems for IoT Devices. They introduce genetic algorithms to generate effective adversarial samples and can perform black-box testing on the system. And in [24], the authors propose GenDroid, a framework for crafting Android AEsin black-box scenarios. They adopt an evolutionary strategy and introduce Gaussian process regression to guide the evolution, which substantially improves the attack efficiency. They demonstrate their attack on two state-of-the-art Android malware detection schemes, Drebin (Arp et al., 2014) and MaMaDroid (Mariconti et al., 2016) . GenDroid has a higher misclassification rate compared to some state of the art attack.

A synthesis of the presented works reveals that the most attacked model is the Malconv model, a state-of-the-art deep learning model specifically designed for detecting malicious PE files.

The Malconv architecture Fig.1 is a meticulously designed deep learning model specifically designed for detecting malicious PE files[25].

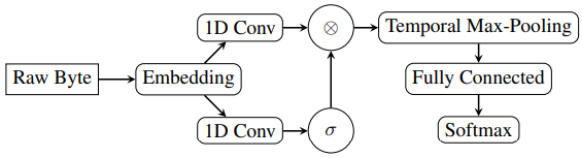

Fig.1. Malconv model architecture[25]

The model starts with an input layer that takes the raw byte sequences of PE files as binary content. It then includes an embedding layer to transform the byte sequences into a continuous vector representation, capturing essential semantic information. Convolutional layers are employed to extract meaningful features from the embedded byte sequences. These layers utilize filters that slide over the input data, capturing local dependencies and patterns. Two convolutional layers with 128 filters each are utilized, learning hierarchical representations of the data. In addition to the convolutional layers, the Malconv architecture incorporates an attention layer, assigning importance weights to different segments of the byte sequences. This enables the model to focus on relevant parts of the input data. The outputs of the convolutional and attention layers are combined using an element-wise multiplication operation called gating, selectively emphasizing informative patterns. A global max pooling layer is applied to aggregate relevant features across the entire sequence, capturing the essential characteristics of the PE files while reducing dimensionality. The pooled features are then passed through fully connected layers, performing nonlinear transformations and extracting high-level representations. A dense layer with 128 units and a rectified linear unit (ReLU) activation function is utilized. The final layer of the Malconv model is a dense layer with a sigmoid activation function, producing a single output representing the probability of the input file being classified as malicious. During training, the model's weights are updated using the stochastic gradient descent (SGD) optimizer with specific hyperparameter. The model is trained using a binary cross-entropy loss function, which measures the discrepancy between predicted probabilities and true labels.

In another hand the synthesis of the presented evolutionary based attacks reveals that most of these methods employ genetic algorithms. However, the main problem of genetic algorithm when generating adversarial malwares, is that it fail to generate an adversarial example from most of the original malwares even over a high number of

generations, these original malwares are judged unmodifiable. To address this problem, we investigate the harness power of Memetic algorithms (MAs), a hybrid optimization technique that combines global search of evolutionary algorithms with local search methods[26].

MAs embody the principles of natural selection and evolution, mimicking the processes observed in biological systems[27]. By employing genetic algorithms, which serve as the global search component, memetic algorithms can explore a wide range of solutions within the search space. Genetic algorithms leverage evolutionary operators such as selection, crossover, and mutation to generate diverse and promising candidate solutions. However, genetic algorithms alone may struggle with fine-tuning and can be prone to premature convergence where the algorithm gets trapped in a local minimum or fails to explore the entire search space thoroughly. In order to reduce the likelihood of premature convergence, different local search methods, such as simulated annealing, Tabu search or hill climbing can be used. In this paper, we propose a Memetic algorithm that hybridize genetic algorithm with a Hill Climbing Algorithm in order to generate for each malware a successful adversarial example.

3. Proposed Approach

The proposed approach to modify the PE file with the aim of generating AEs is depicted in Fig. 2, it provides a workflow chart of the main steps of the developed Memetic algorithm, while Algorithms 1,2,3,4 below the workflow chart, outline the detailed process and operations involved in our proposed approach.

Hill climbing is incorporated within our proposed Memetic algorithm framework as a local search method to enhance its ability to refine and optimize the generated adversarial examples.

Malconv model is used to evaluate the effectiveness of the generated AE, thus, before generating any AE from PE malwares, we trained and evaluated the Malconv model to produce a model that will be used in the prediction of any input file as a malware or a goodware. The objective function of the MalConv model is the binary cross-entropy loss eq.(1):

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

Where:

- N is the number of data files.
- yi is the true label for the ith file, where (yi = 1) for malware and (yi = 0) for goodware.
- (pi) is the predicted probability that the i th file is malware according to the MalConv model.

The objective is to minimize this loss function during the training process. This means that the model aims to make predictions (pi) that are as close as possible to the true labels (yi). The loss quantifies the dissimilarity between the predicted probabilities and the true labels. The MalConv model is trained using this loss function and backpropagation to update its parameters (the weights and biases) to improve its classification performance and enhance the detection of malicious files. The model is trained on over 1000 known labeled malware samples, and over 1000 known labeled goodware samples.

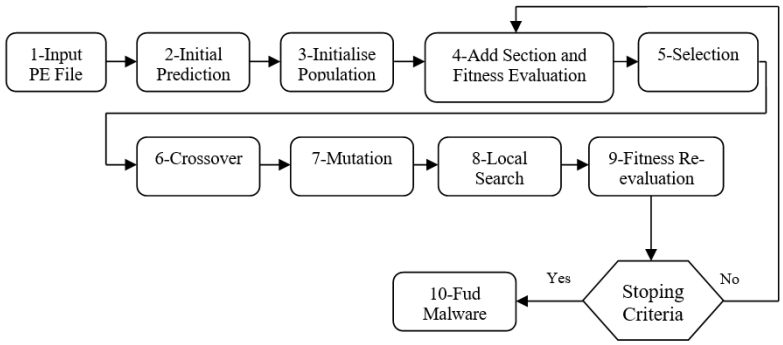

Fig.2. Workflow diagram

Algorithm 01: Pseudo code of Generating AE with MA

Initialisation:
m: input PE malware;
ngen : number of generations(25);gen=0;
N: population size(10);
Undetected='False';
best_individual,best_pred=None;
i*: best_individual;

```
m*:Generated AE;
Begin
Prediction1=predict(input PE malware); //with malconv model;
If (Prediction1<0.5) Then Print('PE not detected as malware') Else:
Create initial binary population (i1..i10 randomly) ;
While (gen<=25) AND (Not(Undetected)) Do
    Add-sections to PE malware (i1..i10) (Algorithm 02);
    Evaluate Fitnesses of all new PE files  with added sections (Algorithm 04);
    Select two individuals with tournament operator(tournsize=3);
    Apply Crossover;
    Apply Mutation;
    Evaluate Fitnesses of (offsprings);
    Apply Local search:Hill climbing on offsprings (subset=50) (Algorithm03);
    Keep best individuals from new pop;
    gen=gen+1;
Endwhile
If Undetected Then : logging.info('PE Not Detected as malware!') Else:  Restore best_individual(i*);
m*=Add-section to input PE malware m (i*);
End.
```

