Transforming Malware Behavioural Dataset for Deep Denoising Autoencoders

Mohd Razif Bin Shamsuddin, Fakariah Hani Hj Mohd Ali, Mohd Shahril Bin Zainol Abidin
Fakulti Sains Komputer & Matematik, Universiti Teknologi MARA, Malaysia

Corresponding author’s e-mail address: razif@fskm.uitm.edu.my

Abstract. This research is a part of a major research on automation of malware identification using Deep Denoising Autoencoders. Malicious software, or in short called malware refers to any software designed to cause damage to a single computer, server, or computer network. This malware term includes all kind of malicious software such as computer virus and spyware. All these malicious malware behaviour is monitored, logged and recorded using a cuckoo sandbox with the help of an x86 hosted supervisor software. The intent of recording the malware behaviour is to understand the pattern of behaviour of each known malware family. This collected data will be further trained to a Deep Denoising Autoencoders to automate the identification process of new malware within the identified malware families. However, the raw behaviour data is not suitable for an optimum training process. This paper will discuss the process of transforming the text based behavioural dataset to a more suitable dataset for deep learning purposes. At the end of the research a cleaned bit string format that should represent a unique malware behaviour will be produced.

1. Introduction
This paper will explain the process of transforming malware executable dataset into bit-string format that is suitable for Machine Learning input. Explanation and appropriate figure are shown to give clear look on each process that will be implemented throughout this research paper.

As the malware dataset consists only binary executables (raw format), several processing steps need to be done to gain useful data representation from every each of the malware samples. Behaviors extraction technique was chosen as this shown to be very reliable to generalize malware and their harmful actions. Moreover, generalization of malware through its behaviors proved to be a popular method among literature, and it allows Machine Learning to predict new unseen malware even if the malware has been modified to evade traditional detection system [1][7].
2. Malware Family Scopes

The scopes of this study comprised the following families of malware:

2.1. Cerber (Ransomware)
Cerber ransomware was first seen in the mid of 2015. Cerber is categorized as a Ransomware-as-a-Service (RaaS), where the developer will rent the ransomware functionality with split profit between the buyers and the sellers. Among many others, it is a type of ransomware that displays ransom message after it has infected the user’s computer. It encrypts files and adds an extension “.cerber” which means that particular files have been encrypted. Furthermore, like any other high-profile malware, it has a “malware factory” form; which means it uses code obfuscation technique that can automatically generate large volumes of unique-hash malware variants from the original malware code [2][4].

2.2. Cryptowall (ransomware)
CryptoWall is a ransomware family that is designed to use sophisticated encryption (RSA-2048) algorithm to make files inaccessible on the targeted computers. Malware researchers spotted the first version of ransomware in 2013. Since then, crypto-virus was updated several times, with additional functionalities (such as stealth mode and deletion of volume shadow copies) have been added in order to avoid ongoing antivirus detections [2][4].

2.3. GandCarb (ransomware)
GandCrab ransomware was discovered as Ransomware-as-a-Service (RaaS) near the end of January 2018 and soon became the most popular and widely used ransomware of the year. GandCrab was potentially born from the need to monetize encrypted data from organizations further by customizing the ransom notes based on the victim's profile and the type of encrypted data. As a result, demand for GandCrab ransom could be between $600 and $700,000 per victim. This behavioral change has probably led to a significant increase in revenue for cybercriminals, especially since they started to deliver it as-a-service [2][4].

2.4. Petya (ransomware)
Petya is a ransomware encryption family that discovered for the first time in March 2016. The malware targets Microsoft Windows systems which it infects the master boot record to execute a payload that encrypts the file system table on a hard drive and prevents Windows from booting. Petya has evolved in several ways. One such example can be seen in June 2017, where a new variant of Petya (dubbed as NotPetya by researchers) was used at a global scale cyberattack and could propagates itself via computer worm technique [2][4].

2.5. Sality (computer botnet)
As the only non-ransomware type malware inside this project scope, Sality surely has its own reputation. Sality is a family of malware which infects file on Microsoft Windows systems and made its first debut in 2003. It has gone through several evolutions and advancement over the years to become a dynamic, enduring and full-featured form of malicious code. Because of its continued development and capabilities, Sality is one of the most complex and formidable forms of malware to date [4][5].

