Improving Classification of Metamorphic Malware by Augmenting Training Data with a Diverse Set of Evolved Mutant Samples

1st Kehinde O. Babaagba
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
K.Babaagba@napier.ac.uk

2nd Zhiyuan Tan
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
Z.Tan@napier.ac.uk

3rd Emma Hart
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
E.Hart@napier.ac.uk

***Abstract*—Detecting metamorphic malware provides a challenge to machine-learning models as trained models might not generalise to future mutant variants of the malware. To address this, we explore whether machine-learning models can be improved by augmenting training data-sets with samples of potential variants. These variants are generated using an evolutionary algorithm that evolves a behaviourally diverse set of mutants, optimised to avoid detection by a large set of existing detection-engines. Using features calculated from the behavioural trace of a sample as input, we evaluate the ability of five machine-learning methods to detect the new variants, show that the detection rate is considerably improved by including the new samples as training data, and that the classifiers still generalise over a range of malware. We then repeat this experiment using a sequence-based deep-learning method as the classifier, which is shown to out-perform the feature-based classifiers.**

***Index Terms*—Machine-learning, Evolutionary computing, Malware and Computer security**

I. Introduction

Metamorphic malware represents a dangerous class of malware which changes its form stochastically over generations. They often use varying obfuscation methods to enable them to go undetected by antivirus engines and other detectors. Obfuscation approaches include techniques such as garbage code insertion (the addition of junk code into the malware's code), instruction substitution (which replaces instructions with valid ones that do not change the code's function) and instruction-reordering. These mutation methods make metamorphic malware hard to detect by existing detection-engines.

Machine-learning (ML) methods are now common in malware detection [1], trained using samples of known malware. However, this provides an attack surface for adversaries to launch attacks, by creating malware that is dissimilar from the training data and designed to avoid detection. For metamorphic malware which can continually change its form, this is even more of an issue. One approach to this is to use an adversarial learning method [2] in which a feedback loop is created in a system that (1) continually searches for malware samples that are misclassified by the machine-learning model used for detection, and (2) retrains the model based on these samples. Here we propose a modified version of this method which uses an Evolutionary Algorithm (EA) first to evolve a diverse set of malware-mutants that are optimised with respect to their ability to evade detection by a large set of detection engines, while retaining their maliciousness. The evolution of these mutants has been previously described in recent papers [3], [4]; the former uses a classical EA to evolve an optimised mutant, whereas the latter uses a quality-diversity algorithm MAP-Elites to return a set of mutants which are diverse with respect to their behavioural and structural similarity to the original malware. In this paper, we investigate whether these mutants can be used to train better machine-learning models capable of detecting other potential unseen mutants.

This paper seeks to answer the following research questions:

- To what extent are models trained only on existing samples of known malware capable of detecting potential mutants?
- Can these models be improved by augmenting the training set with evolved samples, and if so, what is the most appropriate method of combining the evolved samples with existing samples?
- Do models trained on evolved data representing future mutants retain their ability to recognise existing known malware?

The questions are answered by conducting a study using two types of ML models, namely non-sequential and sequential models. We compare five classical ML models with Long-Short-Term-Memory (LSTM) [5], a deep-learning method. The former methods rely on features describing the frequency of system-calls made by a malware, while the latter uses the ordered sequence of system calls directly as input.

The contributions of this paper can be summarised as follows. Firstly, we provide evidence to show that models trained only on existing samples of malicious code are vulnerable, failing to detect mutant samples. Secondly, we demonstrate that the samples evolved using Evolutionary Algorithms provide a rich-source of training data that improves the ability of classifiers to detect new mutants while retaining their ability to generalise across existing samples. Finally, we show that sequential models yield a better performance than non-sequential models. We also make the feature vector data available [6], so the readers can make use of the data.

We structure the rest of the paper as follows. In Section II, we present the background of this research and review related work. Section III describes our research methodology. Experiments and result analysis are discussed in Section IV. In Section V, we conclude the paper and suggest areas of future research.

