The Thesis Committee for Xiruo Wang
Certifies that this is the approved version of the following Thesis:

MDEA: Malware Detection with Evolutionary Adversarial Learning

APPROVED BY

SUPERVISING COMMITTEE:

Risto Miikkulainen, Supervisor
Greg Durrett

MDEA: Malware Detection with Evolutionary Adversarial Learning

by

Xiruo Wang

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

December 2019

Acknowledgements

I would thank my supervisor Prof. Risto Miikkulainen for providing me direction in this interesting domain of research and bringing me inspiration when I encountered difficulties. I really appreciate his patience and kindness. I also want to thank Prof. Greg Durrett for taking out his valuable time to be the reader for my thesis.

I would also like to thank my uncle and my aunt, Eric and Lucy Zhao for their tremendous help for the past 6 years. Also, I want to thank my grandparents for their great care.

At last, I would like to thank all my friends for their support during this research.

Abstract

MDEA: Malware Detection with Evolutionary Adversarial Learning

Xiruo Wang, M.S.Comp.Sc

The University of Texas at Austin, 2019

Supervisor: Risto Miikkulainen

Many applications have used machine learning as a tool to detect malware. These applications take in raw or processed binary data to feed neural network models to classify benign or malicious files. Even though this approach has proved effective against dynamic changes, such as encrypting, obfuscating and packing techniques, it is vulnerable to specific evasion attacks to where that small changes to the input data cause misclassification at test time. In this paper, I propose MDEA, an Adversarial Malware Detection model that combines a neural network and evolutionary optimization attack samples to make the network robust against evasion attacks. By retraining the model with the evolved malware samples, network performance improves a big margin.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

The high proliferation of and dependence on computing resources in daily life has not only greatly increased the potential of malware to harm consumers but has also significantly increased the attack space (Acquisti et al. 2010). It is estimated that almost one in four computers operating in the U.S. were already infected by malware in 2008 (Plonk et al. 2008) and according to Kaspersky Lab, up to one billion dollars was stolen from financial institutions worldwide due to malware attacks in 2015 (K. Lab 2015). More recently, the notorious and widespread NotPetya ransomware attack is estimated to have caused $10 billion dollars in damages worldwide. Even worse, as reported by McAfee Labs, the diversity of malware is still evolving in expanding areas such that in Q1 2018, on average, five new malware samples were generated per second (Beek et al. 2018). As a specific example, total coin miner malware rose by 629% in Q1 to more than 2.9 million samples in 2018 (Beek et al. 2018).

As a result of the magnitude of the threat posed by malware, a great deal of research has been conducted on the problem of malware identification. At the moment there are two widely used approaches for malware detection: dynamic analysis, which obtains features by monitoring program executions and static analysis, which obtains feature from binary programs without running them. Intuitively, the dynamic analysis will be the first choice, since it can provide us the most accurate program behavior data. However, there are many issues in dynamic analysis in practice. Dynamic analysis requires a specially constructed

running environment such as a customized Virtual Machine (VM), which will be a huge computational burden to test numerous samples. Furthermore, in order to bypass this defense, some malwares will alter the behaviors when they are detected (Raffetseder et al. 2015, Garfinkel et al. 2007). Even the malwares don't change its behavior during the detection process, the analysis environment can get false positive data that may result from other software.

On the other hand, the static analysis methods also have their disadvantages. The signature-based method, in which malware features are extracted by computer security experts, provides the basis for most commercial antivirus products. While they are widely used, static pattern analysis (Reddy et al. 2006, Narouei 2015) (like API calls, N-grams, and so on) is limited in their ability to combat various encryption, polymorphism and obfuscation methods used by malware attackers. As for machine learning based malware classification technologies, several successful attempts (Rieck 2011, Zakeri 2015) have been applied to malware detection, which rely heavily on relevant domain knowledge for to provide malware analysis and determine features. This approach cannot adapt to fast-changing malware patterns and comes with the high cost of artificial feature engineering.

In recent years, researchers have begun exploring a new frontier in data mining and machine learning known as deep learning. Deep learning techniques are now being leveraged in malware detection and classification tasks, (Hardy et al. 2016, Gibert 2016, Drew et al. 2017, Yan et al. 2018) for exploring SAE, CNN, and RNN models to devise malware detection architectures. Although research thus far has provided promising

results, there are still many open challenges and opportunities for DL to improve performance in malware identification and classification tasks. First of all, due to the quick increase in the amount of novel malware, malicious techniques and patterns are changing and evolving rapidly. As a result, handling novel malware is one of the most pressing issues that deep learning methods might handle. In addition, in contrast to the natural language processing or computer vision tasks that are usually explored in deep learning tasks, malware byte files and assembly instructions have less understandable patterns. The difficulty of adapting these traditional deep learning methods directly to the task of malware classification and identification brings us to further explore data prepossessing techniques for network inputs. Furthermore, the adversarial attacks against neural networks, which only manipulate small portion of the input data and causes misclassification, has been proven to be one of the biggest vulnerabilities of DL. Even though these types of adversarial attack are less common on malware detection models because of the complexity and fragility of binary executables, there have been research shown that evading deep neural network for malware binary detection is possible. In their work, they trained a gradient-based model to append bytes to the overlay section of malware samples. Even though the model successfully evaded the deep neural network, both the model and the modification method are rather simple and cannot cover the complicated modifications real malware writers do.

In order to explore the data space more thoroughly, an action space is defined. The action space consists of 10 different modification methods of binary programs. On top of

that, an evolutionary optimization is used to search the best action sequence of a specific malware since many researches have shown that evolutionary learning sup

In this paper, I propose MDEA, an Adversarial Malware Detection model that combines a deep neural network and an evolutionary optimization. MDEA consists of a convolutional neural network that classifies raw byte data from malware binaries, and an evolutionary optimization that modifies the malwares that are detected. In contrast to simply append bytes to the end of each file, an action space is defined for the evolutionary optimization to pick from and choose best action sequences for each malware sample. With the evolutionary learning, the probability that the generated input sample is classified as benign can be maximally increased. The new samples then will be fed into the detection network for retraining. These three steps form a cycle that is similar to Generative Adversarial Nets (GAN) (Goodfellow et al. 2014).

The experiments are performed on 7371 Windows Portable Executable (PE) malware samples and 6917 benign PE samples. The results show that MDEA not only drastically decreases the detection model accuracy, but also increases the overall detection performance from 90% to 93% after the retrain process. With this result I aim to claim that adversarial evolutionary training can improve both the robustness and the performance of malware detection network.

The rest of the paper is structured as follows. Chapter 2 presents the related work. Chapter 3 describes the overview of my malware detection system and the details of each component. Chapter 4 describes my experimental setups and discuss the results. Finally,

Chapter 5 draws conclusions for my experiment and Chapter 6 provides suggestions for further research on this topic.

Chapter 2: Related Work

Malware detection and classification has been a well-studied problem for many years. Notably, in 2015, the open Kaggle Contest: Microsoft Malware Classification Challenge (BIG 2015) (Kaggle, 2015) created a large burst of energy and attention towards the goal of malware classification. The champion of this contest used machine learning with sophisticated static pattern analysis in order to achieve high accuracy in classifying the malware samples provided in the challenge. Observing this, my goal is to leverage deep learning models without sophisticated feature engineering. In this section, I briefly introduce related works in signature-based, learning-based (Yan et al. 2018), adversarial-based (Anderson et al. 2018) approaches and evolutionary techniques.