2.6. WannaCrypt (ransomware)
WannaCrypt is a ransomware computer-worm that targets the family of Microsoft Windows operating systems. It was first discovered on May 2017, targeting Microsoft Windows system and has affected more than 230,000 computers in 48 hours to over 150 countries. The ransomware demanded payment to unlock the infected system. WannaCrypt caused ambulances to be diverted, shutting down nonemergency services, nabbed machines at Telefonica in Spain, and affected airlines and ministry [2].
3. Cuckoo Sandbox to Record Malware Behavioural data

To obtain the malware behaviours in the form of text data, computer sandbox has been chosen to articulate this task. A sandbox is a computer controlled-environment in which it can monitor program behaviours through several logging mechanisms such as API tracing, system-call logger and man-in-the-middle network traffic recorder. Among every sandbox that has been released into the internet, Cuckoo Sandbox is the one that shown to be the most potential, as it is constantly get updated with new features over time. Furthermore, Cuckoo Sandbox could monitor executable behaviours until the kernel level, where it is the lowest point of every operating system. Using this feature, it helps computer sandbox to monitor any executable behaviours even if the malware tries to hide by injecting into any kernel modules [3].

To achieve these feet, Cuckoo Sandbox has been set up together with the help of an x86 hosted supervisor software. Oracle VirtualBox has been chosen as it is open source and has great support from the developers of Cuckoo Sandbox themselves. To fully utilize host resources for more efficient and less consuming analysis, a cluster of three sandboxes have been set up. These three sandboxes have the hardware specifications which are Windows 7 x64 bit and 2GB of memory access. An operating system in which Windows 7 has been chosen as this OS is popular among malware analysts and can run most of modern malware without much hassle as current Windows version put many protections in order to thwart malware infections.

![Figure 2. A cluster of three sandboxes in which all named as cuckoo prefix](image)

In order to let Cuckoo Sandbox uses these three sandboxes, further configuration files have been written to point out which virtual machines that it should use when processing and analyzing malware. Using ‘cuckoo’ command-line interface program, all malware has been submitted into Cuckoo Sandbox’s system, in which it uses queue technique to process malware which has been submitted first into the system.
By using these clustering and sandboxing techniques, Cuckoo Sandbox was able to analyze 70 executables in one-hour timeframe, speeding up the time required to analyze the malware and its behaviors. For every executable that has been analyzed by Cuckoo Sandbox, it will produce a report file containing the executable behaviors while it is running inside the computer sandbox. The report produced has the format of JavaScript Object Notation (JSON), in which this format can be easily parsed by common JSON parser.

However, as the recorded behaviors report file is in the JSON format, there is still another preprocessing step that needs to be done. The problem with this report file is that for every malware recorded behaviors, the output of the report file is going to be varied, depending on how much behaviors the malware spits out. This data representation cannot be fed into Machine Learning algorithms as most
of them expected the size of the input to be fixed. One such example is neural network: it requires the input to be a vector of fixed-size or otherwise cannot proceed because the different number of neurons in the input layer does not match with input vector size.

4. Using n-Gram Extraction Method to Produce Fixed Sized Binary Files for Machine Learning

As explained in the previous section, raw malware data representation cannot be fed into Machine Learning algorithms as it is. A raw malware data behaviour contains many text based information such that is not suitable for Machine Learning training process. Thus, a method to transforms varying sized behavioural report file into sets of fixed-size binary string is needed. In order to process the datasets, a method from Natural Language Processing (NLP) is used to complete this task. One such method is by using n-gram extraction. The simplest form of n-gram extraction is 1-gram extraction (or called as unigram). 1-gram is an algorithm which finds the most frequent words in text sample [4][5][6]. Below are the steps taken for the unigram’s data transformation process.

4.1. Step 1, For all report files, count all occurrences of unigram words.
Treat all reported JSON file as normal text file. Split the text into unigrams by doing text tokenization such as splitting by spaces and removing word which size is less than or equals to three. Build a dictionary which for every unigram, count its occurrences.

4.2. Step 2, Sort all words by their frequency.
Sort the dictionary with respect to their occurrences, using Python’s sort function in which it uses Tim sort algorithm to speed up the sorting operation.

4.3. Step 3, Choose top 10,000 unigram words.
Split the dictionary and choose top 10,000 most frequent unigrams. As the dictionary has first been sorted out, this operation can be done quickly.