Algorithm 02: Pseudo code of Add-section to PE malware (individual)

```
Parse the binary from the input PE malware;
Create a new section with a random name;
Set the contenent ( individual) to the new section;
Set the virtual adress of the new section to the end of the existing sections of the input malware;
Add the new section to the binary;
Write the modified binary to the output file (the new PE malware =AE);
Return the output file.
```

Algorithm 03: Pseudo code of Hill climbing(Individual,50)

```
Evaluate Fitness of the individual :current fitness;
Repeat from 1 to 50 :
  Generate neighbor: Generate neighboring solution by making modification (1/0) to the individual ;
  Select the best neighbor: Evaluate Fitness for the modified individual and select the one with the highest value:
 If new fitness > current fitness Then :
 Move to the best neighbor and update the current solution with the new solution:
current fitness= new fitness; Else cancel individual modification.
Go back to step 3 and continue the process;
Return individual (the best one).
```

Algorithm 04: Pseudo code of Evaluate Fitness (individual)

```
a=Add-section to PE malware (individual);
pred=predict(a) with Malconv model;
If (pred <= best_pred) Then: update and save the best_individual and best_pred;
   If (best_pred < 0.5) Then : undetected =True.
End ;
Fitness = 1-pred;
Return Fitness.
```

- Parsing the PE File

In this step, the PE file is read and parsed into our program. The input and output directories are defined, and a list of PE files in the input directory is obtained. Each PE file will go through the subsequent steps of the adversarial example generation process.

- Initial Prediction using the Malconv Model

The Malconv model is utilized to make an initial prediction on the original PE file. This prediction serves as a baseline for evaluating the effectiveness of the generated adversarial examples. The prediction result is logged for reference.

- Initializing the Population

A population for potential AEs is created. The population size is set to 10, and the length of each individual in the population is defined as 2056, this length represents the virtual size of the PE file.

- Adding a New Section and Evaluating Fitness

For each individual in the initial population, a new section is added to the end of the PE file (Algorithm 02). The binary data, is represented by 0s and 1s, the strategic placement of binary data within the added section plays a vital role in our approach. There are many possible and simple solutions to perform manipulations on PE files but, there are few special cases in which it is possible to directly perform changes to the executable without compromising its functionality. Anderson et al. [10] are the first researchers who study how to automatically manipulate the original PE malware such that the modified PE malware are no longer detected as malicious by Black-box Adversarial Attacks while do not break the format and functionality. They demonstrate that one of the possible manipulations that preserve functionality is to add binaries at the end of the PE file. For the sake of simplicity, in this paper we only refer to byte appending after the end of the file as modification strategy.

After adding this section, the fitness of the modified PE file is evaluated using the Malconv model (Algorithm04). The fitness value represents the likelihood of the malware being detected. The fitness is to minimize the probability of a modified PE binary being classified as malware by the Malconv model, as (Fitness = 1-probability), thus, higher fitness values indicate a lower chance of evading detection.

- Selection Phase

In the selection phase, two different selection operators: elitism and tournaments were explored to select candidates solutions, as a result, the tournament operator has been adopted. With this style selection mechanism, the population is divided into groups of three (03) individuals when the population size is equal to 10 and into groups of six (06) individuals when the population size is equal to 50, and the best-performing individual from each group is selected for the next generation. This selection process favors individuals with lower fitness values.

- Crossover Phase

The crossover operation is performed on the selected individuals. Crossover introduces genetic diversity by exchanging genetic information between parent individuals to produce offspring, in our study, the probability of the crossover operation is set to 50% to favoritize the diversity because the population size is only 10 so it do not compromise the speed up process.

- Mutation Phase

Mutation helps to introduce new genetic material and potentially discover more effective adversarial examples. A mutation operation is applied to the offspring, each index in the individual has a 20% probability of flipping its value (0 to 1 or 1 to 0). The probability rate of mutation has been chosen experimentally over different values to further enhance diversity and explore different variations.

- Local Search using Hill Climbing

In this step, a local search technique (Hill climbing) is applied to a subset of the individual(Algorithm 3). The size of the chromosome is 2056, but only a subset of 50 indices is selected for the hill climbing operation.

Hill climbing algorithm is a well-known local search algorithm, it complements the exploration capabilities of genetic algorithms in memetic algorithms[27]. It focuses on making incremental improvements by iteratively moving towards the best neighboring solution. Neighboring solution is generated by making modification (1 to 0 or 0 to 1) to an index of the individual , then, the fitness of the current solution is evaluated and updated until a local optimum that improves the adversarial example's effectiveness is reached.

- Reevaluating Fitness

After the hill climbing phase, the fitness of the individuals that underwent local search is reevaluated using the Malconv model. This step ensures that the modifications made during hill climbing have improved the adversarial examples' evasion capabilities.

- Fitness Threshold and Termination

Like it has been assumed in many proposals [20,21,28], we consider that prediction values larger than 0.5 indicate that a sample must be classified as malware, elsewhere, the sample must be classified as goodware. Thus, after the fitness evaluation of each individual(Algorithm 4), we determine the success of the adversarial example. If the prediction is below 0.5 (or fitness is above 0.5), indicating that the malware successfully evades detection, we consider it a successful adversarial example.

However, if the prediction is above 0.5 (or fitness is bellow 0.5), the malware is considered unsuccessful in evading detection. In this case, we continue to the next generation, repeating the process from step 4 onwards. The

individual which generated the unsuccessful malware is discarded, and new offspring are generated through selection, crossover, and mutation operations.

