An Evolutionary based Generative Adversarial Network Inspired Approach to Defeating Metamorphic Malware

Kehinde O. Babaagba
Edinburgh Napier University
Edinburgh, United Kingdom
K.Babaagba@napier.ac.uk

Jordan Wylie
Edinburgh Napier University
Edinburgh, United Kingdom
40286030@live.napier.ac.uk

ABSTRACT

Defeating dangerous families of malware like polymorphic and metamorphic malware have become well studied due to their increased attacks on computer systems and network. Traditional Machine Learning (ML) models have been used in detecting this malware, however they are often not resistant to future attacks. In this paper, an Evolutionary based Generative Adversarial Network (GAN) inspired approach is proposed as a step towards defeating metamorphic malware. This method uses an Evolutionary Algorithm as a generator to create malware that are designed to fool a detector, a deep learning model into classifying them as benign. We employ a personal information stealing malware family (Dougalek) as a testbed, selected based on its malicious payload and evaluate the samples generated based on their adversarial accuracy, measured based on the number of Antivirus (AV) engines they are able to fool and their ability to fool a set of ML detectors (k-Nearest Neighbors algorithm, Support Vector Machine, Decision Trees, and Multi-Layer Perceptron). The results show that the adversarial samples are on average able to fool 63% of the AV engines and the ML detectors are susceptible to the new mutants achieving an accuracy between 60%-77%.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; **Malware and its mitigation**; • **Computing methodologies → Machine learning algorithms**.

KEYWORDS

Metamorphic Malware, Evolutionary Algorithm, Generative Adversarial Network

ACM Reference Format:
Kehinde O. Babaagba and Jordan Wylie. 2023. An Evolutionary based Generative Adversarial Network Inspired Approach to Defeating Metamorphic Malware. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3583133.3596362

This work is licensed under a Creative Commons Attribution International 4.0 License.
GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0120-7/23/07.
https://doi.org/10.1145/3583133.3596362

1 INTRODUCTION

Malicious attacks continue to pose a challenge to information assets and computer systems as a whole. This has been heightened due to ubiquitous computing which implies that every and anything is now connected to the internet. The recent 2022 global threat report by Crowdstrike highlights the increase in malware attack with 21 newly named adversaries discovered, 45% increase in interactive intrusions and 82% increase in ransomware based data leakages [8].

The aforementioned malware attacks originate from various malware groups. A particularly dangerous category being metamorphic malware. This malware change their form between generations through several code mutation techniques thus evading detection. These mutation techniques range from garbage code insertion which adds junk code to the source program or variable renaming that renames valid variables within the program code among others [7].

Different techniques have been proposed for detecting metamorphic malware such as signature based detection as in [30], heuristic based detection as in [29], malware normalisation and similarity-based detection as in [21] among others. However such methods often fail to detect new mutant variants of malicious code. Recently, in order to detect metamorphic malware, adversarial learning has been introduced to create novel mutant variants of malware as data source which are then used to train ML models. This includes the use of Evolutionary Algorithms (EAs) [11] - population based meta-heuristic search technique such as in [3], [4], [5], [2] and [31].

In the previous adversarial learning methods used for defeating metamorphic malware as in [3] and [5], the idea was to use an EA to create mutant variants of malware which will then serve as rich data source for training ML based detectors in detecting them. However, the task of creating the adversarial samples and detecting them are considered as two separate tasks. Generative Adversarial Network (GAN) was introduced in [13] by Ian Goodfellow in 2014. GAN differs from the gradient-based adversarial generation method in that rather than training an ML model to create malicious code against a pre-trained detector, both models learn through competition with each other. The GAN framework was designed to build generative models that use an adversarial approach that trains two distinct models concurrently. This has been used previously for malware detection by authors such as [18] and [24].

In this paper, an Evolutionary based GAN-Inspired approach is proposed to simultaneously create novel mutant variants of malware by an EA based generator guided by three fitness functions - evasiveness of the variants, behavioural and structural similarity of the mutants to their parent malware. Then, an ML based detector detects the created mutant variants of malware.

Three research questions are addressed in this paper:

(1) To what extent can an Evolutionary based GAN-Inspired approach be used in generating novel mutant variants of malware?
(2) What ML detectors yield the best accuracy in detecting the novel mutant variants of malware?
(3) How well can the generated variants evade known public AV detectors?

This paper adds to the body of knowledge in this domain by proposing an alternative method of generating adversarial examples of mobile metamorphic malware. The method is tested thoroughly with respect to the quality of the mutant variants of malware created i.e., evasiveness and diversity. It also studies the best ML models that detect the new adversarial samples created and their classification performance.