II. Related Work

In spite of the fact that ML approaches have been shown to offer cutting edge solutions for automatically detecting malware as seen in [7], [8], [9] among others, the recent research focus on adversarial sample generation has also demonstrated how susceptible ML models are. For instance, 84.24% of adversarial examples created in [10] were misclassified on MetaMine (a well know deep learning API available online). Other examples of susceptible ML models are described in [11], [12], [13].

Recent research in the field of Evolutionary Computing suggests that the field offers useful methods in creating modified versions of malware optimised to evade detection. An EA is described in [14] that evolved samples that evaded a number of detection engines. Our recent work advanced this, using a multi-objective fitness function in [3] which optimised for samples that were evasive but also structurally and behaviourally diverse compared to the original malware, and using a quality-diversity algorithm in [4] to enhance the range of diversity and return multiple samples in one run. Moreover, an adversarial approach to generating pdf malware was described in [15], which used a genetic programming approach.

Although many studies have demonstrated that machine learning models are susceptible to evasive attacks [10], fewer provide solutions that help secure machine-learning models from these adversarial samples. The work of [16] presented a wrapper based feature-selection approach that is adversary-aware to enhance the security of classifiers against adversarial attacks. Their model takes into consideration the data perturbations employed by the adversaries with experimental results demonstrating the efficiency of their technique when tested on samples from spam and malware detection domain.

Here, we focus on a two-phase approach to training better malware detectors: in the first phase, an EA is used to evolve a diverse set of mutants that are both malicious and evasive. This was described in previous work [3], [4]. The second-phase is proposed in this paper, in which the evolved data is shown to be beneficial in training improved ML detection models.

III. Methodology

This section briefly discusses the algorithms employed in the generation of the training samples, which have been documented in [3], [4]. Then, the data-collection and pre-processing steps taken to obtain both normal and malware samples are described. Last but not least, we introduce the machine-learning algorithms (sequence and non-sequence based) employed in the detection process.

A. Generation of mutant variants of existing malware

The adversarial samples used as training data and test data in this work were created using two evolutionary algorithms as described in [3] and [4]. The first algorithm used a steady state mutation-only EA [3] to generate variants, which were evaluated according to one of three fitness functions: minimises the behavioral similarity between a variant and the original malware; minimises the structural similarity between a variant and the original malware; minimises the detection rate with respect to 63 detection-engines. Only evolved samples that are both evasive and malicious are retained. The aforementioned methods also ensure that all samples are distinct (more details can be found in [3] and [4]).

In order to test that the evolved mutants retain their malicious nature, we use Droidbox[1] which is an Android based sandbox designed for analysing Android files dynamically. This involves monitoring the sample's behavior while it executes. Droidbox runs Android files submitted to it and logs key information of the sample such as connections opened, API call traces, file related activities (such as file deletion and creation) among others.

The second EA uses a quality diversity algorithm — MAP-Elites — to produce a set of adversarial samples that are diverse with respect to their structural and behavioural similarity to the parent malware, denoted by s and b respectively. For each descriptor $< s, b >$, the algorithm optimises the evasiveness of the variant associated with the descriptor. This method was shown to produce a larger variety of mutants, and to maximise the evasiveness.

B. Data collection

In this work, our ultimate focus is on metamorphic malware. However, since it is not easy to collect sufficient samples and their mutants, we use Android malware from well-known families as a proxy to generate mutant samples which represent potential future variants, in order to prove the concept that augmenting training data with a diverse set of evolved mutant samples improves classification of metamorphic malware is a viable method.

The Android samples used are archived as APK files. Benign and malicious data are collected from various sources as shown in Table I. The adversarial samples generated by the EA described in Section III-A are mutant variants of the malware samples collected from Contagio Minidump[2] and Malgenome[3] as well as malicious samples from three different

[1]Droidbox - https://www.honeynet.org/taxonomy/term/191

[2]Contagio Minidump - http://contagiominidump.blogspot.com/2015/01/android-hideicon-malware-samples.html