This iterative process allows us to refine and improve the AEs over multiple generations. The goal is to find AEs that consistently evade detection by the Malconv model leading to potential insights into the vulnerabilities of the model and other models from the same family namely deep neural network models, and the robustness of malware detection systems. The termination condition for this process is defined as a combination of the number of generations and the fact that the malware is detected as goodware (this corresponds to a prediction value below 0.5).

4. Experiments and Results

4.1. Dataset Description

The dataset we used in our approach is the DikeDataset[29], a meticulously curated collection of labeled benign and malicious PE and OLE files, it counts 9000 malware and 1000 goodware. The primary objective of this dataset is to train and evaluate artificial intelligence algorithms for predicting the maliciousness of files and determining their membership in specific malware families. The dataset labels in the DikeDataset are represented numerically, with values ranging from 0 to 1, indicating the degree of malice associated with each file. The lowest size of malware used in our experiments is about 80kb and the highest is about 600kb.

All our experiments were performed on a computer with the following configuration: CPU: Intel Core i5-4210M, 2.60GHz, RAM: 8GB, x64-bit, Graphics Card: Intel HD Graphics 4600.

4.2. Parameters Settings

Used parameters with our methods are in table1:

Table 1. Parameters of all used algorithms

Algorithm	Pop.	Nb Gen	Selection	Crossover	Mutation
AG	50/10	25+	Seltournament (6/3)	cxtwopoint crossover operator with probability = 50%	mutflipbit mutation operator with probability =20%
Memetic	10	25	Seltournament (3)	cxtwopoint crossover operator with probability = 50%	mutflipbit mutation operator with probability= 20%
Hill climbing	Subset size of 50				

4.3. Results

To evaluate the performance of the proposed method, many experiments were conducted on a variety of malwares and goodwares. To omit the problem of imbalanced data, we select randomly 1000 malware and 1000 goodware from the 9000 malwares of the dataset to train Malconv network. In another hand we select randomly 100 malwares from the remaining unselected malwares to generate Adversarial examples.

In order to assess the efficiency of our Memetic based evasive malware:

- First, we tested the genetic algorithm approach with a population size of 50 individuals (Table 2: Columns 6,7,8). An initial prediction is computed for each original malware, and then at each generation, we compute the prediction rate of the new modified malware. This rate is reduced until it reaches 0.5 or lower, in this case, the produced malware is judged a successful adversarial malware that can effectively evade the detection. If the maximum number of generations which is 25 is reached, but the prediction fails to reach 0.5, the original malware is ignored and is no longer valid to produce new malware variants.
- Second, because the Memetic algorithm is suggested to be slower than genetic algorithm, our objective is to minimize the number of generations, thus, we tested the Memetic algorithm with 10 individuals (Table 2: Columns 3,4,5) and conducted the same experiments as with genetic algorithm.

With both algorithms, we deduced the number of the generation in which the prediction rate of 0.5 is reached and the required time in minutes.

Table (2) presents obtained results for 100 original malwares. This evaluation allows:

To examine the impact of population size on the effectiveness of both algorithms in generating AEs to evade malware detection, and,

To show the success of Memetic algorithm in generating AEs in contrast of the genetic algorithm in doing so.

- -Third, to study the impact of population size in generating adversarial examples, we tested the genetic algorithm approach with a population size of 10 individuals (Table 3).

Table (3) focuses on comparing genetic algorithm against the Memetic algorithm approach with the same population size which is 10. We only chose the malwares that succeed to be AEs in table 2 (26 malwares).

It is worth noting that in this paper, we estimated that there is no need to implement a sandbox to make sure all new mutations are functional before checking whether they are also evasive, because the used perturbation consists of adding a section at the end of the PE file, and in this case, the functionality of the PE file is preserved according to[7,10].

Table 2. Memetic algorithm with population size 10 and genetic algorithm with population size of 50 in the generation of 100 adversarial malwares

Malware Name	Initial Prediction	MA Time (mn)	MA Min Prediction 10 individual	MA Generations	GA Time (mn)	GA Min Prediction 50 individual	GA Generations
000d1ba	0.999412835	13.61123461	0.452880472	13	4.828982	0.98936367	25+
000d623	0.993823886	7.723150333	0.39293316	7	5.188897	0.70996803	25+
000e731	0.973636925	5.481930563	0.436591864	5	5.166708	0.897034764	25+
002d72a	0.997302771	13.04369545	0.35421133	12	5.223968	0.967905521	25+
00ab1c6	0.997607529	11.0853188	0.439802468	10	5.04205	0.893437207	25+
00dbed	0.993287206	12.26620739	0.347544253	11	4.964102	0.85128814	25+
02ad00	0.998556852	19.98556223	0.211905688	18	5.287467	0.961169481	25+
02b6cfb	0.988032639	8.986126105	0.381986976	8	5.175056	0.758048773	25+
0b06ed	0.97939539	9.90253845	0.420599759	8	5.387898	0.952501178	25+
0b25e6	0.99427563	8.925917108	0.324236035	12	5.22251	0.765558779	25+
0b269a	0.998900294	13.19422797	0.412822217	13	5.492412	0.967009425	25+
0b328c	0.999649346	20.57058286	0.394680083	16	5.659335	0.965153337	25+
0b41fe	0.969264925	11.40305769	0.443954319	8	4.110614	0.447938204	18
0b445	0.961407006	15.87464245	0.458567291	11	3.381857	0.498321176	14
0b575e	0.999579489	26.09760502	0.244422078	19	6.062447	0.954518318	25+
0b626ff	0.992215514	19.51573189	0.475713193	16	5.840186	0.82665652	25+
0b66c3f	0.928288996	79.05951942	0.475568503	16	7.661204	0.673520327	25+
0b673d	0.993508995	10.39063851	0.413135231	8	8.157501	0.987458885	25+
0b7ba7	0.728037477	10.05325291	0.464648604	8	7.148687	0.758048773	25+
0b8ad8	0.991936743	14.58298917	0.379579246	11	4.968175	0.852547288	25+
0b9996	0.999588668	25.35407868	0.27461499	19	4.574189	0.990362763	25+
0bc38d	0.998643875	25.11935322	0.470483989	14	5.09409	0.992877901	25+
0bd575	0.991705775	21.48439821	0.492647409	10	5.086544	0.907415092	25+
0be469	0.993354678	18.32547148	0.418112665	8	5.2439	0.780185044	25+
0bf37d	0.952174366	20.04276294	0.463632643	8	3.494744	0.493274212	16
0bfa7a	0.912130654	19.37016685	0.495369732	7	4.439243	0.48886624	20
0c839f9	0.90747726	8.301146825	0.3259148	3	0.218519	0.391690314	1
0ca892a	0.852082014	3.511094312	0.394941151	3	0.484081	0.330154955	2
0cb1707	0.964990258	8.413723738	0.428128928	8	5.255518	0.829692483	25+
0ccdac	0.98633343	33.38413339	0.496122658	10	6.383397	0.503566444	25
0cd8f5d	0.996880233	10.57553939	0.450523138	10	8.102604	0.976029158	25+
0cdc2fd	0.993904173	11.69725081	0.480871767	10	8.649563	0.937586725	25+
0d148c6	0.996237338	21.61184148	0.433713347	18	5.302816	0.994839847	25+
0d4d4	0.977840602	12.36783206	0.346672982	11	9.437703	0.904402852	25+
0de38	0.980777323	10.99880155	0.469530106	9	5.20164	0.723987103	25+
0e0c	0.97841239	15.7671218	0.410084248	10	5.188034	0.853684068	25+
0e5147	0.888022363	2.224776093	0.453321993	2	0.203077	0.452500671	1
0e545	0.997968674	12.20115267	0.351854712	11	4.110614	0.378388226	9
0e568d	0.99295646	17.19283314	0.454013258	9	5.312664	0.801853418	25+
0e5834a	0.997475266	7.523782227	0.492886633	7	8.061629	0.890783191	25+
0e6552a	0.908392727	3.257904787	0.475143641	3	4.766958	0.433918595	22
0e76364	0.997101903	11.01411014	0.44733876	10	8.87966	0.950139284	25+
0e807899	0.998297393	13.62635151	0.474462569	12	8.37379	0.922519863	40+
0e80947db	0.989724934	9.096953155	0.454938024	7	6.11979	0.460484058	25
0e83016a20	0.998579979	17.51728673	0.280992955	10	8.953578	0.924651861	25+