2.1 Signature-based Malware Detection

Signature-based and behavior-based methods are widely used in the anti-malware industry and are often used to identify "known" malware (Cloonan 2017). When an anti-malware solution provider identifies an object as malicious, its signature is added to a database of known malware. These repositories may contain hundreds of millions of signatures that identify malicious objects. One of the major advantages of signature-based malware detection is its thoroughness since it follows all conceivable execution ways of a given document (Souri et al. 2018). Because of the simplicity of building such a system, signature-based malware detection has been the primary identification technique used by malware products and remains the base approach used by the latest firewalls, email and network gateways. Therefore, many researches have been done in this field. Santos et al. (2010) created an opcode sequence-based malware detection system. Preda et al. (2007) proposed a semantics-based framework for reasoning about malware detector. Chaumette et al. (2011) created an automated virus signature extraction system with abstract

interpretation. Ojugo and Eboka (2019) investigated detection of metamorphic malware attacks using the Boyer Moore algorithm for string-based signature detection scheme. Venugopal and Hu (2008) showed that a signature matching algorithm can be suitable for use in mobile device scanning due to its low memory requirements. Fan et al. (2015) expanded the normal signature approaches to the API log domain and utilized with data mining techniques. Fraley and Figueroa (2016) presented a unique approach leveraging topological examination using signature-based techniques. They also used data mining techniques in order to uncover and spotlight the properties of malicious files. Despite the widespread adoption of signature-based malware detection within the information security industry, malware authors can easily evade this signature-based method through techniques such as encryption, polymorphism, and obfuscation. Signature-based analysis is, therefore, poorly equipped to handle the current state of malware generation.

2.2 Learning-based Malware Classification

Because of the weaknesses of signature-based malware detection, machine learning is a popular approach to signatureless malware detection. Many different malware detection approaches using machine learning technology have been proposed in recent years, such as static analysis, which learns statistical characteristics (e.g. API calls, N-grams), or dynamic behavior analysis, which analyzes the behavior of a system against a baseline in order to determine anomalous (and possibly malicious) behavior. I focus on static analysis within the scope of this paper. In the Kaggle Microsoft Malware Contest (Kaggle 2015), the winner used many sophisticated features for their knn model in order to achieve high performance. Some other machine learning techniques are also studied in different works. Elkhawas et al. (2018) proposed a machine learning model with Support

Vector Machine (SVM). Wang et al. (2015) developed an automatic malware detection system by training an SVM classifier based on behavioral signatures and across-validation scheme was used for solving classification accuracy problems by using SVMs associated with 60 families of real malware. Rehman et al. (2018) have reverse engineered the Android Apps to extract manifest files, and employed machine learning algorithms to efficiently detect malwares. They observed that SVM in case of binaries and KNN in case of manifest.xml files are the most suitable options in robustly detecting the malware in Android devices. Altaher (2016) proposed a hybrid neuro-fuzzy classifier (EHNFC) for Android malware classification using permission-based features. The proposed EHNFC not only has the capability of detecting obfuscated malware using fuzzy rules but can also evolve its structure by learning new malware detection fuzzy rules to improve its detection accuracy when used in detection of more malware applications. Yuan et al. (2016) proposed to associate the features from the static analysis with features from dynamic analysis of Android apps and characterize malware using deep learning techniques. Yuxin and Siyi (2017) presented a deep belief network (DBN) that represents malware as opcode sequences. DBNs can use unlabeled data to pretrain a multi-layer generative model, which can better represent the characteristics of data samples.

Unlike all the above research works, in order to cut down on the necessity of expert analysis, I will only take the basic features as input for deep learning model. Many different deep learning models have been proposed on malware detection. Some people intend to solve this problem by LSTM model (Yan 2018), while others proposed "malware image" that are generally constructed by treating each byte of the binary as a

gray-scale pixel value, and defining an arbitrary “image width” that is used for all images. From all these approaches, I found that the MalConv network (Raff et al. 2017), which use raw byte embeddings as the input, achieved the highest accuracy. Therefore, I choose to use this model as my detection model.

2.3 Adversarial Model for Sample Generation

Nguyen et al. ‘s paper (2015) arguably inspired the later research on adversarial model. They found that it was easy to produce images that were completely unrecognizable to humans, but deep neural networks (DNNs) could recognize it with high confidence. They trained DNN on ImageNet and MNIST datasets and produced many human-unrecognizable images. Goodfellow et al. (2014) proposed the first generative adversarial nets (GAN). GAN consists of two models. One of the models is the generative model, which captures the data distribution and a discriminative model that estimates the probability that a sample came from the training data rather than from generative model.

GAN has a very huge potential since it can learn to mimic any data distribution because of the generality of its structure. Therefore, GAN has been used in many domains such as computer vision, natural language processing etc. Zhu et al. (2017) used cycle GAN to build a mapping function from two image domains. Karras et al. (2018) proposed a way to modify the network structures for both generator and discriminator in GAN. The method they used greatly increased the variation in generated images.

Recent work in adversarial machine learning has shown that deep learning models for machine learning are susceptible to gradient-based attacks. Anderson (Anderson et al. 2018) proposed a more general framework based on reinforcement learning (RL) for attacking static portable executable (PE) anti-malware engines. They show in experiments that this adversarial learning method can attack a gradient-boosted machine learning model

and evade components of publicly hosted antivirus engines. Suciu et al. explored the area of adversarial examples for malware detection by training an existing model on a production-scale dataset. They showed that some previous attacks are less effective than initially reported, while simultaneously highlighting architectural weaknesses that facilitate new attack strategies for malware classification. Grosse et al. (2017) expanded on existing adversarial example crafting algorithms to construct a highly effective attack that uses adversarial examples against malware detection models. Maiorca et al. (2019) presented a system to generate malwares embedded in PDF files. Kolosnjaji et al. (2018) proposed a gradient-based attack model that is capable of evading a deep network by only changing few specific bytes at the end of each malware sample, while preserving its intrusive functionality. This work is interesting because they used the same malware detection network structure (MalConv) and similar approach as mine. They showed that by simply appending learnt bytes at the end of malware samples, they were able to decrease the detection accuracy by more than 50%. Even though this work achieved good result, neither of them used their generated attack samples to improve the detection model. Their learning methods are also mathematically unsophisticated, which may result in nonoptimal solution.

I will reproduce and improve upon the methods in these paper with evolutionary learning and leverage the generated adversarial samples back to the deep learning model to further improve the accuracy.

2.4 Evolutionary Algorithm

Evolutionary algorithm (EA) uses mechanisms inspired by biological evolution, such as mutation, recombination and selection to select the best individual of a population to solve an optimization problem. At the beginning, EA is considered as a scalable

alternative to reinforcement learning (Salimans et al. 2017). In recent years, there have been many researches showed that EA performed much better than traditional optimizing method such as gradient descend in different domain (Real et al. 2019; Podryabinkin et al. 2019). This advantage of EA has been proven even more in machine and deep learning because of the diversity and complexity it provides (Young et al. 2015; Hooman et al. 2018). Martín et al. (2016) created an Android malware detection system with evolutionary strategies to leverag third-party calls to bypass the effects of these concealment strategies. Petroski et al. (2018) evolved the weights of a DNN with genetic algorithm (GA) and it performed well on hard deep RL problems, including Atari and humanoid locomotion. Esteban et al. (2019) evolved an image classifier— AmoebaNet-A—that surpassed hand-designs for the first time. They modified the tournament selection evolutionary algorithm by introducing an age property to favor the younger genotypes. They also showed that their evolved model worked well in their benchmark against reinforcement learning approaches. Chen et al. (2019) built a model to generate groundwater spring potential map. They utilized GA to perform a feature selection procedure and data mining methods for optimizing set of variables in groundwater spring assessments.

In all the above literatures, EA showed a great advantage against the traditional optimization method such as stochastic gradient descent and reinforcement learning. The gradient-free nature makes EA less vulnerable to local minimum issue and easier to approach more general solutions for a complicated parameter search problem. EA also has higher robustness and performs well when the number of time steps in an episode is long, where actions have long-lasting effects, or if no good value function estimates

are available (Salimans et al. 2017). Therefore I chose EA as the optimization method for my project.