[3]Malgenome - http://www.malgenomeproject.org/

TABLE I
DATA-SETS UTILISED

Samples	Acronym	Data Source	Total number of Samples	Description
Benign Samples	B	Google Playstore[7]	50	Comprised of games, beauty, communication and entertainment applications
Malicious Samples from web	M_w	Contagio Minidump[2]	50	Comprised of Dougalek family[4], Droidkungfu[6] and other families of malware
Evolved malware	EM	Variants of malware samples from Contagio Minidump[2] and Malgenome dump[3]	50	Samples generated using MAP-Elites and Steady state EA (This set of samples contains three families; 21 from Dougalek[4], 15 from GGtracker[5] and 14 from Droidkungfu[6])
Evolved malware - unseen set for testing	EM_u	Variants of malware samples from Contagio Minidump[2] and Malgenome dump[3]	30	Samples generated using MAP-Elites and Steady state EA (This set of samples contains two families; 21 from Dougalek[4], and 9 from Droidkungfu[6])

families, namely Dougalek[4], GGtracker[5] and Droidkungfu[6]. The samples selected from the dump are chosen on the basis of their malicious payload and they fall into four categories, namely those that either escalate privilege, try to gain remote control of phones, those that result in financial charges or those that steal personal information of users.

Additional malicious samples M_w collected from the web are also from the Contagio Minidump which consists of families such as Dougalek, DroidKungfu among other malicious software. The clean samples B on the other hand are sourced from Google play store[7] and downloaded using Apkdownloader[8]. The benign samples are selected from various categories such as entertainment, gaming, communication among others.

A further data-set EM of malicious samples is formed from the evolved mutants generated by the methods previous described, containing mutants generated from 3 malware families. A final dataset EM_u consists of an additional 30 generated mutants and is held out as an unseen set for testing. This dataset only contains mutants from those malware families present in the dataset collected from the web.

C. Data Processing

We use information obtained from the behavioural trace collected from running a sample as an input to the classifiers. We employ Strace[9] to monitor the behavior of the malware and Strace keeps a log of the system calls made by each malware. By running the malware's main activity using MonkeyRunner[10], we are able to simulate user interaction with the malware. We then generate both sequential and non-sequential features from this log as follows:

1) Non-sequential data processing: For the non-sequential data processing, the logs generated from Strace are converted to a fixed sized vector with each element corresponding to the observed frequency of a potential system-call employed. 251 systems calls are considered.

2) Sequential feature generation: The sequential features are also extracted from the Strace log. However, the feature vector consists of the time-ordered list of system-calls extracted from the log.

D. Machine-Learning

We select five non-sequential models to test as predictors, based on their prevalence in the metamorphic malware detection literature. They are Logistic Regression, Support Vector Machine, Naïve Bayes, Decision Trees, and K-Nearest Neighbour.

For comparison with the non-sequential models, LSTM is chosen for detection based on sequences of system calls. Although deep-learning models, such as LSTM network, have shown their superiority in handling time-ordered information in other problem domains, they are less explored in malware detection [17], [18], [19].

The algorithms employed are briefly explained below.

[4]Dougalek - https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/androidosdougalek.a

[5]GGtracker - https://www.f-secure.com/v-descs/trojan_android_ggtracker.shtml

[6]Droidkungfu - https://www.f-secure.com/v-descs/trojan_android_droidkungfu_c.shtml

[7]Google Play - https://play.google.com/store?hl=en

[8]Apkdownloader -https://apps.evozi.com/apk-downloader/

[9]Strace - https://linux.die.net/man/1/strace

[10]Monkeyrunner - https://developer.android.com/studio/test/monkey

1) Logistic Regression (LR): This algorithm is often used for binary data and when the target variable is categorical. For instance to predict whether a software code is benign (0) or malicious (1). A logit transformation is employed to force the Y value to take on varying values between 0 and 1. The probability $P = 1/(1 + e - (c + bX))$ is first computed after that X is then linearly associated to $log_n P/(1 - P)$ [20].

2) Support Vector Machine (SVM): SVM has as its goal identifying a hyper plane in a feature space of N dimensions that uniquely classifies data points. The hyper plane discovered is ideally one that maximises the distance between data points of the classes i.e. has the maximum margin. This results in greater confidence in future classification attempts. SVM requires little computational power yet producing high accuracies [21].