4.4. Step 4, Map each executable’s unigrams with top 10,000 unigrams.
By having top 10,000 unigrams, bit-string vector can be created by mapping every unigram inside each executable and mark it 1 if the unigram exists or 0 if otherwise.

![Figure 5. Example of bit-string conversion based on samples’ unigrams.](image)

All the processes listed above have been done with the help of Python programming languages. The Python script will parse all reports, tokenize every text, and find top 10,000 most occurrences unigram. Developing the script is technically challenging as with every report averagely sized more than 50 megabytes, it is time-consuming to process all 6,000 executable samples. Many optimizations have been
applied such as using Python’s dictionary data structure which has a great algorithmic complexity to speed up the occurrences counting process.

Once the script has done processing, for every executable sample it will generate a bit string representation of ones and zeros as its data. These ones and zeros are treated as the executable’s features and will have fixed-size of 10,000 in length. By having this representation of input, machine learning algorithms such as neural network will not have a problem dealing with it for learning purposes.

5. Data Pre-Processing Result
As explained earlier, several methods have been applied to the malware samples to convert the file executable representation into features that can be understood by machine learning algorithm.

The first process manage to capture all behavioural data from the malware executable samples. Using 6,000 executable samples, all of them have been submitted into Cuckoo Sandbox for dynamic analysis. It took roughly one minute for each executable sample, thus approximately 100 hours of processing time to successfully examine the malwares and capturing their behaviours.

However, as mentioned previously in Section 3 where each JSON report file going to be varied depending on the behaviors spitted out by the malware thus incompatible with machine learning that expecting input data to be all the same. To solve this problem, pre-processing steps need to be applied for all these JSON behaviors file. Each figure should have a brief caption describing it and, if necessary, a key to interpret the various lines and symbols on the figure.

![Figure 6. List of behaviours file produced by Cuckoo Sandbox](image-url)

By applying Python script coded with techniques mentioned in Section 3.6, every malware’s behaviors have been sampled, unigram extracted, and top 10,000 unigrams chosen based on their repetition frequency. These top 10,000 unigrams are then marked as a feature-map, where each of malware behaviors’ unigrams have been mapped to either one or zero based on their availability in the
top frequent unigrams. With having these transformed files, initial visual differences can be done to have an initial look on how our data has been transformed. Below is an example of two completely different malwares and their families on the bit-level representation. Figure 7 shows data visualization that give an idea about their differences and how machine learning algorithms such as deep Denoising Autoencoders can exploit this non-similarity to create generalized form of malware signatures which can represent these bit-string behaviors on much lower dimensions.

Figure 7. Bit-string visualization of two malware samples on bit-level

Acknowledgments
The authors are grateful to the sponsor of the research Ministry of Education, Universiti Teknologi MARA and the Research Management Centre (RMC) UiTM Shah Alam for the support under the national Fundamental Research Grant Scheme FRGS/1/2019/ICT02/UITM/02/9.

References
[1] Rieck K, Trinius P, Willems C, & Holz_aff2n3 T. (2011). Automatic Analysis of Malware Behavior Using Machine Learning. *Journal of Computer Security*, 19(4), pp 639–668.
[2] Nieuwenhuizen, D. (2017). A behavioural-based approach to ransomware detection. https://labs.mwrinfosecurity.com/assets/resourceFiles/mwri-behavioural-ransomware-detection-2017-04-5.pdf
[3] Claudio Guarnieri. (n.d.). Cuckoo Sandbox - Automated Malware Analysis. Retrieved June 22, 2018, from https://cuckoosandbox.org/blog/206-interim-release
[4] Daniele U,Leonardo A, Roberto B. (2019) Survey Of Machine Learning Techniques for Malware Analysis. *Computers & Security*. Volume 81, pp 123-147.
[5] Şerif B Mehmet B Y, Can Y A. (2019). A multi-dimensional machine learning approach to predict advanced malware. *Computer Networks*. Volume 160, pp 118-129.
[6] Sen C, Minhui X, Lingling F, Shuang H, Lihua X, Haojin Z, Bo L. (2018). Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Computers & Security*. Volume 73, pp 326-344.
[7] Wei Z, Feng G. (2019) multi-level deep learning system for malware detection. *Expert Systems with Applications*. Volume 133, pp 151-162.