Chapter 3: Model

This chapter discusses the design and process of my proposed MDEA model. Section 3.1 describes an overview of the model structure. Section 3.2 discusses the dataset information and the details of the detection model. Section 3.3 explains the definition and details of action space, which is used by evolutionary optimization method. Finally, Section 3.4 describes the evolutionary optimization algorithm in details.

3.1 Structure Overview

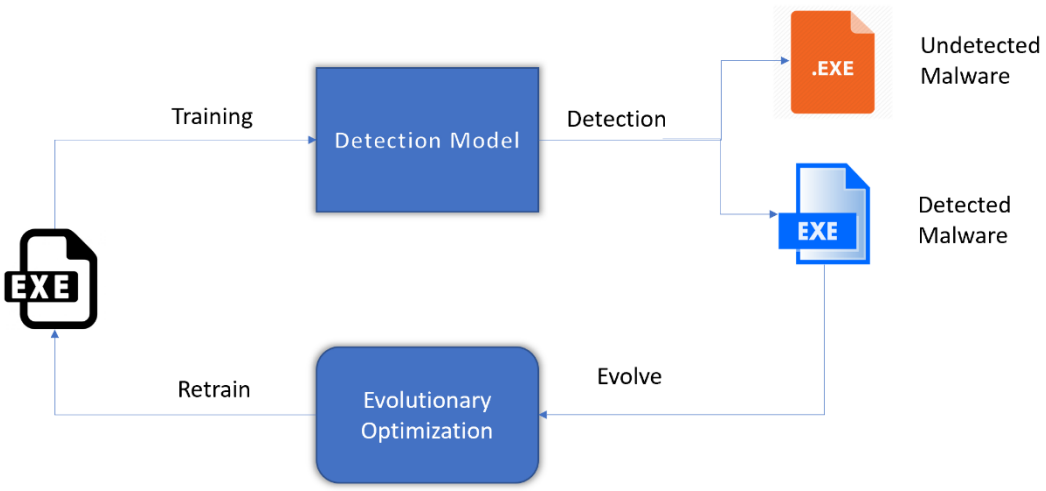

Figure 1: Model Diagram
This diagram shows the overall model flow of MDEA.
Top: Detection Model based on the MalConv network Raff et al (2017).
Bottom: The Evolutionary Optimization Algorithm

Overall, the malware detection process consists of two major parts. A diagram of our model is shown in Figure 1. The first part involves preprocessing malware sample data and feeding these data to our 1-D Convolutional network. The second part uses evolutionary optimization to evolve adversarial malware samples to evade the network. All the newly generated malwares that successfully bypass the detection model will be added

into the training set to retrain the detection model. The first and second part together form a loop as shown in Figure 1. During the training phase, this loop is iterated multiple times until the detection accuracy of the detection model converges to an optimal value.

3.2 Dataset and Detection Model

The dataset consists of 14,288 PE files. 7371 of them are malware samples, which were downloaded from VirusShare. The rest 6917 PE files are benign files that were gathered by crawling different websites. The deep neural network I trained and attacked in this paper is the MalConv Network proposed by Raff et al (2017). Figure 2 shows the detailed structure of MalConv Network. MalConv takes in up to k bytes data as input. Each byte is represented by a number $\mathbf{A} = \{0, \dots, 255\}$. The k bytes $\in \mathbf{A}^k$ data that are extracted from input file are padded with zeros to form a vector $\mathbf{x}$ (if there are more than k bytes in the file, just take the first k bytes without padding). Then each element of vector $\mathbf{x}$ is fed into a trainable embedding layer to get an embedded vector $\mathbf{z}$ of 8 elements. After this embedding process, one dimensional vector x becomes a matrix $\mathbf{Z} \in R^{dx8}$. This matrix $\mathbf{Z}$ is then fed into two layers 1-d convolutional layers. These two layers use Rectified Linear Unit (ReLU) and sigmoidal activation functions respectively. By combining these two layers with gating, the vanishing gradient problem caused by sigmoidal activation functions is avoided. The obtained values are then fed to a temporal max pooling layer followed by a fully connected layer with ReLU activations. The final classification is made by the probability output from the last fully connected layer as f(x). If $f(x) > 0.5$ then the sample is a benign file, otherwise it is classified as malware.

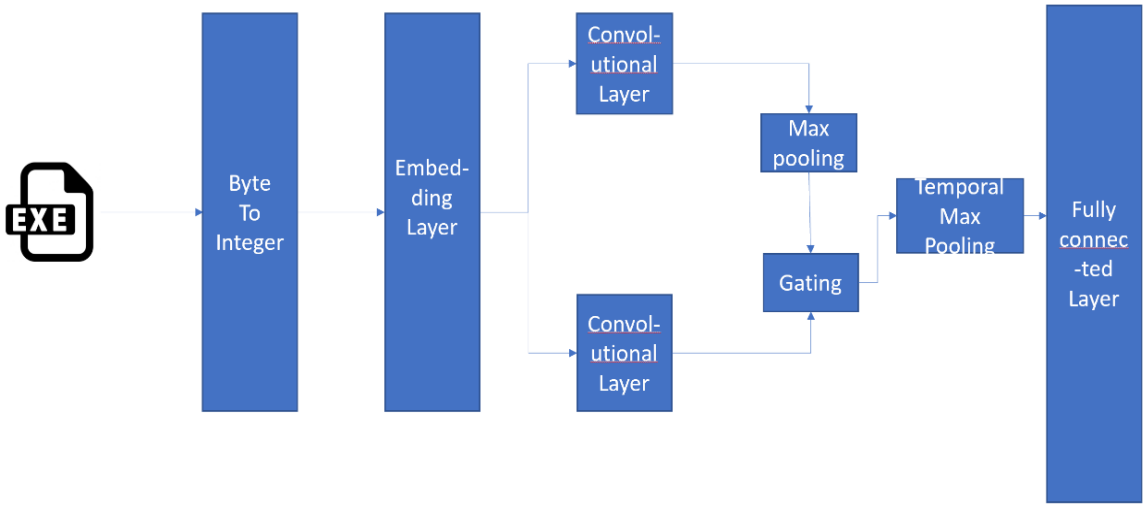

Figure 2: Detection Model

This diagram shows the layer details of MalConv.

3.3 Action Space

The action space is built on top of some PE file layout knowledge. Therefore, before introducing the action space, I will provide a brief description of the structure of PE files to better understand the actions.

A PE file consists of a number of headers and sections that tell the dynamic linker how to map the file into memory. In general, there are three types of layouts in PE: header, section table and data. Header is a data structure that contains basic information on the executables. Section table is a table that describes the characteristics of each file section. The data layout contains the actual data that related to each section. Malwares usually modify some of these structures to create malicious activities that are hard to detect. The stealth and sensitivity of these modifications make it even harder to alter some bytes without breaking the malware functionality. However, some of the sections are not important for the program to run. Some of them are even neglected by OS such as the

overlay section. By knowing such type of knowledge, I can construct the following action space:

The action space consists of 10 different actions with trainable parameters. These actions are inspired by Anderson et al.'s (2017) work. In their paper, they make each action accept random parameters to test evasion on a gradient boosted decision tree model with reinforcement learning. However, this randomness becomes a big issue in MDEA since evolutionary algorithm already introduced enough generality to the problem and more randomness would cause the model to not converge. Therefore, I introduce a parameter set corresponds to each action to make the model converge with acceptable time.

