ISSN 2278-3091
Volume 8, No.5, September - October 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse82852019.pdf
https://doi.org/10.30534/ijatcse/2019/82852019

Malware Detection through Machine Learning Techniques

Ahmed Amer[1], Normaziah A. Aziz[2]
[1] International Islamic University Malaysia, Malaysia, ahmed3amerai@gmail.com
[2] Department of Computer Science, International Islamic University Malaysia, Malaysia, naa@iium.edu.my

ABSTRACT

Malware attack is a never-ending cyber security issue. Since traditional approaches are less efficient in detecting newly appeared malware, researchers are applying machine learning methods. In this research we started by an overview of the domain and went over available malware datasets. Then we discussed disadvantages of traditional Anti-Malware methods and reviewed possible Machine Learning techniques used in this domain. A study on EMBER dataset has been made with an objective of improving the baseline Gradient Boosted Decision Tree model by optimizing its hyper-parameter and eliminating noisy features from the dataset. EMBER dataset consists of 1.1M observations of static features extracted from executable files. Our optimized model has achieved 99.38% accuracy with 0.004 false positive rate in 7 minutes running time. We conclude that Machine Learning techniques are practical to be applied as anti-malware solutions including for Zero-day attacks.

Key words: Artificial Intelligence, Machine Learning, Cyber Security, Malware Analysis, Smart Anti-Malware, GBDT Algorithm, Anti-virus.

1. INTRODUCTION

Malware writers continuously create new malware and spread them to attack their targets. Although traditional anti-malware techniques helped a lot in protecting users, they still not effective in detecting zero-day malware. Therefore, some companies depend on White-listing techniques "default deny" which is helpful in securing devices, but it has many limitations in accessibility what makes it suitable for organizations more than end-users. One of the popular solutions today is using machine learning techniques to train a model on large amount of malware considering specific features which enables prediction on whether the specific software is malware or benign using machine learning algorithm. There are many attempts to apply machine learning in malware detection domain starting in 2001 [1] whereby the researchers introduced the problem by saying, “Eight to ten malicious programs are created every day, and most cannot be accurately detected until signatures have been generated for them”. This statistic figure is alarming as it shows that an average of 700,000 malware samples created on a daily basis [2]. This huge number of malwares demands the necessity of automating the process of detection to minimize the risk. Fortunately, many researchers work to find solutions on this matter using Machine learning. Some companies have launched AI-based solutions in their Anti-malware products. Example of product solutions including {Endgame (MalwareScore), Cylance, ESET (NOD32), Windows Defender Anti-malware, and others} are using variety of machine learning techniques in their malware detection strategy which is a clear evidence of feasibility.

2. UNDERSTANDING MALWARE

The first step in any machine learning experiment is to get some domain knowledge which help understanding the data and accomplish the experiment. In this section we will briefly highlight common types of malware, malware analysis techniques, and available malware datasets. Which will demonstrate the flow of ideas and show the importance of the experiment.

2.1 Types of Malware

Malware is [3] any software that do something harmful or unwanted to the user. The harm varies in its negative implication - from annoying user such as “adware”, stealing confidential information “spyware” or “Keylogger” which store the wanted keyboard input typically passwords and send to attacker, and up to encrypting all user’s data and asking for money to give the key “ransomware”. Moreover, malware authors may aim to gain full access on victim machine by installing “backdoor” by which he can command directly on victim local system or even use it in his zombie army of machines “botnet” which gives attacker ability to implement “DDOS” and shutdown any service or server he don’t like. These are just some examples of different potential harm and different families of malware. To further understand how critical to prevent malware, imagine what a “backdoor” can do in a hospital systems and patients management. Malware can also be a fatal weapon in the cyberwar world; “Stuxnet” malware which attacked Iran's nuclear power plant in 2010 and “DDOS” 2007 cyberattacks on Estonia is a clear example. Malware can also be categorized based on the way of infecting victim’s machine whether it’s hidden within legitimate

program as a Trojan Horse or it spreads and replicates itself via internet Worm or it's just a virus using the traditional way such as email attachment or USB. The attacker's objective affects the nature of malware whether it takes a shotgun approach and designed to affect as many machines as possible or it's a targeted malware which targets a specific organization or victim and usually it's more sophisticated and harder to catch since in most cases it's a Zero-day malware. Zero-day means the malware has never been seen before and no malware analyst had deal with it or developed a signature to capture it. A recent study [4] found that 46% of all malware in 2017 are zero-day malware. Which indicate that it's a serious threat.