3) Naïve Bayes (NB): This is a probability based machine learning model employed in classification tasks. It follows the Bayesian theorem summarised in (1) below.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

From the theorem, we note that provided that B has occurred, we can then compute the probability of A occurring. This implies that while A is the hypothesis, B is the evidence. Naïve Bayes assumes that features are independent and so the absence of a feature has no effect on the other [22].

4) Decision Trees: A binary decision tree results from the split of a node into two child nodes severally, starting with the root node which comprises of the entire learning sample [23].

5) K Nearest Neighbour (KNN): This algorithm seeks to find the instances in the training data in a feature space of N dimensions that are closest 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 [24].

6) Recurrent Neural Network (RNN): This is a type of machine-learning algorithm that is built for learning from time series sequence data. An RNN is a deep learning algorithm, which comprises of layers connected in such a way that they go from the output of one layer to the input of the next layer and have feedback loops returning to the preceding layer. An RNN learns from sequences of time series data particularly, where temporary information is deduced from the sequences and employed in finding associations that exist between data and the expected network output [25].

In order to learn from the sequential data obtained from the samples described in Section III-C, we use LSTM, a neural network specially designed for learning long term dependencies from such data. It consists of gates that are able to hold, recover and forget information over a long time span. Traditional neurons in the hidden layer are substituted 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. 1 [5].

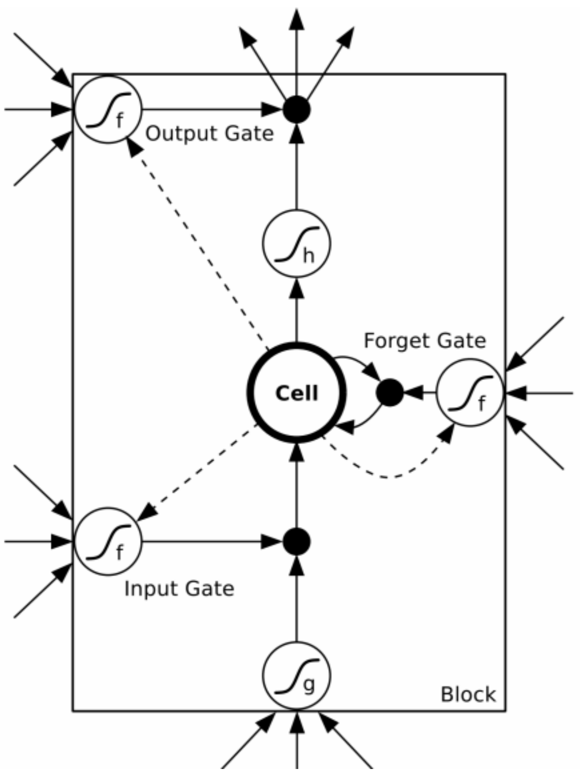

Fig. 1. An example LSTM memory block where the connections with weights proceeding from the cells to the gates are illustrated using dashes and the black circles are multiplications [5]

IV. Experiments and Discussion

All classical non-sequential ML models were implemented using Scikit-learn libraries for Python, with the Keras library[10] implementation of LSTM in python for the sequence based machine learning. 10-fold cross validation was used to train and validate models. An unseen test-set containing evolved mutant malware was used in testing.

The LSTM and its hyper-parameters were empirically tuned. As a result of its documented success in terms of its accuracy and computational power, "Adam" optimiser [26] was employed. We used batch sizes between 10 and 500, and 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 used as the loss function (this function was chosen as our classification is binary). Moreover, as our problem is a classification problem, we employ a Dense output layer comprising 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 model was fitted using just three epochs as it speedily over-fitted the problem.

[10]Keras - https://github.com/fchollet/keras

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

A. Are models trained on existing samples of malware capable of detecting potential mutants?

To answer the first research question, we trained the sequential and non-sequential machine-learning models on a data-set that is comprised of the benign samples (dataset B, Table I) and the malicious samples collected from the web (dataset M_w, Table I). Table II shows the results from the cross-validation experiment, indicating that the trained models perform well on this data-set which only contains samples of known malware from the web. However, when the best trained model (Naïve Bayes) is tested on the unseen set of evolved mutants EM_u (top line of Table V), the model performs very poorly with an accuracy of 0.4.