The rest of the paper is structured as follows. In Section two, we present a background of the work and review related works. Section three presents our methodology. We explain our experimental design in Section four. Our results are discussed and analysed in Section five. Section six concludes the paper and presents areas of future research work.

2 BACKGROUND

A number of research works have been proposed to defeat adversarial attacks that often evade detection by standard Antivirus engines. This includes systems such as ADAM [33], DroidChameleon [27] among others designed to create adversarial examples using various code obfuscation techniques.

Among the many techniques used for generating adversarial samples, includes the use of Evolutionary Algorithms (EAs). This includes the use of Genetic Programming (GP) as used by the authors in [32] to create adversarial pdf malware. Furthermore, the authors in [1] also employed GP in creating evasive mobile based malware. The authors in [3] used a standard EA in the creation of adversarial samples guided by a fitness function that evolved for both evasive as well as structurally and behaviourally dissimilar mutant variants of malware. They extended their work in [4] by using a Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) algorithm to generate a large set of novel, malicious mutants that were diverse with respect to their behavioural and structural similarity to the original mutant.

Furthermore, GAN has been used for malware analysis. For instance, the authors in [18] proposed MalGAN, which comprised of a generator that produced a suite of adversarial samples against current detectors which also had the ability to go undetected by future malware detectors. It trained the generator to trick the detector in a hybrid system which consisted of a stacked generator as well as a substitute detector. It was also used in [24] which built on the work of [18] in proposing a graph based adversarial malware generator using Neural Networks which extracted malicious features, ordered them and used them in graphing and encoding of malicious code to create a model of the behavioural pattern. The work of [31], proposed and evaluated a Generative Neural Network (DCGAN) in improving the detection of metamorphic malware based on behaviour profiling.

In this paper, we propose an EA based GAN for the generation and detection of novel mutant variants of malware whose samples are analysed for their adversarial accuracy and a deep learning based detector detects the created mutant variants of malware. We also compare four ML models to see which is better at correctly classifying the mutant variants of malware. As far as we are aware, this is the first time that an Evolutionary based GAN-Inspired approach has been used in the exploration of the search space of metamorphic malware.

3 METHODOLOGY

In this section, we describe the GAN-Inspired network architecture - the single objective EA used as generator to create a large, evasive and diverse archive of malware mutants, and the deep learning model - LSTM used as detector for detecting the generated mutants.

3.1 Network Architecture

The detector model is initially trained and defined using benign samples from different application categories - education, music, tools, health and maps, as well as malicious samples. After this, the generator (single objective EA) is executed first using the original malware. Then, the EA based generator generates a sample, using the parameter setting defined in Section 4. The features of this sample are then extracted which are sequential features (the feature vector consists of the time-ordered list of the sample's system calls). If the sample is guessed correctly - detected as malicious by the detector, this generated sample is then passed back into the generator (replacing the original parent malware). Otherwise, if it does not guess the sample correctly - deems it benign, the process is complete, e.g., the detector is unable to detect the generated mutated sample as malicious. This process is then executed for a maximum number of times defined at run-time. The overall GAN framework is given in Fig. 1.

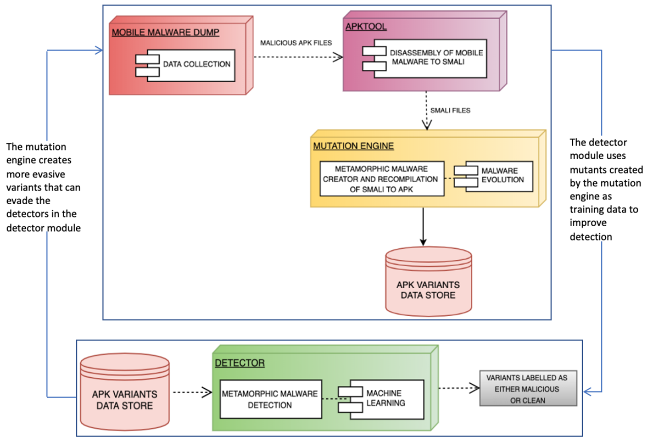

MOBILE MALWARE DUMP
DATA COLLECTION
MALICIOUS APK FILES
APKTOOL
DISASSEMBLY OF MOBILE MALWARE TO SMALI
SMALI FILES
MUTATION ENGINE
METAMORPHIC MALWARE CREATOR AND RECOMPILATION OF SMALI TO APK
MALWARE EVOLUTION
APK VARIANTS DATA STORE
The mutation engine creates more evasive variants that can evade the detectors in the detector module
The detector module uses mutants created by the mutation engine as training data to improve detection
APK VARIANTS DATA STORE
DETECTOR
METAMORPHIC MALWARE DETECTION
MACHINE LEARNING
VARIANTS LABELLED AS EITHER MALICIOUS OR CLEAN