2.2 Malware analysis techniques

Malware analysis is one of the most important field in cyber security. Malware analyst job is to determine the features and function of a suspected malware, measure its impact or damage caused, and develop a signature which can be used to detect later appearance of this malware. The two main approaches to treat a malware is static analysis, and dynamic analysis which has been explained in "Practical malware analysis" book [3] as following:

Static Malware analysis is the process of analyzing the code or structure of a program to determine its function without running the code. It includes reverse-engineering the malware internal by loading the executable code into a disassembler and looking at the program instruction in order to discover what the program does. This process requires specialized knowledge of assembly language and windows operating system concepts.

Dynamic Malware analysis involves monitoring malware as it runs or examining the system after the malware has executed. In its advance version it uses debugger to examine the internal state of a running malicious software. Debugger is a piece of software or hardware used to examine the execution of another program.

2.3 Malware Datasets

Data is very valuable and important. Machine learning model cannot be built without suitable datasets. In the context of malware, the data is important and dangerous too.

Collecting Binary malware is doable however, since those malwares are executable then they may be harmful. Dealing with executable file needs analyst to set virtual machine and examine or extract features from malware cautiously. A quick visit to VirusShare.com show that they have 30,386,102 malware samples but of course "Access to the site is granted by invitation only".

In 2015 Microsoft published a big dataset for public access during "Microsoft Malware Classification Challenge" on Kaggle [5]. The dataset consists of 20K malicious samples from nine families available in binary format and in assembly (.asm) form which has been disassembled using IDA Pro disassembler. Many scientific papers have been done depending on this dataset however its size (400GB) has put a constraint to use it in our experiment and the dataset contains no benign files. The dataset has been cited by more than 50 research papers and thesis which tabulated in this paper [6].

There are some malware datasets of static and dynamic extracted features, but most of public are relatively small. A good choice to start our experiment with was ClaMP (Classification of Malware with PE headers) [7] which contains 5210 samples; 2722 are malware and rest are benign. ClaMP has been published in 2016 and consist of 69 extracted features including md5, size, entropy, fileInfo, VirusTotal report, file type, etc.

This choice has been changed after publishing EMBER (Endgame Malware BEnchmark for Research) on in 16 April 2018. EMBER is an open source malware dataset consist of 1.1M observations of static features extracted from PE files. The dataset has 400K malicious, 400K benign, 300K unlabeled [8] to be used in further studies to build semi-supervised model or other research purpose. More details of features and algorithm will be in the experiment section."

3. ANTI-MALWARE TECHNIQUES

Due to serious risk and negative impact of malware, different approaches are worked out to minimize and prevent spreading of malicious software. Each method has its pros and cons. The best way is to hybrid by leveraging advantages of each method. However, it's not always practical. In this section we will explain briefly some available Anti-malware techniques which will clarify why we choose machine learning for our solution.

3.1 Traditional Anti-malware Techniques

Signature Based used to be the first and most common method of detecting malware. It uses a database contains huge amount of signature of known malware. Whenever malware analyst analyses a malware and develop a signature, he will feed it into this database. There are many limitations in this approach including that recent malware can alter itself hence change its signature in what's known as polymorphism. The main limitation is the time factor which is very critical in malware analysis domain since each hour after spreading a malware may mean hundreds of infected systems. In general signature based mostly detect attacks carried out by user not author.