TABLE II
10-FOLD CROSS VALIDATION WITH MODEL THAT USES SAMPLES OF B AND M_w AS TRAINING DATA

Model	10-fold Mean Accuracy (std)	Validation Accuracy
LR	0.889881 (0.115217)	0.9
KNN	0.910317 (0.103585)	0.95
CART	0.912103 (0.080542)	1
NB	0.937103 (0.084450)	0.95
SVM	0.899603 (0.093919)	1

B. Can the models be improved by augmenting training data with evolved mutants?

In order to determine whether adding evolved mutants to the training set results in better models we conducted two further experiments:

1) The training set consists of the benign samples B and 50 evolved mutants EM
2) The training set consists of the benign samples B, 25 evolved mutants randomly selected from EM and 25 samples randomly selected from M_w (the samples from the web)

The results are shown in Tables III and IV respectively. We note that both approaches achieved good accuracy on the validation sets, with once again the Naïve Bayes method performing well. The trained models were then tested on the unseen set of evolved mutants EM_u, with results shown in the second two lines of Table V, obtained using the Naïve Bayes model. Compared to the results from the same model trained with the malicious samples from the web, a clear difference is observed, with both of the trained models resulting in accuracy of over 80%. This clearly indicates that the evolved mutants provide useful additional data by which to train a model to recognise future variants of metamorphic malware.

TABLE III
10-FOLD CROSS VALIDATION WITH MODEL THAT USES SAMPLES OF B AND EM AS TRAINING DATA

Model	10-fold Mean Accuracy (std)	Validation Accuracy
LR	0.903770 (0.102321)	0.9
KNN	0.871032 (0.124375)	0.85
CART	0.922817 (0.084685)	0.95
NB	0.949603 (0.062133)	0.95
SVM	0.899206 (0.071488)	075

TABLE IV
10-FOLD CROSS VALIDATION WITH MODEL THAT USES SAMPLES OF B, M_w AND EM AS TRAINING DATA

Model	10-fold Mean Accuracy (std)	Validation Accuracy
LR	0.914881 (0.093706)	0.9
KNN	0.897817 (0.099469)	0.9
CART	0.913492 (0.097603)	0.9
NB	0.949603 (0.062133)	0.95
SVM	0.905556 (0.101645)	0.95

C. Is a sequential model (LSTM) better in terms of accuracy than the best classical model (Naïve Bayes) in detecting potential mutants?

The experiments described above were repeated using the LSTM. That is, three methods of training a model are considered. (1) using equal amounts of benign data and malware collected from the web (B and M_w); (2) using equal amounts of benign data and evolved mutants (B and EM); (3) 50 samples of benign data, and 25 samples each of malware from the web and evolved mutants (B, M_w and EM). We compare the results of the best non sequence based model (Naïve Bayes) and LSTM on the aforementioned test cases.

From Tables V and VI, we see that as in the classical models, the model trained using only malware collected from the web performs poorly on the mutant samples, although the LSTM has slightly higher accuracy than Naïve Bayes with an accuracy of 53% as opposed to an accuracy of 40% for Naïve Bayes. The LSTM outperforms Naïve Bayes for the model trained purely on benign and evolved samples ((B and EM), with accuracy of 90%). On the other hand, the model trained with equal proportions of malicious samples from the web and the evolved mutants performs less well than the Naïve Bayes model, obtaining accuracy of 62% compared to 82%.