0e876a	0.974769592	3.761154995	0.328944176	2	1.601469	0.350069433	7
0e90505	0.993205428	27.46842651	0.389202327	13	5.345003	0.978252172	25+
0e9159a9	0.995217085	39.92487917	0.445388645	17	6.11979	0.991322637	25+
0ea38c5a	0.999344289	47.11197315	0.313687176	15	5.154608	0.969277918	25+
0ec199c	0.99750644	43.4542839	0.245217428	12	5.492412	0.776708543	40
0ecc	0.937462389	24.43710787	0.386530787	6	5.816572	0.498843789	11
0ecf8a4	0.996136725	9.886277807	0.37090826	9	5.492412	0.830397487	25+
0eddc7a	0.899862289	4.336961035	0.310796797	4	2.423934	0.407560825	5
0f368926	0.889388978	17.13377888	0.335829049	12	1.131349	0.368695855	2
0f571ef0	0.955163479	21.03077984	0.353987753	10	8.673237	0.622172654	25
0fbbf	0.999575436	28.6478361	0.371554047	13	5.188034	0.996301472	40
1a41f358	0.995136738	11.93352423	0.411915153	20	5.405453	0.992308378	25+
1aad4af9	0.956178606	16.00078299	0.43273136	12	25.1713	0.636435151	40
1abde801	0.846284568	1.269326928	0.251989245	1	0.631026	0.256334662	1
1ad26d8e	0.972956121	14.68194296	0.448429853	11	8.034011	0.778618693	25+
1aee80e2	0.997985303	11.48720547	0.479220062	8	8.541651	0.79992193	25+
1b389fa	0.580829024	3.197176643	0.455786765	5	1.135259	0.498354107	5
1ccd12c7	0.993967474	27.79395788	0.45682174	12	5.345003	0.87714982	40
1cd7afd	0.881123066	19.25717144	0.411600381	8	4.65551	0.487330586	23
1dbfbe7	0.998214304	20.74066933	0.405044824	8	9.074766	0.731464028	25+
1fcf55	0.967099965	12.18178064	0.323096752	11	8.673237	0.535451293	25
1fd1d4	0.999581337	23.46870101	0.459344983	19	5.382717	0.997136056	25+
1fda73	0.922158599	11.86566439	0.469772816	10	5.202302	0.790673614	25+
1fda77	0.999606669	18.63499727	0.454215825	14	5.404464	0.98446697	25+
1fe3475	0.847312987	2.742010397	0.406707525	2	2.03475	0.446347713	9
1ff93a9	0.998492658	14.07401216	0.473918766	10	5.345003	0.97271353	25+
1ffd84	0.864335716	1.488715513	0.415620357	1	0.236249	0.337292492	1
2b411b	0.997850358	35.56751657	0.292053044	20	0.997559	0.992228329	25+
2b8274	0.758167863	17.61481672	0.486374974	25	2.952722	0.401086777	14
2b832e	0.994533181	25.23878159	0.34646821	12	5.426925	0.888349891	25+
2b934ca	0.986499846	37.13563555	0.34646821	16	5.303906	0.936501563	25+
2b97ca2	0.999088645	29.52651323	0.471598506	12	5.205182	0.970559061	25+
2dc5531	0.999356687	20.52495456	0.411891967	18	5.516382	0.986080825	25+
2dcaf04	0.9536466	7.686784427	0.226673603	7	1.537329	0.435638368	8
2dd089b	0.999582767	21.42179483	0.381497711	18	5.706327	0.997525096	25+
2df88c9	0.998740911	10.63417082	0.41247955	9	5.557326	0.816071332	25+
2dfe	0.997882009	25.0040904	0.38482058	19	5.283966	0.996241331	25+
2e14d	0.997941911	13.8404796	0.443197846	9	5.318006	0.892261207	25+
2e2a45	0.988497853	43.626	0.535451293	25	5.154608	0.942445457	25+
2e5fb	0.99960649	42.91828385	0.47227934	17	5.232804	0.981294096	25+
2e6e825	0.996190012	29.71324144	0.478178144	11	5.658289	0.943344414	25+
3fa44906	0.999176085	27.79000452	0.481719941	10	5.774209	0.99601692	25+
3fa9c141	0.909792662	5.311738343	0.436534494	2	0.621312	0.426870763	3
3faec1e7	0.907690346	14.18755539	0.496204615	5	1.942501	0.49906978	9
3fb50cf	0.970266283	50.36	0.673520327	25	5.915463	0.830421269	25
4a9e7e2	0.980166137	4.162888417	0.495502084	4	5.32331	0.951628208	25+
4ae1ad6	0.980782568	14.20816338	0.478706539	10	5.478883	0.630299449	25
4b0aaf5	0.985360205	13.11142762	0.471595705	12	5.345003	0.751044154	25+
4b125	0.999469876	21.83711277	0.471595705	12	5.483574	0.980629981	25+
4b769d	0.984389901	11.68125295	0.394959778	11	5.182304	0.846409321	25+
984541e	0.736750126	6.608465762	0.450218827	9	5.6951	0.798329294	25+
bf534f3	0.950698376	6.095476988	0.290416598	5	7.97575	0.505183637	25