Behavior based is the dynamic way of detecting malware. It uses programmed rules to detect abnormal behavior. Some defined sequential of activity indicate the file is malicious such as attempt to discover sandbox or virtual machine or disabling security controls. Some disadvantage of this method is that it takes more time and resources to analyze the file than signature based which may affect the performance of the end user machine. Besides, this approach is helpful in detecting some enhanced versions of known malware which use similar steps or belong to the same family, but it cannot detect real zero-day malware which use creative ways or mix different attributes from different malware families.

Whitelisting is to prevent all software except those which explicitly allowed by system administrator in what known as "default-deny". However, that looks safe from confidentiality perspective it still has many availability limitations. This approach can be used in companies which limit staff access to some predefined software but not for end user who usually use software which is benign but not "trustworthy" enough. Besides if trusted program got some vulnerability it may be used to spread malware.

3.2 Machine Learning Techniques

Machine learning is the science of using algorithm to analyze data, learn patterns from it, and then use these patterns to predict or make a decision regarding extra samples of this data. Since behavior-based solutions follow rules and take explicit guidance from expert it may not be helpful in our case if the malware uses a new approach or belong to totally new family. Using machine learning has the advantage of detecting zero-day malware by analyzing huge amount of malware and benign file and let the algorithm learn the pattern which differentiate between them. Experts here only involve in choosing the most significant features and the machine will mimic their work. Considering PE files, we may categorize ML techniques into three main categories:

Static extracted features model in which features can be obtained without running the executable file. Some argue that static features are not that effective but for us we propose that it's the most suitable methodology for this domain because we mainly aim to detect maliciousness and according to [3] "Basic static analysis can confirm whether a file is malicious." so why we go to complexity while light simple solution exist. Many useful features can be extracted statically, and they are easier and cheaper to extract. The first feature is PE header which has been explained in detail in this paper [8]. It contains many useful information including type of machine, number of sections, number of symbols, size of the code, size of initialized and uninitialized data, address of the entry point, and data directories which provide pointers to the sections which include tables for exports, imports, resources, exceptions, debug information, certificate information, and relocation tables. Some research paper used entropy to measure percentage of randomness in the code which indicate obfuscation. As defined in [9] "Obfuscation techniques are used by developers to either protect legitimate intellectual property such as software or to make malware more difficult to understand".
In [10] they used suggestive strings such as URLs, IP addresses, names of special file system or Windows registry locations. This variety of available features give wide opportunity to researcher to use different machine learning algorithm. [11] has used eight different machine learning algorithm using API windows calls and among those algorithms SVM (normalized poly kernel) has performed the best with 0.932 ROC Area.

Dynamic extracted features model is similar to behavior-based approach since It needs virtual environment to run the executable file safely and it requires more time and resources than static models. According to [12] many dynamic features has been used in different research including dynamic analysis API calls, system calls, instruction traces, registry changes, writes memory, and others. This research [12] targeted to do a comparison between static feature, dynamic features, and hybrid models. Hidden Markov Models has been trained using two significant features: opcode sequences, and API windows call. Both features can be extracted statically by considering the overall program structure or dynamically by collecting the actual execution path taken when the program is traced. They used IDA Pro as disassembler and debugger to generate assembly file from which opcode sequences and API calls can be extracted. In case of opcode sequences, they have discarded all operands, labels, directives, etc., and only retain the mnemonic opcode {call, push, call, add, etc.}. For API calls they have collected API call names and discarded the arguments. Many API calls has been mentioned including {CreateFile, OpenProcessToken, AdjustTokenPrivileges, SetNamedSecurityInfo, GetComputerName, QueryProcessInformation, DeleteFile}. The results of this experiment indicate that for API calls and opcode sequences, a fully dynamic strategy is generally the most effective approach.