Figure 1: The GAN framework adapted from [14]

3.1.1 *Generator - Single Objective EA:.*

The generator used by the GAN is a classical EA as in [3] that employs a single objective performance based fitness function to drive evolution with the goal of generating novel archive of mutant malware variants that evade current detection engines, and are diverse with respect to their behavioural and structural similarity. The EA is given in Algorithm 1.

The EA begins with an initial random population of solutions - malware mutants, generated through the application of a single mutation to the original malware. In creating the mutants, one of three mutation operators are selected at random and applied to an existing malware and they include:

- Garbage Code Insertion (GCI) - This inserts a piece of junk code, e.g. an invalid line number into the original program code.
- Instructional Reordering (IR)- This adds a goto statement in the original program code that jumps to a label that does nothing.
- Variable Renaming (VR) - This renames a variable with another valid variable name in the original program code.

Each of the mutants are then evaluated using a fitness function which is minimised by the algorithm - the smaller the value the more evasive the mutant and the more it is behaviourally and structurally dissimilar to the original malware. Since the original malware is in the form of an executable apk file, to generate mutants, it is necessary to reverse engineer the apk by converting it to a smali format using apktool[1]. Then, a mutation operator is applied to the smali code. Furthermore, the smali is recompile to apk to test that the created mutant is executable. The recompiled apk is then signed using apksigner[2] and aligned using zipalign[3]. The fitness of the mutant is then calculated using Equation 3.1.1 and returns a value between 0 and 1.

$$f(x) = \begin{cases} 1 & \text{if variant not executable} \\ w_0 DR(x) + w_1 BS(x) + w_2 SS(x) & \text{otherwise} \end{cases}$$

$$\text{subject to } 0 \le DR(x), BS(x), SS(x) \le 1$$

The fitness defined above is either set to 1 for non executable variants which are not taken into account or equals the weighted sum of three metrics - DR(x), BS(x) and SS(x). These are the detection rate of the new mutant, the behavioural similarity between the original malware and its mutant and the code level similarity between the original malware and its mutant respectively.

DR(x) measures the detection rate of the malware mutants. It does this by checking the mutants' ability to evade detection by a set of well known Antivirus engines. The malware mutant variants are evaluated using Virustotal[4] which consists of 63 up-to-date Antivirus engines and reports the number of its engines that flag a sample as malicious. Thus DR(x) returns the percentage of the Antivirus engines that *fail* to detect a mutant with a value of 0 denoting no engine detected the mutant while a value of 1 denotes all the engines detected the mutant.

BS(x) computes the behavioural similarity between the original malware and its mutants by carrying out their behavioural analysis. This involves utilising Strace[5] for monitoring the system calls of the original malware and its mutants as well as using Monkey runner[6] to simulate user action. This generates a log which is employed in constructing a feature vector with each element corresponding to the frequency of 251 possible system calls made by the mutant. The behavioural similarity between the original malware and its mutants is computed as the cosine similarity between their system call vectors, which returns a value between 0 and 1, where the former indicates that the original malware and the mutant share no behavioral similarity while 1 means the original malware and the mutant have equivalent behaviour.

SS(x) measures the structural similarity between the original malware and its mutant variants using both text similarity metrics (Cosine Similarity, Levenshtein Distance [17] and Fuzzy String Match [10]) and source code similarity metrics (Jplag and Sherlock Plagiarism detectors [16] as well as normalised compression distance [26]). The structural similarity is then the average of all the similarity metrics used where a value of 0 means the original malware and its mutant variants are completely dissimilar and 1 means the original malware and its mutant variants are identical.

3.1.2 *Detector - Recurrent Neural Network (RNN).*

This is a deep learning algorithm designed for learning from time series sequence data. It consists of connected layers done in such a way that they go from the output of one layer to the input of the next layer and comprise of feedback loops returning to the previous layer. An RNN learns from arrays of time series data, specifically, where temporary information is derived from the sequences and used in finding associations that exist between data and the expected network output [22].