bf839e	0.769838214	1.268298588	0.356319249	1	0.331381	0.40473628	1
d5c26fed	0.630749285	2.240088618	0.376110941	2	0.61834	0.31850341	2
f8ef3e3b	0.606728137	6.956787637	0.443802714	5	8.147903	0.381122231	22

Table 3. Memetic algorithm vs genetic algorithm with population size of 10 in the generation of adversarial malwares

Malware Name	MA Time (mn)	MA Min Prediction 10 individual	MA Generations	GA Time (mn)	GA Min Prediction 10 individual	GA Generations
0b41fef68	1.269327	0.251989	1	1.1423	0.903218508	25
0b445bba	2.240089	0.376111	2	1.2094	0.907194257	25
0bf37d24	3.511094	0.394941	3	1.187	0.75009501	25
0bfa7a40	1.488716	0.41562	1	0.7684	0.460218042	16
0c839f90	3.761155	0.328944	2	1.1291	0.619088173	25
0ca892a6	17.13378	0.335829	12	0.0526	0.35329923	1
0e5147b	12.20115	0.351855	11	1.1603	0.516726613	25
0e545292	6.956788	0.443803	5	1.1881	0.982938886	25
0e6552ac	8.301147	0.325915	3	1.1649	0.957994282	25
0e80947d	17.61482	0.486375	25	1.1896	0.925188959	25
0e876ae	1.268299	0.356319	1	1.1521	0.68820709	25
0eccf6ef	4.336961	0.310797	4	1.2424	0.852847397	25
0eddc7ab	5.311738	0.436534	2	1.0993	0.75313884	25
0f3689265	3.257905	0.475144	3	0.7119	0.39843604	14
1abde801a	7.686784	0.226674	7	0.0458	0.260212004	1
1b389fab	2.74201	0.406708	2	0.5404	0.493309379	11
1cd7afd4	11.40306	0.443954	8	1.1232	0.756913006	25
1fe3475	2.224776	0.453322	2	1.2335	0.812570512	25
1ffd84c2	9.096953	0.454938	7	0.0596	0.417993784	1
2b82747	19.25717	0.4116	8	1.2146	0.81317544	25
2dcaf04	19.37017	0.49537	7	1.0941	0.673018873	25
3fa9c1416	20.04276	0.463633	8	1.1515	0.54717207	25
3faec1e7	15.87464	0.458567	11	1.1947	0.537118375	25
bf839e2f	3.197177	0.455787	5	0.429	0.428981602	13
d5c26fed	24.43711	0.386531	6	0.824	0.485638946	18
f8ef3e3b18	14.18756	0.496205	5	1.3631	0.727774918	25

4.4. Discussion

In order to evaluate the generated AEs, we first discuss the obtained results with genetic and memetic algorithms evading Malconv network. Then to evaluate the generated AEs against other malwares detectors , We implemented four more options, two top commercial scanners :ESET and Kaspersky , and , two ML models: KNN and Decision tree (DT). Our aim is to obtain more robust AEs capable of evading all five classifiers, all these classifiers are used to simulate a black box classifiers.

A. Memetic vs. Genetic Algorithms: Malconv model

- Evading Rate

From table 2, it can be observed that when using a population size of 50 with the genetic algorithm approach, only 26 out of 100 samples successfully bypassed the MalConv model under 25 generation. This indicates limitations in finding effective modifications to the PE files that can evade detection. In contrast, the memetic algorithm, leveraging hill climbing, demonstrated a significantly higher success rate with 98 out of 100 samples successfully evading detection. Averaging 9 generation per each malware sample, the memetic algorithm is able to refine modifications and navigate the search space effectively contributed to its superior performance compared to the genetic algorithm. We can

also notice that malwares that could not be modified with memetic algorithm, could not be modified with genetic algorithm either.

Additionally, from table (3), it can be seen that when the population size was reduced to 10 as shown, the success rate of genetic algorithm further decreased, with only 8 samples evading detection. However, memetic algorithm with population size 10, achieves a higher success rate (98%) compared to the genetic algorithm (26%) when the population size is 50, and (8%) when the population size is 10. These results underscore the memetic algorithm's ability to iteratively improve modifications and explore the search space more effectively resulted in a higher success rate, regardless to population size.

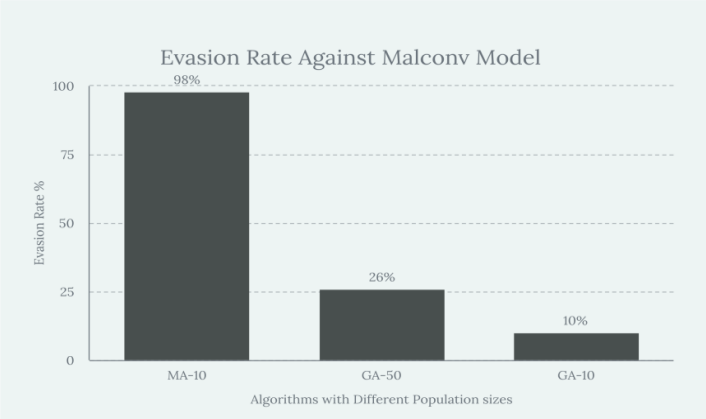

Fig.3. Evasion rate against MalConv model

- Number of Generations

Upon comparing the number of generations reached by each approach, a clear distinction emerges. Our memetic approach demonstrates a notable advantage by requiring significantly fewer generations to generate successful adversarial examples. In the genetic algorithm with a population size of 50, a substantial rate of 74% of the samples fail to evade detection even after 25 generations. Similarly, in the genetic algorithm with a population size of 10, a significant rate of 92% of the samples also fail to evade detection after 25 generations. These results underscore the limitations of the genetic algorithm in finding effective modifications to bypass detection within the specified generation constraints. This was the limitation of the work in [17], where the authors state that 76% of PE malware files are unmodifiable under a greater number of generations. In contrast, our memetic algorithm exhibits a higher efficiency, enabling a greater number of samples to successfully evade detection within the given 25-generation limit. This suggests that the memetic algorithm's incorporation of hill climbing and local search techniques facilitates more effective exploration of the search space, leading to quicker convergence towards successful adversarial examples.