The actions are:

1. Add a function to the import address table that is never used
2. Manipulate existing section names
3. Append bytes to extra space at the end of sections
4. Create a new entry point
5. Manipulate signature
6. Manipulate debug info
7. Pack the file
8. Unpack the file
9. Modify header checksum
10. Append bytes to overlay section.

Note that some of these actions are not recoverable such as delete a signature, which means once the evolutionary optimization algorithm chooses to perform this action, all later generations of this malware will not be able to effectively perform the same action again. This irreversibility causes the diversity to drop drastically. I addressed this issue with more details in the experiment chapter later.

3.4 Evolutionary Optimization

I used a framework called DEAP (Distributed Evolutionary Algorithms in Python) to construct our evolutionary optimization algorithm. The evolutionary algorithm consists of three major parts: population initialization and evolution, binary modification and individual evaluation. Figure 3 shows an overview of the evolution algorithm.

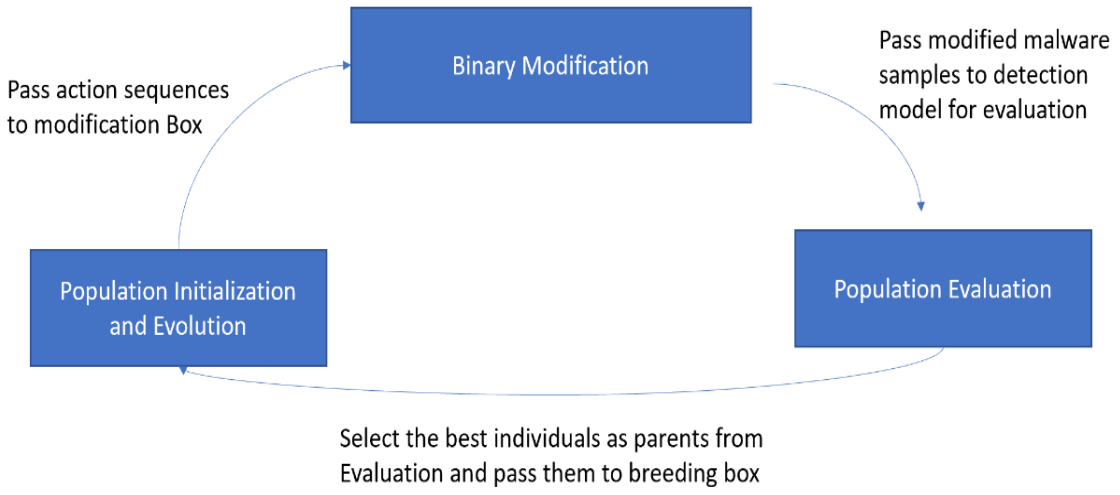

Figure 3: Evolutionary Algorithm

This diagram shows the evolutionary optimization cycle

The population evolution part has different methods to breed new children to evolve. Mutation and crossover as two major evolution methods. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. There are many different types of mutation such as shrink mutation (Ronco et al. 2013), uniform mutation and boundary mutation etc. I chose to implement the "mutShuffleIndexes" method in DEAP since most of the mutation

method only works for integers or floats and cannot be applied to action sequences. The "mutShuffleIndexes" method shuffles the attributes of the input individual. During the shuffle phase, there is also a probability to replace some of the elements in those attributes with randomly chosen elements. Crossover combines the genetic information of two parents to generate new offspring. It is one way to stochastically generate new solutions from an existing population, and analogous to the crossover that happens during sexual reproduction in biology. I implemented the uniform crossover method, which chooses each attribute from either parent with equal probability resulting in offspring which inherit more genetic information from one parent than the other. Selection is another important method in evolutionary algorithm. After the population is evolved, the action sequence will be sent to the binary modification section to produce modified malwares. The modified malwares will be evaluated by the detection model and the statistics for evolving next generation will be calculated. After the evaluation step, selection method picks the best offspring as the parent for next generation. For the perspective of simplicity, I implemented the "selectBest" selection algorithm for this project. This cycle continues until there is enough data for further training, or the generation limit is reached.

Let us denote all the malwares that got detected as M_d and all the malwares that were not detected as M_n, then I want to find a function f(x) that makes $f(M_d) = M_n$. The function f(x) is presented by action sequences $\{(a_0, p_0), (a_1, p_1), (a_2, p_2) \dots (a_i, p_i)\}$ where a_i is an action from action space and p_i is the corresponding parameter for that action (i.e.: for appending overlay method, *p* indicates how many bytes append at the end). Our evolutionary optimization then aims to find a group of f(x) by crossover, mutation and selection. Let us denote the detection model as D(x), and for any malware sample m, if $D(m) > 0.5$, *m* is benign software. For each individual m_d in M_d, our evolutional model can be expressed by:

Equation 1: $f_{opt}(m_d) = \text{argmax}(D(f(m_d))$.

With Equation 1, the overall algorithm for the evolution is expressed in the following form:

Algorithm 1: Evolution Algorithm

Result: Action sequence A with Parameter set B
initialization: Population Size P, Action Sequence Limit a, Crossover probability C, Mutation probability M, Generation Limit G and Data size Limit D;
d = 0;
g = 0;
while $d < D$ *and* $g < D$ **do**
    m = ModifyMalwares(p);
    s = Evaluate(m);
    d = FindGoodActions(s,p);
    g += 1;
    p = NextGeneration(p,s,P,a,C,M);
end
A,B = OutputResult(d);
return A,B;

Chapter 4: Experiments

This chapter will discuss the experimental setups and results. Section 4.1 describes the details of the datasets, the division of training set and validation set, and the hardware I used to run this experiment. Section 4.2 describes the experimental results and their importance with two data graphs. Section 4.3 discusses the dead species problem I encountered during the experiment. Section 4.4 shows how evolutionary algorithm can overfit to a never-seen develpment set and potential solutions to the overfit problem. Section 4.5 presents the result of test modified malwares test against VirusTotal, a popular malware detection website.

4.1 Experimental Setup

To set up the experiment I collected 7371 malware samples from VirusShare, a malware sample data website, and 6917 benign samples from web crawling. All these 14,288 samples are Windows Portable Executables (PE). I divided dataset into two sets with 9:1 ratio as training set and validation set. I ran both neural network training and evolutionary optimization method on Texas Advanced Computing Center (TACC) Maverick2 server with 4 Nvidia 1080-TI GPUs and 16 Intel Xeon CPUs. The training time for detection network was around 10 hours and the running time for evolutionary optimization was around 24 hours each cycle. I recorded the accuracy and the number of modified bytes after each cycle.

4.2 Experimental Result

This section presents the experimental result. The result graphs are shown in Figure 4 and 5.