In this work, a Long Short-Term Memory (LSTM) is used in detecting the sequential data (preprocessed malware mutants), which is a neural network particularly created for learning long term dependencies from such data. It comprises of gates that are able to hold, recover and forget information over a long time span. Traditional neurons in the hidden layer are replaced with memory blocks in LSTM with these blocks accepting inputs from the network via the input node and outputs from the output gate multiplication as seen in Fig. 2 [14]. Although deep-learning models, such as LSTM network, have demonstrated their advantage in handling time-ordered information in other problem domains, they are less explored in malware detection [23], [9], [28].

4 EXPERIMENTAL DESIGN

The original malware used in this work was selected from the Contagio Minidump[7] which consists of Android malware archived as APKs and it belongs to the Dougalek[8] family. Dougalek family of malware represent the personal information thieves that steal users' personal information from mobile phones such as account details, test messages, contacts among others. They do this by gaining unauthorised remote control of mobile phones.

The EA (generator) begins with a population size of 20. Parents are selected using tournament selection [12] with $k = 5$, for a fair

[1]APKTOOL - http://ibotpeaches.github.io/Apktool
[2]APKSIGNER - https://developer.android.com/studio/command-line/apksigner
[3]ZIPALIGN - https://developer.android.com/studio/command-line/zipalign
[4]Virustotal - https://developers.virustotal.com/reference#getting-started
[5]Strace - https://linux.die.net/man/1/strace
[6]Monkeyrunner - https://developer.android.com/studio/test/monkey
[7]Contagio Minidump - http://contagiominidump.blogspot.com/2015/01/android-hideicon-malware-samples.html
[8]Dougalek - https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/androidosdougalek.a

Algorithm 1: GAN-Inspired Evolutionary Algorithm

```
1:  define and train detector model
2:  while $max\_iterations$ not reached do
3:      initialize population $P$ of size $n$.
4:      assign fitness $f(x)$ to each mutant $x \in P$
5:      while $max\_iterations$ not reached do
6:          $R \leftarrow$ randomly select $k$ variants from $P$
7:          $x_{best} \leftarrow \text{argmin}\{f(x), x \in R\}$
8:          $mut\_type \leftarrow$ select a mutation operator at random with
            uniform probability
9:          $x_{new} \leftarrow mutate(mut\_type, x_{best})$
10:         $fit_{new} \leftarrow f(x_{new})$
11:         $x_{worst} \leftarrow \text{argmax}\{f(x), x \in P\}$
12:         $fit_{worst} \leftarrow f(x_{worst})$
13:         if $fit_{new} < fit_{worst}$ then
14:             replace $x_{worst}$ in $P$ with $x_{new}$
15:         end if
16:     end while
17:     $x_{bestpop} \leftarrow \text{argmax}\{f(x), x \in P\}$,
18:     return $x_{bestpop}$
19:     if $x_{bestpop}$ is detected as malicious then
20:         add $x_{bestpop}$ to P
21:     else
22:         break
23:     end if
24: end while
```

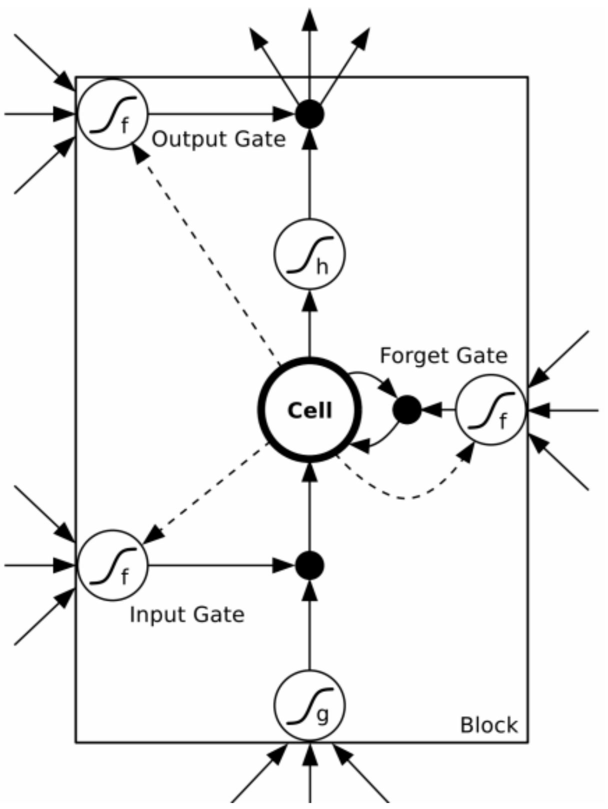


Figure 2: An example LSTM memory block where the connections with weights proceeding from the cells to the gates are illustrated using dashes and the black circles are multiplications [14]