Neural Network is well referred ML Algorithm that is applied in many domains varying from Arabic Handwriting Recognition [15] to Wireless Ad-Hoc Networks prevention [16]. Featureless model using end-to-end deep learning neural network is one of the most recent research published on 25 Oct 2017 [13]. It doesn't depend on neither static nor dynamic features instead it uses raw byte sequences what explain the title of the research paper "Malware Detection by Eating a Whole EXE". They used a very huge amount of data started with around 0.5M and then increased to 2,011,786 binaries samples which nearly half of them are malware. The model treats each byte as a unit in a sequence which leads to produce the first network architecture with the ability to process raw byte sequence of over two million steps. The main target of this research is to minimize the use of domain knowledge and explore how effectively the problem can be

solved without specifying any such information. The experiment successfully got 94% accuracy with 98.1% AUC. The main limitation of this approach is the computational constraint due to the extreme memory use of their architecture. It takes two months to train the model on those 2 million observations using data parallelism across 8 GPUs.

4. EXPERIMENT CONDUCTED

After understanding the theoretical techniques, it is time to demonstrate with a practical experiment. It is always advisable to continue from where others stop rather than starting from scratch. Therefore, we will study and improve the model published with EMBER Dataset.

The experiment has been conducted using Dell G3 15 core i7 eighth generation laptop. We started by training the baseline model on the 600K labelled data. According to [8] it took 20 hours to vectorize the raw features into model features and 3 hours to train the model using 2015 MacBook Pro i7 (fourth generation). Surprisingly, it took only 6 minutes to vectorize and 5 minutes to run the model on our local machine. We did some research to find out the reason behind this huge gap considering that CPU processing power are similar since both are core i7. We checked whether there is any contribution of GPU in Dell machine while running the code _since it has GTX 1050 Ti GPU which is more powerful_ but there were no any GPU processes. The main difference is that Mac machine is fourth generation which may be up to 3.4 Ghz while Dell machine is eighth generation which may be up to 4.1 Ghz and is consist of 6-cores. This speed of running made the experiment much easier for us since we don't need to wait for a day with every parameter change.

4.1 Data Description

EMBER dataset consists of eight groups of raw features that include parsed features, histograms, and counts of strings [8]. String information are simple statistics about printable strings and special string such as those which start with {C:\, http://, HKEY_, MZ, etc.} since each one of those string may indicate maliciousness for example a file used too many registry key {HKEY_} may be suspicious. They provide statistical summary instead of raw strings to mitigate privacy concerns that may exist for some benign files. Histograms involve Byte histogram which is the count of each byte value within the file, and Byte-entropy histogram which is the joint distribution of entropy and byte values. Eventually, parsed features are five groups: general file information, header information, imported functions, exported functions, and section information. Each parsed group consist of useful features for instance size, resources, and whether the file has a debug section belong to general file information group while timestamp, and target machine is header information. Complete details of each feature exist in [8].

After extracting raw features, they transform them to model (vectorized) features. Model features represent a feature matrix of fixed size used for training a model, representing a numerical summary of the raw features, wherein strings, imported names, exported names, etc., are captured using the feature hashing trick [17].

4.2 GBDT Baseline Model

GBDT is a lightweight model therefore it's suitable for end user machine. Besides, it requires less processing power when comparing with the other machine learning algorithm. Considering that classifying new sample after building the decision tree is fast and easy. EMBER dataset has been tested by training a baseline gradient-boosted decision tree (GBDT) model using LightGBM with default parameters (100 trees, 31 leaves per tree) [8]. The baseline model achieves 98.2% detection rate with approximately 1% false positive rate.

4.3 Optimize Hyper-Parameters

We wrote down a script to test the accuracy of the lightGBM GBDT model with default parameters using 200K observations and it was 98.6% with 0.01 False rate using 0.5 as a threshold. This percentage may consider more than enough in many domains but in Anti-malware field it's a real risk. For example, out of this small dataset this percentage means 1667 malware has successfully reach the target and of course numbers of samples are much higher. After studying deeply LightGBM documentation we decided which parameters may be optimized to improve the results. The first parameter we choose is min_data_in_leaf which prune the tree by specifying minimum number of records can be in one leaf which helps to reduce overfitting. By increasing this parameter from 20 to 50 we managed to slightly increase accuracy and significantly speed up the the model. Then we increased num_leaves in each tree from 31 to 100 which means increasing complexity of the model. Increasing complexity has increased accuracy as expected. Finally, we attempted to choose the best number of boosted trees. We separated random 600 samples from the training set to use it as a validation set, then we assign 500 to num_boost_round parameter and put early_stopping_rounds = 10. That will force model to stop if 10 rounds passed without improving classifying those 600 valid samples. We found that optimal number of boosting rounds are 192.