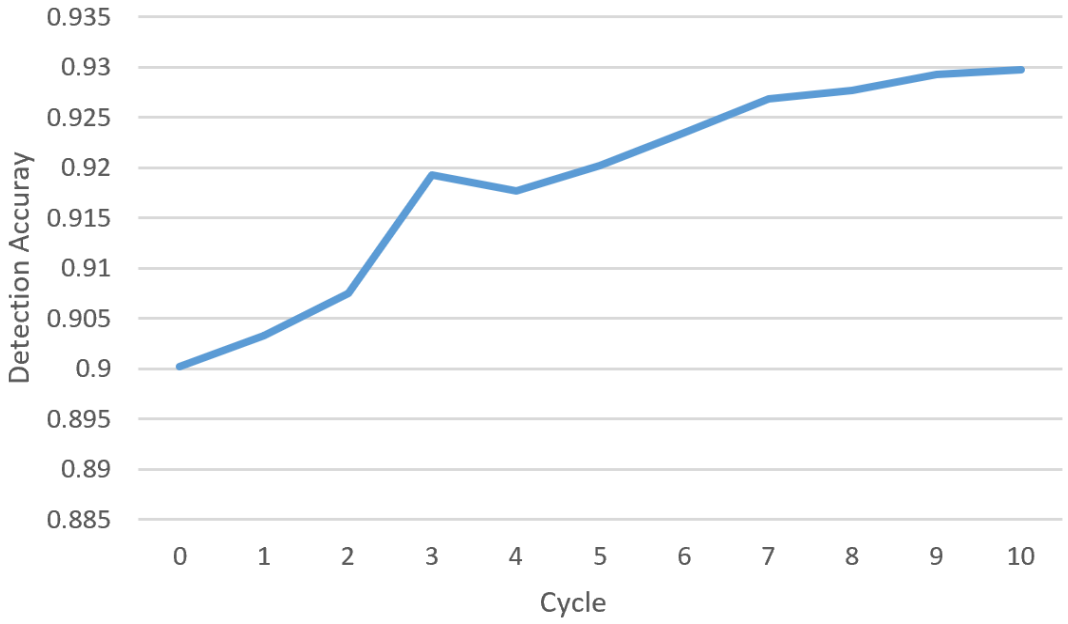

Figure 4: Detection rate against number of cycles

This graph shows how detection accuracy changed with number of cycles

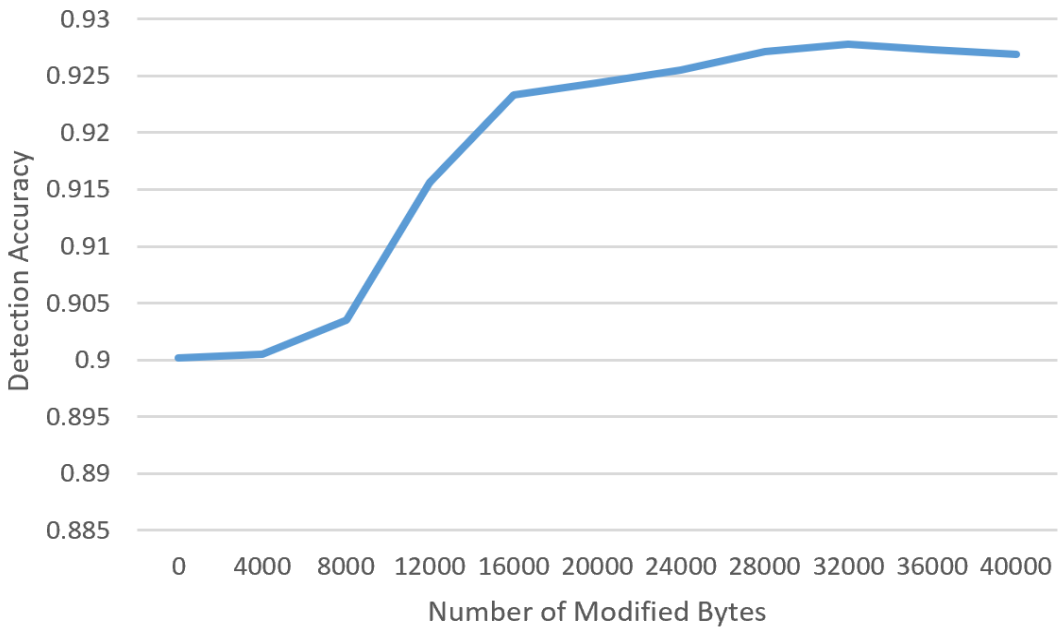

Figure 5: Detection rate against number of modified bytes

This graph shows how detection accuracy changed with number of modified bytes

Figure 4 shows the detection accuracy of the detection model increases with the number of cycles. After 10 cycles of training, which is around 12 days, the accuracy increases from 90% to around 93%. Note that there is a drop on the Cycle 4 in the graph, which I suspect that is caused by a dead species issue. Section 4.3 addresses this issue with detailed discussion.

Figure 5 shows the relation between the detection accuracy and the number of modified bytes. I noticed that there is a big jump from 8000 bytes to 12,000 bytes. I think the reason why this happens is because some of the sections in the PE file have a certain length and when the number of modified bytes increases above a certain threshold, the modification bytes start to capture more malware patterns, which increases the accuracy by a big margin.

4.3 Dead Species

This section discusses the dead species problem that caused the accuracy drop on Cycle 4 in Figure 4. There are many actions in the action space that are not reversable such as removing signatures, modifying checksum sections etc. If any of these actions are performed on the malware, a later generation will not be able to reverse it. Since it is unlikely to find an optimal action sequence at the beginning of evolution, picking such irreversible actions will drastically reduce the search space and the offspring that contain those actions will be stuck at bad local optimum.

Figure 6 shows one of the examples of such dead species. There is an obvious gap between the normal species and the dead species after certain number of modified bytes. After investigating the records of these two evolutions, the result showed that the dead species picked “delete-checksum” and “remove signature” actions when number of modified bytes was equal to 15,879. This result validates my conjecture that irreversible actions caused the dead species problem.

In order to solve this problem, validation weights are introduced into the evolutionary optimization algorithm. These weights represent the probability of each action being picked. The actions that can cause dead species are assigned with a very low probability. This countermeasure worked well in practice and increased the average accuracy by 1%. However, the validation weights only delay the occurrence of dead species instead of solving the problem. Any individuals that picked irreversible actions will lead to dead species and lose their diversity in evolution. Further possible techniques for dealing with this problem will be discussed in Chapter 5.

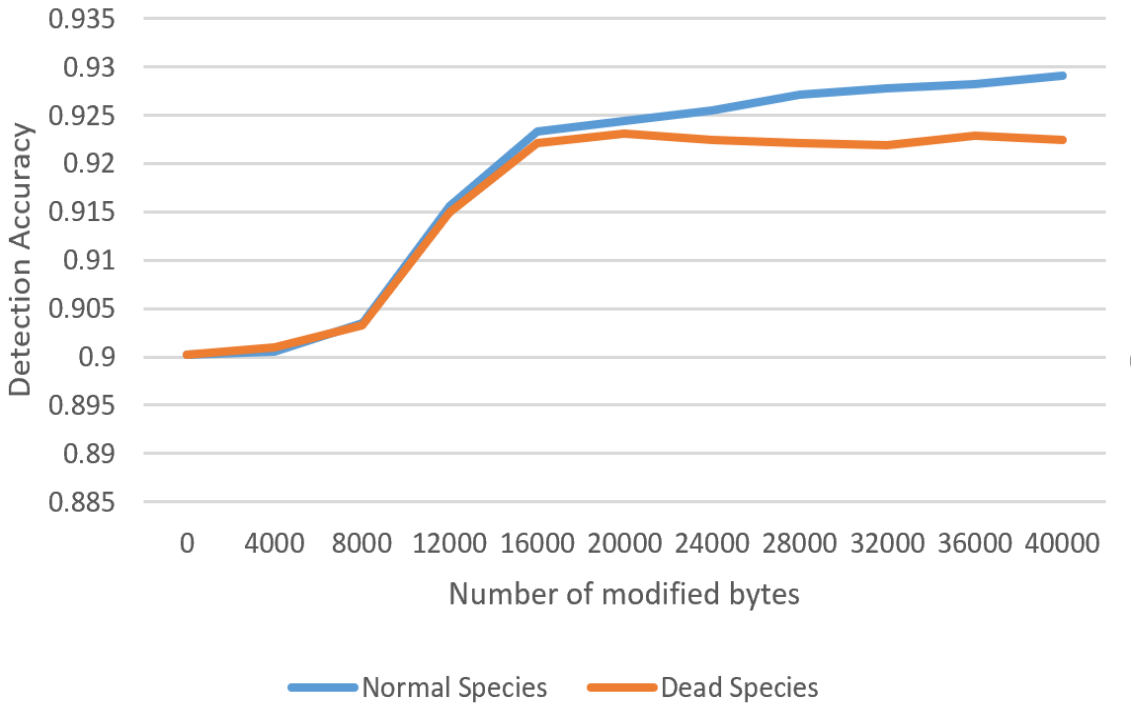

Figure 6: Example of dead species

This graph shows the performance difference between a dead species and normal species. The big gap of the two lines shows how dead species can affect the performance.