level of selection pressure. The EA uses uniform mutation with a mutation rate of 1; this value is chosen to ensure that mutation always occurs, given that this is the only variation operator used, as crossover is not used in the experiments. The crossover operator is not used, as it leads to the generation of non-executable variants. The best of the k selected parents is mutated by adding either garbage codes, reordering its variables or distorting the program code's control flow through the insertion of a goto statement that jumps to a label that does nothing as described in Section 3.1.1. The EA is then run 10 times for 100 iterations, and the best mutant generated in each of the 10 runs is recorded. The EA parameter settings are summarised in Table 1.

The LSTM (detector) and its hyper-parameters were empirically tuned. Due to its documented success in terms of its accuracy and computational power, "Adam" optimiser [20] was used. Batch sizes between 10 and 500 were employed, and we experimented using either 1 or 2 layers of LSTM. Following this, we chose LSTM with 2 layers: each layer has 128 neurons. The binary cross entropy function was employed as the loss function (this function was chosen as our classification is binary). Furthermore, since our problem is a classification problem, a Dense output layer was used consisting of one neuron with a sigmoid activation function. We employed a batch size of 50 so as to space out the updates of weight. The LSTM was initially trained on a dataset that comprised of 30 benign samples (sourced from Google play store[10] and downloaded using Apkdownloader[11]) and 30 malicious samples (selected from Contagio Minidump). An average training loss of 0.69 was gotten, which although is high, the focus of the paper is on the analysis of the adversarial accuracy of the generator in generating an archive of malware samples. However, with more experimental work in tuning the hyper-parameters and potentially the use of other models, the result could be improved.

As a separate task done for evaluation purposes, four ML models are used for evaluating the evasiveness of the generated mutant variants of malware and they are - k-Nearest Neighbors algorithm (kNN), Support Vector Machine (SVM), Decision Tree (DT), and Multi-Layer Perceptron (MLP) and they are briefly explained.

- **k-Nearest Neighbour (kNN)**: This algorithm seeks to identify instances in the training data in an N-dimensional feature

[10]Google Play - https://play.google.com/store?hl=en
[11]Apkdownloader -https://apps.evozi.com/apk-downloader/

space that are nearest to the instance to be classified. It assumes that if an instance is close together in the feature space with the instance to be classified, then it is likely to be similar and have the same class as the instance to be classified. Although, it is one of the simplest ML models to implement, it is highly sensitive to the occurrence of parameters that are irrelevant [25].

- **Support Vector Machine (SVM)**: SVM has as its goal, finding a hyper plane in an N-dimensional feature space which classifies data points uniquely. The hyper plane identified is ideally one that maximises the distance between data points of the classes i.e., has the maximum margin. This leads to increased confidence in future classification attempts. SVM needs minimal computational power yet producing high performance [19].
- **Decision Trees (DT)**: A binary decision tree is derived from the split of a node into two child nodes severally, beginning with the root node which consists of the whole learning sample [6].
- **Multilayer Perceptron (MLP)**: MLP is a deep artificial neural network with several layers consisting of at least an input layer, a hidden layer and an output layer. The input layer is often employed for input reception, the hidden layer is the computation engine and the output layer is utilised for decision making or predictive analysis [15].

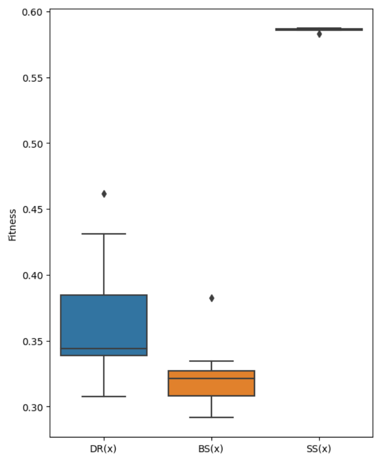
The ML models were trained on a dataset containing two classes, benign (0) and malware (1). Both of these classes comprised of 30 samples. In addition to this, a further 20 benign samples were collected. The benign samples were sourced from Google play store and also downloaded using Apkdownloader. The benign samples were selected from various categories such as entertainment, gaming, communication among others and were all individually executed to ensure that they worked correctly. For the malware samples, 30 samples were selected from Contagio Minidump.