4.4 Eliminate Noisy Features

After vectorizing the dataset, it consists of 2291 feature which is a big number and may affect the performance or the computation power required. We started by checking the

importance of each feature using LightGBM function and the result was better than what we expected. We found that 761 of vectorized feature has zero impact on the model. The impact of the features is shown in Figure 1.

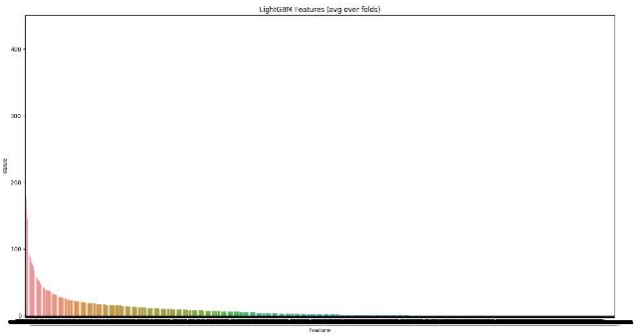

Figure 1: Plotting importance of features of lightGBM model

We eliminated those features and retrained the model that gives us almost same accuracy but by less time for training since number of features now is 1530. The disadvantage of checking importance of vectorized features after training is that it is not interpretable since we don't know which raw feature produce those irrelevant vectorized features. We made an educated guessing depending on what has been suggested in [16] that entropy has low impact on detecting malware since programmers use obfuscation to protect their legitimate programs not only malware author. We tried to test this assumption by eliminating ByteEntropy() from raw features and it looks valid since the retrained model gave almost same accuracy with only 2035 features. Finally, we merged both techniques by rechecking feature importance after eliminating ByteEntropy() and avoid any further feature with zero impact on the model. We end with only 1338 vectorized feature.

5. RESULTS

Our optimized model has achieved 99.38% accuracy with 0.004 false positive rate and the same threshold of 0.5. The most significant advantage in this model is that it takes around 7 minutes for training 600K observation with 2291 features and takes less than a second for testing 200K samples which is leads to very quick detection of malware. Moreover, by eliminating noisy features we managed to reduce training time to 4.5 minutes with 1338 feature without significantly affecting performance which becomes 99.31%. The final model has been tested using dozens of executable files and it works correctly. It classifies legitimate programs such as Code Blocks, Burp Suite, and others as benign. It detects tens of random samples from various malware types such as spam, CoinMiner, Downloader, Ransomware etc. It easily captured the most common malware "wannacry". Finally, as an evidence that this model can detect new malware, we tested it using "grandcrab" which is one of the most common ransomwares of this year (2018). The model successfully detects a sample of "grandcrab" which uploaded only 15 days ago on GitHub.

Table 1: Summary of results with 0.5 threshold on local machine (Dell G3)

	Featureless model (DLNN)	EMBER Model (Published Paper)	EMBER Model (Local Machine)	Optimize Hyper-Parameters	Eliminate Noisy Features
Accuracy	94%	98.2%	98.6%	99.38%	99.31%
FP rate	-	0.01	0.0112	0.0042	0.0045
Time	2 (months)	23 (hour)	11 (minute)	7 (minute)	4.5 (minute)

6. CONCLUSION

Our next stage is to further improve the model by using other 300K unlabeled samples in building a semi-supervised model and compare it with the current GBDT model. Then to put theory in action we will work on developing more precise and efficient anti-malware. The basic architecture of the product is to mix whitelisting approach with machine learning in order to reduce the amount of analyzed suspicious files hence improve the performance. The model will only examine suspicious samples which are not in neither trusted whitelist nor blocked blacklist. If the model detects a file as malware it will pass it to other multi-classification malware model which predict its family and display it to the user who take the final decision whether to delete, ignore, or trust this file. In conclusion, machine learning techniques are practical to be applied in anti-malware industry and protect users to some extent. Therefore, mitigation can be made against malware attack before users become the victim.