4.4 Overfitting to Development Set

During the experimental results evaluation phase, there were some abnormalities in the results. While most of the detection accuracy lies within the range of 93%, few results are above 95% accuracy. With further research, this phenomenon is likely an instance of overfitting to the development set: repetitively checking against the development set and tuning based on the (Blum et al. 2015). Therefore, it is necessary to construct a unseen test set to check the true accuracy. Due to the computation limit, the size of the unseen test set is only 10% of the normal development set. The following Table 1 shows the result.

Normal Development Set Acc	True Test set Acc	Accuracy Difference
95.43%	92.55%	2.88%
95.52%	92.71%	2.81%
92.89%	92.69%	0.2%
92.94%	92.79%	0.15%

Table 1: True Test Set Accuracy Difference Table

The first two rows are the abnormal results. It is clear that the accuracy drops significantly comparing to the other two accuracies. With this result, it is sufficient to conclude that evolutionary learning causes overfitting to development set and the evolved model should be checked against a test set that is not used in evolution.

Recently, there are a few literatures proposed different method to solve this development set overfitting issue (Feldman et al 2019, Russo et al 2019). These issues will be discussed specifically in Chapter 5.

4.5 Comparison to a Signature-based Approach

In order to compare MDEA to a popular signature based anti-virus program, I submitted some modified malwares to VirusTotal, a website that scan files with multiple anti-virus engines to detect malware.

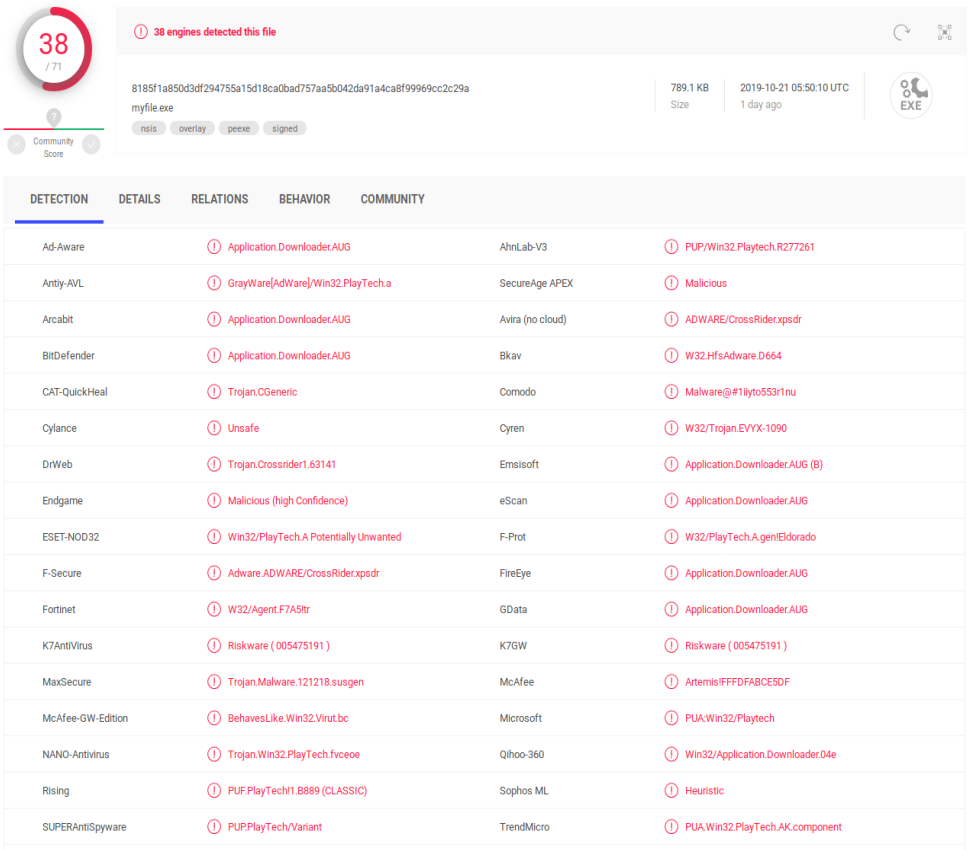

Figure 7: Original Malware Sample
The result from VirusTotal by submitting unmodified malware sample

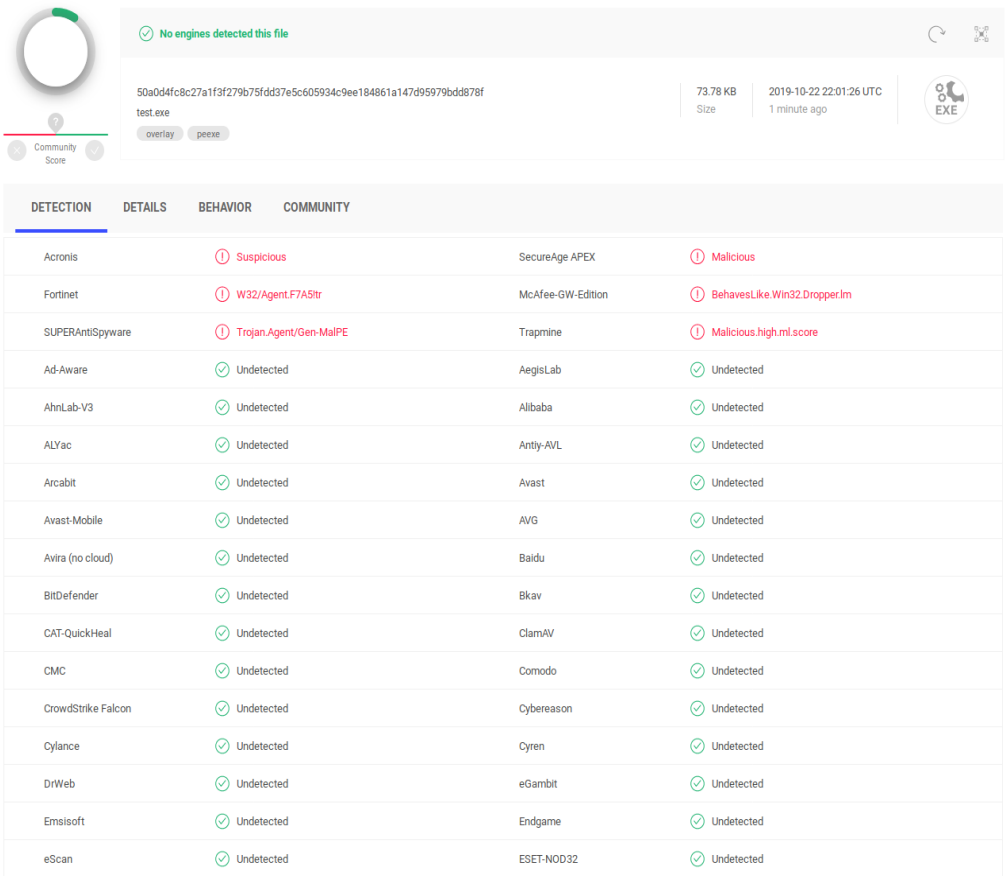

Figure 8: Break Optional Header Checksum
The result from VirusTotal by submitting a modified malware sample.