5 RESULTS AND ANALYSIS

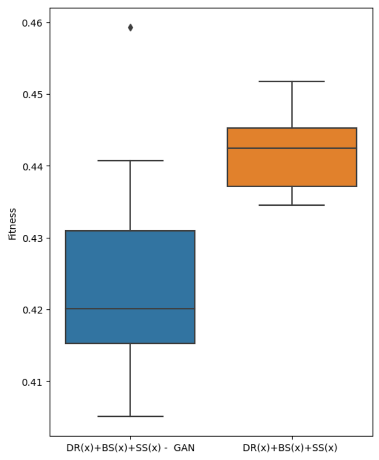
The experiments then were conducted and results are analysed in the subsections below to answer our research questions.

5.1 To what extent can an Evolutionary based GAN-Inspired approach be used in generating novel mutant variants of malware?

In answering the first research question, we run the Evolutionary based GAN in order to create an archive of malware mutants and measure their adversary accuracy (average percentage of Antivirus detection engines that detect the generated samples) as well as evaluate the fitness of the best mutants recorded as described in Section 4. Recall that the EA evolves for variants that are able to evade detection by more detectors than the original malware and are also behaviourally and structurally dissimilar to the original malware. As the fitness is the equally weighted sum of functions - DR(x), BS(x) and SS(x) (weight is set to 1), we analyse the best mutants with respect to these metrics.



(a) Analysis of fitness in terms of Detection Rate ($DR(x)$, Behavioral Similarity ($BS(x)$) and Structural Similarity ($SS(x)$) of the best mutants



(b) Boxplot of best fitness of the EA where the fitness is a weighted combination of $DR(x)$, $BS(x)$ and $SS(x)$ as given in Eq. 3.1.1 in Section 3 for GAN-Inspired EA (DR(x)+BS(x)+SS(x)-GAN) and EA in [3] (DR(x)+BS(x)+SS(x))

Figure 3: Boxplots of Best Fitness for Dougalek family

In terms of the average adversarial accuracy of the malware mutants, this was 0.366 i.e., on average only 36.6% of the Antivirus engines detected the novel mutants. As the original malware had an adversarial accuracy of 0.597 - 59.7% of the Antivirus engines detected the original malware, this shows that we are able to create more evasive mutants than the original malware. Furthermore, we see from Fig. 3(a), that we were able to create mutant variants of malware in which the best one has a detection rate of 0.3 and all the ten best mutants recorded have values less than 0.47. In addition, it can be seen that we were able to create variants that are on average 32% behaviourally ($BS(x)$) similar to the original malware and 59% structurally ($SS(x)$) similar to the original malware, indicating behavioural and structural diversity.

Table 1: Evolutionary based Parameter Settings

Parameters	Settings
Selection	Tournament
Population size	20
Iterations	100
Mutation rate	1

When all the functions in the fitness function are combined as given in Equation 3.1.1, we were able to create a diverse set of mutants that are executable, evasive and behaviourally and structurally dissimilar to the original malware with a median weighted fitness value of 0.42 compared to the original malware which would have a weighted fitness value of 0.8657 (with a value 0.597 for DR(x), 1 for BS(x) and SS(x)), as seen in Fig.3(b). In Fig.3(b), we also show that the evolutionary based GAN-Inspired approach with a median weighted fitness value of 0.42 outperforms our previous work in [3] which uses a standard EA (a median weighted fitness value of 0.44) in creating mutant variants of malware.

5.2 What ML detectors yield the best accuracy in detecting the novel mutant variants of malware

To answer the second research question, we compared the performance of the four ML algorithms described in Section 4, trained using the sequential features of the samples - feature vector consists of the time-ordered list of the sample's system calls with the training data described in Section 4, to see which one produces the best performance in terms of classification accuracy, precision, recall, F1 Score and ROC AUC score in detecting the new malware mutants. From Table 2, we see that the best performance is achieved by the kNN model with values of 0.77 - accuracy, 0.61 - precision, 1 - recall, 0.76 - F1 score and 0.82 - ROC AUC and the worst performance by both DT and MLP with values of 0.6 - accuracy, 0.48 - precision, 1 - recall, 0.65 - F1 score and 0.68 - ROC AUC. It can be seen that the ML models are susceptible to the new malware mutants achieving an accuracy between 60%-77%.

5.3 How well can the generated variants evade known public AV detectors?

In order to gain more insight into which engines are most vulnerable to potential mutated variants of the original malware, we determine the percentage of new variants evolved using the Evolutionary based GAN-Inspired approach that a detector fails to recognise. Only the engines which recognised the original parent malware are considered in this analysis so as to understand which engines are susceptible to potential mutants and which remain capable of detecting the malware. The results are shown in Fig. 4.