TABLE V
COMPARISON OF ACCURACY OBTAINED ON THE UNSEEN TEST SET EM_u USING A NAÏVE BAYES MODEL TRAINED ON 3 DIFFERENT TRAINING SETS

Training Data	EM_u (Accuracy)
B and M_w	0.4
B and EM	0.84
B, M_w and EM	0.82

TABLE VI
COMPARISON OF ACCURACY OBTAINED ON THE UNSEEN TEST SET EM_u USING AN LSTM MODEL TRAINED ON 3 DIFFERENT TRAINING SETS

Training Data	EM_u (Accuracy)
B and M_w	0.53
B and EM	0.9
B, M_w and EM	0.62

TABLE VII
MODELS THAT USE B AND EM AS WELL AS B, M_w AND EM AS TRAINING DATA AND M_w AS TEST DATA FOR BOTH NAÏVE BAYES AND LSTM

Training Data	Naïve Bayes Accuracy (M_w)	LSTM Accuracy (M_w)
B and EM	1	0.73
B, M_w and EM	1	0.91

D. Do models trained on evolved data representing future mutants retain their ability to recognise existing malware?

The previous results show that models trained using evolved mutants are capable of recognising other evolved mutants. However, it is important to evaluate whether these models are fitted to the evolved mutants and therefore fail to recognise existing malware.

Hence, we conduct additional experiments where the models trained using evolved samples are tested on a set of unseen malicious samples from M_w, i.e. the samples collected from the web. The model trained with (B, EM) is tested on the unseen set M_w. The model trained with (B, EM, M_w) is tested on the 25 samples from M_w not used in training.

As can be seen from Table VII, it is clear that models trained with potential mutants are also able to detect other malicious samples. An accuracy of 100% is obtained for both Naïve Bayes models, with 73% and 91% respectively for the LSTM model that use (B, EM) and (B, M_w, EM) as training data. Hence, we conclude that the models are not over-fitting to the new mutants, and losing their ability to generalise.

We also do an analysis that investigates when the sequential (LSTM) and non-sequential (Naïve Bayes) methods agree on which instances are misclassified. For each method, we compare the overlap between the set of mis-classified instances produced by each method in order to understand whether both methods fail on the same instances or not (where *overlap* refers to the number of instances that are mis-classified by *both* methods). The exact instances that are mis-classified are given in [6].

From Table VIII, we see that using the M_w set during training results in a large number of mis-classified instances by both methods, and the overlap is high (10 instances, representing approximately 56% and 71% of mis-classified instances for Naïve Bayes and LSTM respectively). All the overlapping mis-classified instances come from the Dougalek family. For the EM training set the overlap is high, in that it represents 40% of the instances mis-classified by Naïve Bayes and 66% of those mis-classified by the LSTM. Of the 2 overlapping instances, one is from the Dougalek family and the other from Droidkungfu. Finally, for the training set that includes M_w, EM, although the overlap is also 2 instances, this represents a smaller proportion of the instances mis-classified by each method (approximately 33% and 18% respectively). Again the overlapping instances are from the Dougalek and Droidkungfu families.

TABLE VIII
MIS-CLASSIFIED INSTANCES ON THE UNSEEN TEST SET EM_u USING BOTH NAÏVE BAYES (NB) AND LSTM MODELS TRAINED ON 3 DIFFERENT TRAINING SETS. THE FINAL COLUMN NOTES WHICH FAMILIES THE OVERLAPPING INSTANCES CAME FROM. THE FEATURE VECTOR DATA IS AVAILABLE IN [6]

Training Data	# Mis-classified		Overlap	Instances
	NB	LSTM		
B and M_w	18	14	10	Dougalek(10)
B and EM	5	3	2	Dougalek(1), Droidkungfu(1)
B, M_w and EM	6	11	2	Dougalek(1), Droidkungfu(1)

V. CONCLUSION

It is well known that metamorphic malware presents a difficult class of malware which can evade detection by machine learning models, due to the number of mutation techniques used to obfuscate their code. We attempt to address this through a method that combines evolutionary computing and machine-learning. The former evolves a large set of malicious mutant variants of malware. The latter then uses this data to augment training sets, to develop models that are capable of recognising both existing malware and its future variants.

We have shown experimentally that machine learning models trained on existing malicious samples are vulnerable to potential mutants, and that training with evolved data addresses this. In addition to evaluating classical feature-based ML models, we have also applied an LSTM using a time-ordered sequence of system calls as input and shown that this can outperform the classical models in some cases. Finally, our results show that the models generalise over both existing malware and the evolved variants.