- Processing Time

The memetic algorithm exhibits a wider range of execution times, varying from a minute and 26 seconds to around 50 minutes. Although the average execution time is relatively high, most samples are generated within a reasonable timeframe averaging 5 minutes. In contrast, the GA with a population size of 10 or 50 demonstrates significantly shorter execution times, ranging from fractions few second to few minutes. this can be explained by the fact that the MA takes longer due to the additional computational effort of refining modifications by exploring both global and local spaces. The choice between the two approaches should consider the trade-off between success rate and generating time based on specific requirements and constraints. However, it is worth noting that we are about generating successful AEs, and this is done generally in an offline manner, thus processing time is not always a significatif constraint.

Table 4. Memetic algorithm vs genetic algorithm evasion rate comparison

	GeneticAlgorithm :Evasion rate		Memetic Algorithm :Evasion rate
Population Size	50	10	10
KNN	9/100 9%	3/100 3%	35/100 35%
Decision Tree	8/100 8%	2/100 2%	44/100 44%
Kaspersky	8/100 8%	2/100 2%	26/100 25%
ESET	6/100 6%	1/100 1%	10/100 10%

B. Memetic vs. Genetic Algorithms: ML Models and Commercial Antivirus

In Table (4), we present a comparison of the evasion rates between the MalConv model and each of the four detectors individually: KNN, Decision Tree, Kaspersky, and ESET. This analysis allows us to assess the performance of

different algorithms, namely the Genetic Algorithm (GA) with a population size of 50, the GA with a population size of 10, and the Memetic Algorithm (MA) with a population size of 10, in generating AEsthat bypass the detection mechanisms.

Fig.4. Evasion rate: ml models and antiviruses simultaneously with Malconv

- Evading Machine Learning Models KNN-Decision Tree

In the Genetic Algorithm with a population size of 50, 9% of the samples were able to simultaneously evade both the KNN detector and the MalConv model, and 8% evaded the Decision Tree detector and the MalConv model. These results suggest that the algorithm was able to generate AEsthat effectively bypassed both the traditional machine learning detectors and the MalConv model. However, the success rates were relatively low, indicating room for improvement. Decreasing the population size to 10 reduced the success rates against these detectors, with evasion rates of 3% and 2%, respectively. These results suggest that a smaller population size may limit the algorithm's ability to generate highly effective adversarial examples. In contrast, the Memetic Algorithm achieved significantly higher quantities of successful AEsand higher evasion rates. With a population size of 10, 35% were able to evade both the KNN detector and the MalConv model, and 44% evaded the Decision Tree detector and the MalConv model. The Memetic Algorithm's ability to refine modifications using hill climbing and local search techniques contributed to its improved evasion rates, effectively exploiting the weaknesses of the KNN and Decision Tree detectors, while simultaneously bypassing the MalConv model. These results suggest that the Memetic Algorithm has a stronger capability to generate AEsthat can evade both the traditional detectors and the MalConv model, highlighting its potential in developing more robust evasion strategies.

- Evading Commercial Antivirus

The experiments conducted to compare the performance of the Genetic Algorithm (GA) and the Memetic Algorithm (MA) with different population sizes have provided valuable insights. The GA with a population size of 50 achieved evasion rates of 8% for Kaspersky and 6% for ESET. These evasion rates represent the percentage of samples capable of evading both the MalConv model and either Kaspersky or ESET antivirus systems simultaneously. Reducing the population size to 10 resulted in evasion rates of 2% for Kaspersky and 1% for ESET. In contrast, the Memetic Algorithm with a population size of 10 outperformed both versions of the GA in terms of evasion rates. It achieved higher evasion rates of 25% for Kaspersky and 10% for ESET, indicating that a larger proportion of samples generated by the Memetic Algorithm were capable of simultaneously evading Malconv and Kaspersky and ESET antivirus systems. These results highlight the stronger capability of the Memetic Algorithm in generating AEsthat can bypass both the commercial antivirus systems and the MalConv model. The findings suggest that further optimization and improvement of the Memetic Algorithm could lead to the development of more robust evasion strategies against such systems.

- Evading Both Machine Learning Models and Commercial Antivirus

In order to obtain more robust adversarial examples, we tested all successfully generated AEs from Malconv model with the other four classifiers: KNN, DT, Kaspersky and ESET.

Table 5. Memetic algorithm vs genetic algorithm evasion rate against all four detectors at the same time

	Genetic algorithm		Memetic algorithm
Population size	50	10	10
All four detectors	1/100 1%		4/100 4%

The results in Table (5) indicate that the Memetic Algorithm outperformed the Genetic Algorithm in terms of generating examples that could successfully evade all four detectors. In the case of the GA, regardless of the population size (50 or 10), only one example out of 100 instances was able to bypass all detectors. On the other hand, the MA with a population size of 10 produced four examples out of 100 that evaded all detectors. This suggests that the MA was more effective in finding solutions that were able to evade the detection mechanisms of the four algorithms being tested. It implies that the MA, with its combination of genetic and local search techniques, potentially provided a better exploration and exploitation of the solution space, resulting in a higher success rate in evading the detectors compared to the GA.

C. Assessing the Effectiveness of our Method with Other State-of-the-Art Techniques

Finally, we compared the results of the proposed method with those obtained by algorithms in the literature. In Table (6) we recorded the results of two methods for the evasion rates with different sizes. These methods are: Optimization of code caves [20] and Gradient-based attack that can evade Malconv model. The results of the three approaches as shown in Table 6, taking into consideration the sample size factor, reveal interesting insights. The Memetic Algorithm, applied to a sample size of 100, achieves an impressive evasion rate of 98% when tested against the Malconv Model. This suggests that the algorithm performs well in generating adversarial samples that can successfully bypass the detection capabilities of the model. Similarly, the optimization of code caves in malware binaries, with a larger sample size of 2036, demonstrates a high evasion rate of 97.99% when tested against the same Malconv Model. This indicates the efficacy of this approach in generating evasive malware samples on a larger scale. However, it's worth noting that the approach of adversarial malware binaries, with a smaller sample size of 200, achieves a comparatively lower evasion rate of 60% when tested against deep network-based malware detection methods.