It can be seen from Fig. 4, that 13 of the 37 engines that detected the original parent malware also recognise *all* of the mutants evolved by the Evolutionary based GAN-Inspired approach. Examples include Kaspersky, Avast and AVG. Eleven of the engines on the other hand, failed to recognise 100% of the newly generated mutants — AegisLab, AVware and SophosAV among others.

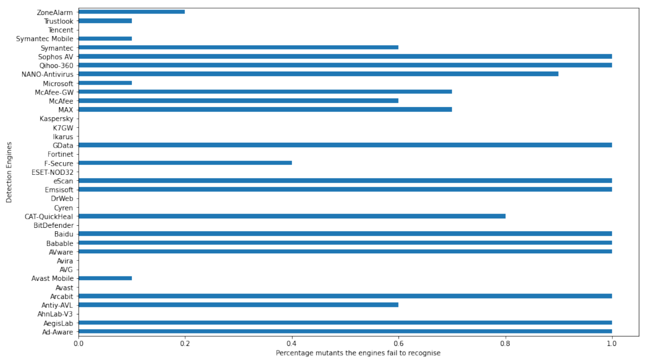


Figure 4: Percentage of the mutants evolved from the Evolutionary based GAN-Inspired approach that a given detection engine failed to recognise

6 CONCLUSION

It has been established that metamorphic malware transform their program codes over generations thus representing a difficult class of malware for detectors to detect, particularly those trained on static datasets. In this work, we have shown that an Evolutionary based GAN-Inspired approach is capable of creating an archive of evasive and diverse set of mutants which are able to evade about 63% of well known AV engines as compared to the parent malware from which they were created which was only able to evade 40% of the AV engines. We also show that the ML models we tested on are susceptible to the new mutants achieving an accuracy between 60%-77%.

For future work, we plan to combine the GAN model with a Quality-Diversity EA to see if this improves the performance of the mutants created and if this improves their detection. More families of malware will be tested to see if the proposed approach generalises to other classes of malware.

REFERENCES

[1] Emre Aydogan and Sevil Sen. 2015. Automatic Generation of Mobile Malwares Using Genetic Programming. In *Applications of Evolutionary Computation*, Antonio M. Mora and Giovanni Squillero (Eds.). Springer International Publishing, Cham, 745–756.
[2] Kehinde O. Babaagba and Mayowa Ayodele. 2023. Evolutionary Based Transfer Learning Approach to Improving Classification of Metamorphic Malware. In *Applications of Evolutionary Computation*, João Correia, Stephen Smith, and Raneem Qaddoura (Eds.). Springer Nature Switzerland, Cham, 161–176.
[3] Kehinde O. Babaagba, Zhiyuan Tan, and Emma Hart. 2019. Nowhere Metamorphic Malware Can Hide - A Biological Evolution Inspired Detection Scheme. In *Dependability in Sensor, Cloud, and Big Data Systems and Applications*, Guojun Wang, Md Zakirul Alam Bhuiyan, Sabrina De Capitani di Vimercati, and Yizhi Ren (Eds.). Springer Singapore, Singapore, 369–382.
[4] Kehinde O. Babaagba, Zhiyuan Tan, and Emma Hart. 2020. Automatic Generation of Adversarial Metamorphic Malware Using MAP-Elites. In *Applications*

Table 2: Comparing performance obtained on the test sets for the ML models

ML Models	Accuracy	Precision	Recall	F1 Score	ROC AUC
kNN	**0.77**	**0.61**	1	**0.76**	**0.82**
SVM	0.7	0.56	0.9	0.69	0.74
DT	0.6	0.48	1	0.65	0.68
MLP	0.6	0.48	1	0.65	0.68

of Evolutionary Computation, Pedro A. Castillo, Juan Luis Jiménez Laredo, and Francisco Fernández de Vega (Eds.). Springer International Publishing, Cham, 117–132.

[5] Kehinde O. Babaagba, Zhiyuan Tan, and Emma Hart. 2020. Improving Classification of Metamorphic Malware by Augmenting Training Data with a Diverse Set of Evolved Mutant Samples. In *2020 IEEE Congress on Evolutionary Computation (CEC)*. 1–7.

[6] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. 1984. *Classification and regression trees.* Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA.

[7] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. 2007. Code normalization for self-mutating malware. *IEEE Security and Privacy* 5, 2 (2007), 46–54.