Figure 7 shows the detection result from VirusTotal for the original malware sample. It clearly detected the malware. Figure 8 shows the detection result for a modified malware where I performed a single action to break its optional header checksum. In this case, only 3 out of 71 engines classified the malware sample as malware. In the meantime, MDEA classified this file as malware with over 99% confidence. Although more experimental analysis is need, this simple example suggests

that Deep Neural network approach is superior to traditional signature approach to malware detection.

Chapter 5 Discussion and Future Work

This chapter will explain the design choices for MDEA. This section also evaluates how well the current approach worked and what could be improved in future work.

There are several design choices to be discussed here. The first one is the detection model, i.e. the MalConv network (Raff et al. 2017). Several malware detection methods were researched such as malware images (Nataraj et al. 2011), n-gram k nearest neighbor (Kaggle 2015), and LSTM sequence model (Yang et al. 2018) etc. After testing with the dataset, MalConv achieved the highest detection accuracy, therefore, MalConv is chosen as the detection network.

Another design choice is to use evolutionary algorithm (EA) as the optimization method to generate malware samples instead of GANs and reinforcement learning (RL). EA has several advantages compared to GAN and reinforcement learning. Existing GANs (GAN and its variants) suffer from training problems such as instability and mode collapse. EA can achieve a more stable training process. GANs usually employ a pre-defined adversarial objective function alternating training a generator and a discriminator (Wang et al. 2019). However, the action space cannot be simply expressed as a single adversarial objective function. EA solves this problem by evolving a population of different adversarial objective functions (different action sequences). Compared to RL, EA does not need to backpropagate the action weights and biases, which makes the code shorter and 2-3 times faster in practice. EA is also highly parallelizable compared to RL since it only requires individuals to communicate a few scalars between each other. Finally, EA is also more robust in the perspective of scaling (Salimans et al. 2017). It is very easy to extend the action space and other parameters to achieve a different learning outcome. With the above benefits, EA was chosen instead of GANs or RL.

The current setup of MDEA increases detection accuracy from 90% to 93 %. Even though the result is promising, there are still some problems that can be fixed. First, the evolutionary optimization still suffers from the dead species problem. The validation weights introduced in Section 4.3 only delay the occurrence of dead species instead of solving it. One possible solution is adding an age function in the evolutionary optimization process to mark the generations and study the relation between the dead species and age. With the age function, it is possible to find the optimal age for each irreversible actions. Then, the final solution will only be selected from the correct age generation.

The overfitting problem of evolutionary optimization is another field for future study. The current solution is to use a unseen test set (described in Section 4.4). However, this solution requires more data gathering time and can only be applied once. Recently, Feldman et al. (2019) showed the benefits that multiple classes have on the amount of overfitting caused by reusing development set. With their theory and method, I can extend the malware detection problem to multi-class malware classification problem, and the overfitting of development set can be reduced. The future plan is to relabel each malware samples into different malware classes and convert the detection problem into a multi-class classification problem to alleviate overfitting.

Another future plan is to expand the search space for the evolutionary optimization algorithm by defining more modification actions. The number of actions in the action space is one of the key factors to ensure the diversity and generality of the evolutionary optimization. However, because of the fragility and sensitivity of binary EXE code, it is very hard to create new modification action without changing the functionality of the program

The size and generality of the dataset can also be improved further. Since the search space is constrained by the diversity of malware types, adding different kinds of malware into the dataset can potentially improve the optimal detection accuracy. The input size of the detection model can also be further increased. Currently, MDEA takes in two million

bytes as the input. There are many malware samples in the dataset that have more than two million bytes and the extra bytes are cut off because of the length limit. The plan is to run MDEA with larger GPU memories, so that more data can be taken into the detection model, which may result in better performance. In addition, it is necessary to run an analysis in the future on how randomly modified malwares can influence the detection model. By conducting this validation, the significance of the evolutionary optimization algorithm can be verified.

Chapter 6 Conclusion

In this thesis, I proposed MDEA, an evolutionary adversarial malware detection model that combines neural networks with evolutionary optimization and generates attack samples to solve the evasion attack problem and to increase detection accuracy. An action space is introduced, which contains 10 different binary modification actions. The evolutionary algorithm evolves different action sequences by picking actions from the action space and then tests different action sequences against the detection model. After successfully evolving action sequences that bypass the detection model, all these action sequences will be applied to corresponding malware samples to form new training set for the detection model. By training the network with this cycle, detection accuracy increased from 90% to around 93 % even with limited computing power.

These results show that deep learning-based malware detection can defend against adversarial attacks and accuracy can be further improved by evolutionary learning. Evolutionary optimization provides generality and diversity that is difficult to achieve by other optimization algorithms.

I believe that MDEA represents a great method to solve adversarial attack on malware detection network.

Bibliography

A. Plonk and A. Carblanc. Malicious software (malware): Asecurity threat to the internet economy. 2008

Acquisti, A., & Grossklags, J. (n.d.). Privacy Attitudes and Privacy Behavior. *Economics of Information Security Advances in Information Security*, 165–178. doi: 10.1007/1-4020-8090-5_13

Altaher, A. (2016). An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features. *Neural Computing and Applications, 28*(12), 4147–4157. doi: 10.1007/s00521-016-2708-7

Anderson, H. S., Kharkar, A., & Filar, B. (n.d.). Evading Machine Learning Malware Detection. Retrieved from https://www.blackhat.com/docs/us-17/thursday/us-17-Anderson-Bot-Vs-Bot-Evading-Machine-Learning-Malware-Detection-wp.pdf.

Blum, Hardt, & Moritz. (2015, February 16). The Ladder: A Reliable Leaderboard for Machine Learning Competitions. Retrieved from https://arxiv.org/abs/1502.04585.

C. Beek, T. Dunton, S. Grobman, M. Karlton, N. Mini-hane, C. Palm, E. Peterson, R. Samani, C. Schmugar, R. Sims, D. Sommer, and B. Sun.Mcafeelab threats report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-jun-2018.pdf, 2018

Chaumette, S., Ly, O., & Tabary, R. (2011). Automated extraction of polymorphic virus signatures using abstract interpretation. *2011 5th International Conference on Network and System Security*. doi: 10.1109/icnss.2011.6059958

Chen, W., Tsangaratos, P., Ilia, I., Duan, Z., & Chen, X. (2019). Groundwater spring potential mapping using population-based evolutionary algorithms and data mining methods. *Science of The Total Environment*, *684*, 31–49. doi: 10.1016/j.scitotenv.2019.05.312

Cloonan, J. (2017, April 11). Advanced Malware Detection - Signatures vs. Behavior Analysis. Retrieved from https://www.infosecurity-magazine.com/opinions/malware-detection-signatures.

D. Gibert. Convolutional neural networks for malware classification. PhD thesis, MS Thesis, Dept. of Computer Science, UPC, 2016.

Elkhawas, A. I., & Abdelbaki, N. (2018). Malware Detection using Opcode Trigram Sequence with SVM. *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. doi: 10.23919/softcom.2018.8555738

Esteban, Aggarwal, Alok, Huang, Yanping, & V, Q. (2019, February 16). Regularized Evolution for Image Classifier Architecture Search. Retrieved from https://arxiv.org/abs/1802.01548.