Table 6. Memetic algorithm vs other state of the art methods

	Memetic Algorithm	Optimization of code caves[20]	Adversarial Malware Binaries[21]
Sample size	100	2036	200
Test Environment	Malconv Model	Malconv Model	Malconv Model
Evasion Rate	98%	97.99%	60%

D. Malware detection Enhancement

The implications of our work extend beyond the generation of adversarial examples. By identifying vulnerabilities in the Malconv model, and, understanding the limitations and weaknesses of machine learning-based defenses, allows us to develop more resilient and effective detection mechanisms in the face of evolving threats. Indeed, to improve malware detection systems, we conducted a final experiment where we first trained the two ML models, namely KNN and DT, on the same initial set of 1000 malwares and 1000 goodwares. Subsequently, we tested all 98 successfully generated AEs with the malconv model. Next, we trained both ML models on the initial set of 1000 malwares, adding half of the newly generated AEs from the malconv model (48). Finally, we tested both ML models on the other half of the generated AEs with malconv model. The objective of this experiment is to study the impact of adding the new generated AEs during the training step on the improvement of the detection of new malwares. The results obtained before and after adding generated AEs in the training step are summurized in table (7).

Table 7. KNN and DT performances when training with and without new generated AEs

Model	Detection rate (without generated AEs)	Detection rate (with generated AEs)
KNN	65%	98%
DT	56%	98%

From table (7), it can be observed that the detection rate of malwares after adding the new AEs has significantly increased from 65% and 56% for both KNN and DT models to 98%. This can be explained by the fact that when the new generatd AEs are used to retrain the ML-based malware detector, an improvement in its robustness is guaranted.

5. Conclusions and Future Work

In conclusion, this paper has presented a novel approach for generating effective AEscapable of evading malware detection systems using the Memetic Algorithm. Through a comprehensive step-by-step outline, we have demonstrated the effectiveness of our approach in generating AEsthat can successfully bypass the detection mechanisms of the Malconv model as well as other machine learning and commercial detectors. By leveraging evolutionary search techniques, crossover, mutation, and local search operations, we iteratively refine and improve the AEsover multiple generations. Our experimental results have shown that the Memetic Algorithm achieves a higher evasive rate of 98% compared to genetic algorithm with an evasive rate of 26% while utilizing fewer generations and population size. Additionally, it achieves a comparative evasive rate with works presented in the literature namely Optimization of code caves wich achieves an evasive rate of 97,9%, and a higher rate compared to Adversarial Malware Binaries wich achives a rate of 60%. In another hand the proposed approach achieved higher evasion rates against other machine learning and commercial malware detectors compared to Genetic-Based Algorithms. It implies that the MA, with its combination of genetic and local search techniques, potentially provided a better exploration and exploitation of the solution space, resulting in a higher success rate in evading the detectors compared to GA. Ultimately, the generated AEs are used to retrain the ML detectors, resulting in an enhancement of its robustness from 65% and 56% for both KNN and DT models to 98% detection rate.

While this work presents significant contribution, it is important to acknowledge its limitations. The experiments conducted focused on a specific set of malware detection systems and may not capture the full spectrum of potential vulnerabilities across different models. Therefore, further investigations are necessary to evaluate the generalizability and scalability of our approach to a wider range of malware detection systems. Moreover, future research should explore the incorporation of additional optimization techniques and advanced algorithms to further enhance the efficacy of adversarial example generation. Additionally, the deployment of real-world case studies and the evaluation of the proposed approach in production environments would provide valuable insights into its practical applicability and performance under real-world constraints.

References

[1] Ibitoye O., Abou-Khamis R., El Shehaby M., Matrawy A., Omair Shafiq M., "The Threat of Adversarial Attacks Against Machine Learning in Network Security: A Survey", 2020. DOI.org/10.48550/arXiv.1911.02621

[2] Alotaibi A., Rassam M.A. "Adversarial Machine Learning Attacks against Intrusion Detection Systems: A Survey on Strategies and Defense". Future Internet, vol.15, No.2, 62, 2023. DOI.org/10.3390/fi15020062

[3] Xiangjun L., Ke K., Su X., Pengtao Q., Daojing H., "Feature selection-based android malware adversarial sample generation and detection method". IET Information Security, Vol.15,No.6,pp.401-416, 2021. DOI.org/10.1049/ise2.12030

[4] Gupta S., Lamba S., Soni N., Priyadarshi P., "Evading Detection Systems by Generating Adversarial Malware Examples", In: Agrawal, R., Sanyal, G., Curran, K., Balas, V.E., Gaur, M.S. (eds), Cybersecurity in Emerging Digital Era. (ICCEDE), Communications in Computer and Information Science, vol. 1436, pp.51-60, 2020. DOI.org/10.1007/978-3-030-84842-2_4

[5] Park D., Yener B., "A survey on practical AEs for malware classifiers", In: Reversing and Offensive-oriented Trends Symposium, ACM, pp. 23–35, 2020. DOI.org/10.1145/3433667.3433670

[6] Deqiang L., Qianmu L., Yanfang Y., Shouhuai X., "Arms race in adversarial malware detection: A survey", ACM Computing Surveys (CSUR), vol.55,No1,pp.1–35, 2021. DOI.org/10.1145/3484491

[7] Xiang L., Lingfei W., Jiangyu Z., Zhenqing Q., Wei D., Xiang C., Yaguan Q., Chunming W., Shouling J., Tianyue L., Jingzheng W., Yanjun W., "Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art". Computers & Security, vol.128, 2023. DOI: 10.1016/j.cose.2023.103134

[8] Yanchen Q., Weizhe Z., Zhicheng T., Laurence T. Y., Yang L., Mamoun A., "Adversarial malware sample generation method based on the prototype of deep learning detector", Computers & Security, vol.119, pp.102762, 2022. DOI.org/10.1016/j.cose.2022.102762.