[8] CrowdStrike. [n. d.]. 2022 Global Threat Report. https://www.crowdstrike.com/global-threat-report/

[9] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. 2013. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 3422–3426.

[10] A. Dhakal, A. Poudel, S. Pandey, S. Gaire, and H. P. Baral. 2018. Exploring Deep Learning in Semantic Question Matching. In *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS '18)*. 86–91.

[11] Agoston Endre Eiben and Jim E. Smith. 2003. What is an Evolutionary Algorithm? In *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 15–35. arXiv:arXiv:1011.1669v3

[12] Yongsheng Fang and Jun Li. 2010. A Review of Tournament Selection in Genetic Programming. In *Advances in Computation and Intelligence*, Zhihua Cai, Chengyu Hu, Zhuo Kang, and Yong Liu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 181–192.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680.

[14] Alex Graves. 2012. *Supervised Sequence Labelling*. Springer Berlin Heidelberg, Berlin, Heidelberg, 5–13.

[15] PRAMOD GUPTA and NARESH K. SINHA. 2000. CHAPTER 14 - Neural Networks for Identification of Nonlinear Systems: An Overview. In *Soft Computing and Intelligent Systems*, NARESH K. SINHA and MADAN M. GUPTA (Eds.). Academic Press, San Diego, 337–356.

[16] Daniel Heres. 2017. *Source Code Plagiarism Detection using Machine Learning*. Ph. D. Dissertation. Utrecht University.

[17] Wael H.Gomaa and Aly A. Fahmy. 2013. A Survey of Text Similarity Approaches. *International Journal of Computer Applications* (2013).

[18] Weiwei Hu and Ying Tan. 2017. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *CoRR* abs/1702.05983 (2017). arXiv:1702.05983 http://arxiv.org/abs/1702.05983

[19] V. Kecman. 2005. *Support Vector Machines – An Introduction*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–47.

[20] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[21] Jared Lee, Thomas H Austin, and Mark Stamp. 2015. Compression-based Analysis of Metamorphic Malware. *International Journal of Security and Networks* 10, 2 (jul 2015), 124–136.

[22] Zachary Chase Lipton. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *ArXiv* abs/1506.00019 (2015).

[23] R. Lu. 2019. Malware Detection with LSTM using Opcode Language. *arXiv:1906.04593* (2019). arXiv:arXiv:1906.04593

[24] Ross A. J. McLaren, Kehinde Oluwatoyin Babaagba, and Zhiyuan Tan. 2023. A Generative Adversarial Network Based Approach to Malware Generation Based on Behavioural Graphs. In *Machine Learning, Optimization, and Data Science*, Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Gabriele La Malfa, Panos Pardalos, Giuseppe Di Fatta, Giovanni Giuffrida, and Renato Umeton (Eds.). Springer Nature Switzerland, Cham, 32–46.

[25] Robert Nisbet, Gary Miner, and Ken Yale. 2018. Chapter 9 - Classification. In *Handbook of Statistical Analysis and Data Mining Applications (Second Edition)* (2 ed.), Robert Nisbet, Gary Miner, and Ken Yale (Eds.). Academic Press, Boston, 169 – 186.

[26] Chaiyong Ragkhitwetsagul, Jens Krinke, and David Clark. 2018. A comparison of code similarity analysers. *Empirical Software Engineering* 23, 4 (2018), 2464–2519.

[27] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. 2013. DroidChameleon: Evaluating Android Anti-malware Against Transformation Attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security* (Hangzhou, China) *(ASIA CCS '13)*. ACM, New York, NY, USA, 329–334.

[28] J. Saxe and K. Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. 11–20.

[29] Annie H Toderici and Mark Stamp. 2013. Chi-squared Distance and Metamorphic Virus Detection. *J. Comput. Virol.* 9, 1 (feb 2013), 1–14.

[30] N P Tran and M Lee. 2013. High performance string matching for security applications. In *International Conference on ICT for Smart Society*. 1–5.

[31] Leigh Turnbull, Zhiyuan Tan, and Kehinde O. Babaagba. 2022. A Generative Neural Network for Enhancing Android Metamorphic Malware Detection based on Behaviour Profiling. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*. 1–9.

[32] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically Evading Classifiers: A Case Study on PDF Malware Classifier. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA*. The Internet Society.

[33] Min Zheng, Patrick P. C. Lee, and John C. S. Lui. 2013. ADAM: An Automatic and Extensible Platform to Stress Test Android Anti-virus Systems. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Ulrich Flegel, Evangelos Markatos, and William Robertson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 82–101.