Future work will focus on designing a Generative Adversarial Network (GAN), where the mutants generated are not only used to train machine learning models for improved detection, but the error in the detection is used to produce more evasive mutants, potentially leading to even better detection rates.

REFERENCES

[1] D. Ucci, L. Aniello, and R. Baldoni, “Survey of machine learning techniques for malware analysis,” *Computers & Security*, vol. 81, pp. 123 – 147, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404818303808

[2] D. Lowd and C. Meek, “Adversarial learning,” in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, ser. KDD ’05. New York, NY, USA: ACM, 2005, pp. 641–647.

[3] K. O. Babaagba, Z. Tan, and E. Hart, "Nowhere metamorphic malware can hide - a biological evolution inspired detection scheme," in *Dependability in Sensor, Cloud, and Big Data Systems and Applications*, G. Wang, M. Z. A. Bhuiyan, S. De Capitani di Vimercati, and Y. Ren, Eds. Singapore: Springer Singapore, 2019, pp. 369–382.

[4] ——, "Automatic Generation of Adversarial Metamorphic Malware Using MAP-Elites," in *23rd European Conference on the Applications of Evolutionary and bio-inspired Computation*, P.A. Castillo et al, Ed. Seville: Springer-Verlag New York, Inc., 2020, pp. 1–16.

[5] A. Graves, *Supervised Sequence Labelling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 5–13. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2_2

[6] K. O. Babaagba, Z. Tan, and E. Hart, "Improving classification of metamorphic malware," https://github.com/KehindeOloye/Improving-Classification-of-Metamorphic-Malware.git, 2020.

[7] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Cloud-based malware detection for evolving data streams," *ACM Trans. Manage. Inf. Syst.*, vol. 2, no. 3, Oct. 2008. [Online]. Available: https://doi.org/10.1145/2019618.2019622

[8] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *AI 2016: Advances in Artificial Intelligence*, B. H. Kang and Q. Bai, Eds. Cham: Springer International Publishing, 2016, pp. 137–149.

[9] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016. [Online]. Available: https://doi.org/10.1007/s00500-014-1511-6

[10] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS 17. New York, NY, USA: Association for Computing Machinery, 2017, p. 506519. [Online]. Available: https://doi.org/10.1145/3052973.3053009

[11] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, April 2014.

[12] N. rndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 197–211.

[13] Z. Yin, F. Wang, W. Liu, and S. Chawla, "Sparse feature attacks in adversarial learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 6, pp. 1164–1177, June 2018.

[14] E. Aydogan and S. Sen, "Automatic generation of mobile malwares using genetic programming," in *Applications of Evolutionary Computation*, A. M. Mora and G. Squillero, Eds. Cham: Springer International Publishing, 2015, pp. 745–756.

[15] W. Xu, Y. Qi, and D. Evans, "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, 2016.

[16] F. Zhang, P. P. K. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 766–777, March 2016.

[17] R. Lu, "Malware detection with lstm using opcode language," *arXiv:1906.04593*, 2019.

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

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

[20] J. I. Hoffman, "Chapter 33 - logistic regression," in *Basic Biostatistics for Medical and Biomedical Practitioners (Second Edition)*, 2nd ed., J. I. Hoffman, Ed. Academic Press, 2019, pp. 581 – 589. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128170847000334

[21] V. Kecman, *Support Vector Machines – An Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–47. [Online]. Available: https://doi.org/10.1007/10984697_1

[22] G. I. Webb, *Naïve Bayes*. Boston, MA: Springer US, 2010, pp. 713–714. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_576

[23] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*, ser. The Wadsworth statistics/probability series. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1984. [Online]. Available: https://cds.cern.ch/record/2253780

[24] R. Nisbet, G. Miner, and K. Yale, "Chapter 9 - classification," in *Handbook of Statistical Analysis and Data Mining Applications (Second Edition)*, 2nd ed., R. Nisbet, G. Miner, and K. Yale, Eds. Boston: Academic Press, 2018, pp. 169 – 186. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780124166325000098

[25] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *ArXiv*, vol. abs/1506.00019, 2015.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.