[9] Xiao G., Li J., Chen Y., Li K., "MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks", Journal of Parallel and Distributed Computing, vol.141,pp. 49-58, 2020. DOI:10.1016/j.jpdc.2020.03.012.

[10] Anderson H.S., Kharkar A., Filar B., Roth P., “Evading machine learning malware detection. In Black Hat” 2017. URL: www.blackhat.com/docs/us-17/thursday/us-17-Anderson-Bot-Vs-Bot-Evading-Machine-Learning-Malware-Detection-wp.pdf

[11] Alzantot M., Sharma Y., Chakraborty S., Zhang H., Hsieh C.-J., Srivastava M. B., "GenAttack: Practical Black-box Attacks with Gradient-Free Optimization", Genetic and Evolutionary Computation Conference (GECCO), pp. 1111–1119, 2019. DOI.org/10.1145/3321707.3321749

[12] Demetrio L., Coull S. E., Biggio B., Lagorio G., Armando A., Roli F., "Adversarial Examples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection", ACM Transactions on Privacy and Security, vol.24, No.4, Article No.: 27,pp. 1–31, 2021. DOI.org/10.1145/3473039.

[13] Xintong. L., Qi. L., "An IRL-based malware adversarial generation method to evade anti-malware engines". Computers & Security, vol.104, No.C,pp. 102118, 2021. DOI: 10.1016/j.cose.2020.102118.

[14] Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P. "Adversarial Examples for Malware Detection". In: Foley, S., Gollmann, D., Snekkenes, E. Computer Security – (ESORICS),Vol 10493, 2017. DOI.org/10.1007/978-3-319-66399-9_4

[15] Yang, C., Xu, J., Liang, S. et al. "DeepMal: maliciousness-Preserving adversarial instruction learning against static malware detection". Cybersecurity, vol. 4,Article No.16, 2021. DOI.org/10.1186/s42400-021-00079-5

[16] Yuan J., Zhou S., Lin L., Wang F., Cui J., "Black-box adversarial attacks against deep learning based malware binaries detection with GAN", In: European Conference on Artificial Intelligence IOS Press, vol. 325, pp. 2536–2542. 2020.
[17] Castro R. L., Schmitt C., Dreo G., "AIMED: Evolving Malware with Genetic Programming to Evade Detection", 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 240-247, 2019., DOI: 10.1109/TrustCom/BigDataSE.2019.00040.
[18] Wang X., Miikkulainen R., "MDEA: Malware detection with evolutionary adversarial learning", 2020, DOI.org/10.48550/arXiv.2002.03331
[19] Demetrio L., Biggio B., Lagorio G., Roli F., Armando A., "Functionality-preserving black-box optimization of adversarial windows malware", 2020, DOI.org/10.48550/arXiv.2003.13526.
[20] Y. Javier, Pardo G.E., Tapiador J., "Optimization of code caves in malware binaries to evade machine learning detectors", Computers & Security,vol.116,PP.102643, 2022.DOI.org/10.1016/j.cose.2022.102643.
[21] Kolosnjaji B., Demontis A., Biggio B., Maiorca D., Giacinto G., Eckert C., Roli F., "Adversarial malware binaries: Evading deep learning for malware detection in executables", In: European Signal Processing Conference, IEEE, pp. 533–537, 2018. URL: https://www.eurasip.org/Proceedings/Eusipco/Eusipco2018/papers/1570440156.pdf.
[22] Aydogan E., Sen, S."Automatic Generation of Mobile Malwares Using Genetic Programming", In European Conference on the Applications of Evolutionary Computation, ser. EvoApplications. Springer, vol.9028, pp. 745–756, 2015. doi.org/10.1007/978-3-319-16549-3_60
[23] Liu X., Du X., Zhang X., Zhu Q., Wang H., Guizani M. "Adversarial Samples on Android Malware Detection Systems for IoT Systems". Sensors (Basel). Vol.19,No.4, 974, 2019. doi: 10.3390/s19040974.
[24] Guangquan X., Hongfei S., Jingyi C., Hongpeng B., Jiliang L., Guangdong B., Shaoying L., Weizhi M., Xi Z., "GenDroid: A query-efficient black-box android adversarial attack framework", Computers & Security, vol. 132, 103359, 2023. DOI.org/10.1016/j.cose.2023.103359.
[25] Raff E., Barker J., Sylvester J., Brandon R., Catanzaro B., Nicholas C., "Malware Detection by Eating a Whole EXE", 2017. DOI.org/10.48550/arXiv.1710.09435
[26] Ahandani M.A., Vakil-Baghmisheh MT., Talebi M., "Hybridizing local search algorithms for global optimization". Comput Optim Appl, vol.59, pp. 725–748, 2014. DOI.org/10.1007/s10589-014-9652-1
[27] "Memetic algorithm–An overview "ScienceDirect Topics URL : https://www.sciencedirect.com/topics/computer-science/memeticAlgorithm #:~:text=Memetic%20algorithms%20(MAs)%20are%20evolutionary, search%20processes%20to%20refine%20individuals.
[28] Kreuk, F., Barak, A. , Aviv-Reuven, S. , Baruch, M. , Pinkas, B. , Keshet, J. "Deceiving end-to-end deep learning malware detectors using adversarial examples", 2018. DOI.org/10.48550/arXiv.1802.04528
[29] https://github.com/iosifache/DikeDataset#description-%EF%B8%8F [Dike Dataset]

Authors' Profiles

Khadoudja Ghanem received her Ph.D. degrees in Institute of Computer Science, Mentouri University. Now, she is Associate Professor at Abdelhamid Mehri Constantine2, Algeria. Her main research interests include Artificial intelligence, Machine and Deep Learning, Optimization and malware analysis.

Ziad Kherbach received his Master's degree in Networks and Distributed Systems from Abdelhamid Mehri, Constantine2 University, Algeria. His research areas include Optimization, artificial intelligence security, malware.

Omar Ourdighi received his Master's degree in Networks and Distributed Systems from Abdelhamid Mehri, Constantine2 University, Algeria. His research areas include networks, artificial intelligence security, malware.

How to cite this paper: Khadoudja Ghanem, Ziad Kherbache, Omar Ourdighi, "Enhancing Adversarial Examples for Evading Malware Detection Systems: A Memetic Algorithm Approach", International Journal of Computer Network and Information Security(IJCNIS), Vol.17, No.1, pp.1-16, 2025. DOI:10.5815/ijcnis.2025.01.01