REFERENCES

[1] M.G. Schultz, E. Eskin, F. Zadok and S.J. Stolfo. **Data Mining Methods for Detection of New Malicious Executables**, in Proc. 2001 IEEE Symposium on Security and Privacy, pp. 38-49, 2001.

[2] Chad Skipper, Carl Gottlieb, and Lawrence C. Miller. **Treating Malware as a Data Problem**. In the book "Next-Generation Anti-Malware Testing For Dummies", Published by John Wiley & Sons, Inc., Hoboken, New Jersey, Chapter I, pp. 1 - 22, 2018.

[3] Michael Sikorski and Andrew Honig. **Practical Malware Analysis**. Published by No Starch Press, Inc.38 Ringold Street, San Francisco, CA 9410, 2012.

[4] **Internet Security Report**, 2017. WatchGuard's Threat Lab, available at https://www.watchguard.com/wgrd-resource-center security-report.

[5] **Microsoft Malware Classification Challenge**, Kaggle, 2015. [Online], available at https://www.kaggle.com/c/malware-classification.

[6] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov and Mansour Ahmadi. **Microsoft Malware Classification Challenge**, arXiv, 1802.10135, 2018.

[7] **ClaMP (Classification of Malware with PE headers)**" 2016. Github, available at https://github.com/urwithajit9/ClaMP.

[8] Hyrum S. Anderson and Phil Roth. **EMBER: An Open Dataset for Training Static PE Malware**, arXiv, 1804.04637,2018.

[9] T. Morgenstern. **Malware Terms for Non-Techies Code Entropy**, 2016. [Online], available at https://www.cyberbit.com/blog/endpoint-security/ malware-terms-code-entropy/.

[10] **Hunting for Malware with Machine Learning.** EndGame, 2016.

[11] Mamoun Alazab, Sitalakshmi Venkatraman, Paul Watters, and Moutaz Alazab. **Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures**, in Proc. of 9-th Australasian Data Mining Conf. (AusDM'11), vol. 121, pp. 171-182, 2011.

[12] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp. **A comparison of static, dynamic, and hybrid analysis for malware detection**, Journal of Computer Virology and Hacking Techniques, vol. 13, no. 1, pp. 1–12, Dec. 2015. https://doi.org/10.1007/s11416-015-0261-z

[13] Raff, Edward, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro and Charles K. Nicholas. **Malware Detection by Eating a Whole EXE**, arXiv, abs/1710.09435, 2017

[14] Jonathan Woodbridge. **The Making of MalwareScore**, 2017. [Online], available at https://www.youtube.com/watch?v=KZBohtfwhIY.

[15] Ravi Tomar and Yogesh Awasthi. **Prevention Techniques Employed In Wireless Ad-Hoc Networks**. International Journal of Advanced Trends in Computer Science and Engineering, Vol. 8, No.1.2, 2019. https://doi.org/10.1109/ICOASE.2019.8723725

[16] Manal Abdullah, Afnan Agal, Mariam Alharthi and Mariam Alrashidi. **Arabic Handwriting Recognition Model based on**, International Journal of Advanced Trends in Computer Science and Engineering, Vol. 8, No.1.1 2019.

[17] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford and Alex Smola. **Feature Hashing for Large Scale Multitask Learning**, Proc. of the 26th Annual International Conf. on Machine Learning (ICML '09), pp.1113-1120, 2009 https://doi.org/10.1145/1553374.1553516.