Fan, C.-I., Hsiao, H.-W., Chou, C.-H., & Tseng, Y.-F. (2015). Malware Detection Systems Based on API Log Data Mining. *2015 IEEE 39th Annual Computer Software and Applications Conference*. doi: 10.1109/compsac.2015.241

Feldman, V., Frostig, R. & Hardt, M.. (2019). The advantages of multiple classes for reducing overfitting from test set reuse. Proceedings of the 36th International Conference on Machine Learning, in PMLR 97:1892-1900

Fraley, J. B., & Figueroa, M. (2016). Polymorphic malware detection using topological feature extraction with data mining. *SoutheastCon 2016*. doi: 10.1109/secon.2016.750668

Fujimoto, Y., Maruta, I., & Sugie, T. (2017). On a relationship between integral compensation and stochastic gradient descent. *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. doi: 10.23919/sice.2017.8105719

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative Adversarial Networks.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., & Mcdaniel, P. (2017). Adversarial Examples for Malware Detection. *Computer Security – ESORICS 2017 Lecture Notes in Computer Science*, 62–79. doi: 10.1007/978-3-319-66399-9_4

H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth. Learning to evade static pe machine learning malware models via reinforcement learning. arXiv preprintar Xiv:1801.08917, Jan. 2018

Hooman, O. M. J., Al-Rifaie, M. M., & Nicolaou, M. A. (2018). Deep Neuroevolution: Training Deep Neural Networks for False Alarm Detection in Intensive Care Units. *2018 26th European Signal Processing Conference (EUSIPCO)*. doi: 10.23919/eusipco.2018.8552944

J. Drew, M. Hahsler, and T. Moore. Polymorphic malware detection using sequence classification methods and ensembles. EURASIP Journal on Information Security, 2017(1):2,2017

K. Lab. Carbanak apt: The great bank robbery. *Securelist*, 2015

K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic Analysis of Malware Behavior using Machine Learning. (n.d.). Retrieved from http://www.covert.io/research-papers/security/Automatic Analysis of Malware Behavior using Machine Learning.pdf.2011

Lehman, J., Chen, J., Clune, J., & Stanley, K. O. (2018). Safe mutations for deep and recurrent neural networks through output gradients. *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO 18*. doi: 10.1145/3205455.3205473

Kaggle: Microsoft malware classification challenge (big2015). https://www.kaggle.com/c/malware-classification,2015.

Karras, Tero, Aila, Timo, Samuli, Lehtinen, & Jaakko. (2018, February 26). Progressive Growing of GANs for Improved Quality, Stability, and Variation. Retrieved from https://arxiv.org/abs/1710.10196.

Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., & Roli, F. (2018). Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. *2018 26th European Signal Processing Conference (EUSIPCO)*. doi: 10.23919/eusipco.2018.8553214

M. Narouei, M. Ahmadi, G. Giacinto, H. Takabi, and A. Sami. Dllminer: structural mining for malware detection. Security and Communication Networks, 8(18):3311–3322, 2015.

Maiorca, D., Biggio, B., & Giacinto, G. (2019). Towards Adversarial Malware Detection. *ACM Computing Surveys*, *52*(4), 1–36. doi: 10.1145/3332184

Martín, A., Menéndez, H. D., & Camacho, D. (2016). MOCDroid: multi-objective evolutionary classifier for Android malware detection. *Soft Computing*, *21*(24), 7405–7415. doi: 10.1007/s00500-016-2283-y

Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011). Malware images. *Proceedings of the 8th International Symposium on Visualization for Cyber Security - VizSec 11*. doi: 10.1145/2016904.2016908

Ojugo, A., & Eboka, A. O. (2019, July). Signature-Based Malware Detection Using Approximate Boyer Moore String Matching Algorithm. Retrieved November 29, 2019, from http://www.mecs-press.net/ijmsc/ijmsc-v5-n3/IJMSC-V5-N3-5.pdf.

Petroski, F., Madhavan, Conti, Edoardo, Joel, Stanley, … Lehman. (2018, April 20). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. Retrieved from https://arxiv.org/abs/1712.06567.

Podryabinkin, E. V., Tikhonov, E. V., Shapeev, A. V., & Oganov, A. R. (2019). Accelerating crystal structure prediction by machine-learning interatomic potentials with active learning. *Physical Review B*, *99*(6). doi: 10.1103/physrevb.99.064114

Preda, M. D., Christodorescu, M., Jha, S., & Debray, S. (2007). A semantics-based approach to malware detection. *ACM SIGPLAN Notices*, *42*(1), 377. doi: 10.1145/1190215.1190270

Raff, Edward, Barker, Jon, Sylvester, Jared, … Charles. (2017, October 25). Malware Detection by Eating a Whole EXE. Retrieved from https://arxiv.org/abs/1710.09435.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 4780–4789. doi: 10.1609/aaai.v33i01.33014780

Rehman, Z.-U., Khan, S. N., Muhammad, K., Lee, J. W., Lv, Z., Baik, S. W., … Mehmood, I. (2018). Machine learning-assisted signature and heuristic-based detection of malwares in Android devices. *Computers & Electrical Engineering*, *69*, 828–841. doi: 10.1016/j.compeleceng.2017.11.028

Ronco, C. C. D., & Benini, E. (2013). A Simplex-Crossover-Based Multi-Objective Evolutionary Algorithm. *Lecture Notes in Electrical Engineering IAENG Transactions on Engineering Technologies*, 583–598. doi: 10.1007/978-94-007-6818-5_41

Russo, Daniel, Zou, & James. (2019, October 8). How much does your data exploration overfit? Controlling bias via information usage. Retrieved from https://arxiv.org/abs/1511.05219.

Santos I. et al. (2010) Idea: Opcode-Sequence-Based Malware Detection. In: Massacci F., Wallach D., Zannone N. (eds) Engineering Secure Software and Systems. ESSoS 2010. Lecture Notes in Computer Science, vol 5965. Springer, Berlin, Heidelberg

Souri, A., & Hosseini2, R. (2018, January 12). A state-of-the-art survey of malware detection approaches using data mining techniques. Retrieved from https://hcis-journal.springeropen.com/articles/10.1186/s13673-018-0125-x.

Suciu, O., Coull, S. E., & Johns, J. (2019). Exploring Adversarial Examples in Malware Detection. *2019 IEEE Security and Privacy Workshops (SPW)*. doi: 10.1109/spw.2019.00015

T. Garfinkel, K. Adams, A. Warfield, and J. Franklin. Compatibility is not transparency: Vmm detection myths and realities, 2007

T. Raffetseder, C. Kruegel, and E. Kirda. Detecting system emulators. Lecture Notes in Computer Science Information Security, page 1–18, 2015

W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li. Dl4md: Adeep learning framework for intelligent malware detection. In Proceedings of the International Conference on Data Mining (DMIN), page 61. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

Wang, C, C. Xu, X. Yao and D. Tao, "Evolutionary Generative Adversarial Networks," in *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 6, pp. 921-934, Dec. 2019.
doi: 10.1109/TEVC.2019.2895748

Wang, P., & Wang, Y.-S. (2015). Malware behavioural detection and vaccine development by using a support vector model classifier. *Journal of Computer and System Sciences, 81*(6), 1012–1026. doi: 10.1016/j.jcss.2014.12.014

Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., & Patton, R. M. (2015). Optimizing deep learning hyper-parameters through an evolutionary algorithm. *Proceedings of the Workshop on Machine Learning in High-*

Performance Computing Environments - MLHPC 15. doi: 10.1145/2834892.2834896

Yuan, Z., Lu, Y., & Xue, Y. (2016). Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Science and Technology*, *21*(1), 114–123. doi: 10.1109/tst.2016.7399288

Yuxin, D., & Siyi, Z. (2017, July 25). Malware detection based on deep learning algorithm. Retrieved from https://link.springer.com/article/10.1007/s00521-017-3077-6.

Zakeri, M., Daneshgar, F. F., & Abbaspour, M. (2015). A static heuristic approach to detecting malware targets. *Security and Communication Networks*, *8*(17), 3015–3027. doi: 10.1002/sec.1228

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*. doi: 10.1109/iccv.2017.244

Zhu, Q.-Y., Qin, A. K., Suganthan, P. N., & Huang, G.-B. (2005, June 15). Evolutionary extreme learning machine. Retrieved from https://www.sciencedirect.com/science/article/pii/S0031320305001809.