Machine Learning Against Malwares: Analysis On Windows Systems Based in Pattern Recognition Algorithms

Victor Picinin Veloso Senna[1], Gustavo Alves Fernandes[2], Antônio Ricardo Leocádio Gomes[3], Luiz Melk de Carvalho[4], Flávio Henrique Batista de Souza[5]*

Centro Universitário de Belo Horizonte – UNIBH, Brazil

flabasouza@yahoo.com.br

Abstract

In 2020, due to the pandemic by COVID-19, the use of digital resources increased considerably, unprecedented for providers and developers of digital services. An issue that accompanies such a process is the security and identification of malware, which spreads at a speed never seen before. Thus, the objective of this paper is to demonstrate an evaluation, based on an experimental process, of pattern recognition algorithms (Multilayer Perceptron-MLP; Naive Bayes, K-Nearest Neighbor and Decision Tree) for the recognition of malware patterns in Portable Executable (PE) Files Structures. As a methodology for experimentation, three steps were taken: collection of malware samples in a repository known as Virusshare (around 285 malware samples - from 6 different families - for Windows® from 2012 to 2019) and PE files (initially 549 were randomly selected PE files from "System32" and "Program and Files") totaling 834 samples; 2,500 features were defined to compose the analysis dataset; finally, the test methodologies were defined according to the pattern recognition algorithm (e.g. in MLP, the number of hidden layers and the number of neurons in these hidden layers were varied). As a result, after 100 execution of each algorithm configuration, an accuracy ranging from 64.5% (Naive Bayes) to 95% (MLP) was obtained.

Keywords: Malwares, Machine Learning, Pattern Recognition, Windows® Systems, Detection Methods

1. Introduction

The earliest documented computer viruses began to appear in the early 1970s. Historians often credit the “Creeper Worm,” an experimental self-replicating program written by Bob Thomas at BBN Technologies as being the first virus. Creeper gained access via the ARPANET and copied itself to remote systems where it displayed the message: “I’m the creeper, catch me if you can!” (Alexandrou, 2019). According to Rothrock (2018) the firsts malwares were simple, relying in floppy disks to spread, carrying itself out from computer to computer. With the widespread use and development of the internet, malwares started to quickly adapt and take advantage of the new communication technology.

Years have passed and now the malwares have ill intent, in most cases financial gain, like stealing bank details or credit cards from the victims. In 2017 an old type o virus came into attention with the WannaCry attack, the ransomwares. According to Savage et al. (2015) this type of malware aims to encrypt all the data in the machine and it requests the victim to transfer a certain amount of money to get the decryption key to recover the encrypted data.

Parallel to the context of the impact of harmful tools on the functioning of digital systems, it is important to consider the importance of protecting digital systems in periods of high accessibility demand by users. Thus, the context of the COVID-19 pandemic enhances the need for systems to be protected, since the world population has massively migrated its activities to the internet. This relevant point exponentially increases the demand for data security studies, since the world population has become totally dependent on connections from cloud systems and software, due to the imposed social isolation scenario. The fact that the world has become totally dependent on communication technologies has not exempted it from attacks and intrusion attempts (Lallie et al., 2020; Santos et al., 2020).

The main objective of this research is to create a tool to extract the features from all the files in a computer system to build a dataset and propose a training method based on static analysis for the detection of malware software. As a specific objective this research aims to measure the assertiveness and efficiency of machine learning algorithms to detect malicious entities, based on the imports and exports of a Portable Executable (PE) Files Structures Header. Such an attempt seeks to allow information and calculations to be inserted into third-party software (or through a Software Development KIT - SDK) so that the computer does not demand a traditional anti-malware solution and its list of antiviruses.

2. Material and Methods

2.1 Theorical Background

2.1.1 Malware Families

According to Chaurasia (2018) a malware, or “malicious software,” is a common term that describes any malicious program or code that is harmful to digital systems. Hostile, and intrusive, the malware seeks to invade, damage, or disable computers, computer systems, networks, tablets, and mobile devices, often by taking partial control over the operation of the device. There are a number of classifications of malware types such as Trojan Horse, Adware, Rootkits, Backdoor, Keylogger and Ransomware.

2.1.2 Detection Methods

The traditional antivirus detection methods can be classified as signature-based, behavior-based or heuristics-based. This research is based in signature-based and behavior-based, where (Idika & Mathur, 2007):

- **Signature-Based:** Like every person has a signature the malware has one too. Based on that, the antivirus software is constantly checking the system for suspicious signatures. This method of detection relies on the antivirus provider to keep their virus definition up to date.
- **Behavior-Based:** Some antivirus software have some level of behavior analysis to detect some anomalies. The most common is the heuristic.

2.1.3 Malware Analysis

According to Tahir (2017) malware analysis is a step towards malware detection. To detect malwares, firstly it is necessary to analyze how the malware performs its function and what is the purpose behind the malware development.

By having a deep understanding about the malware development it is possible for the developers to create malware detectors to implement the defensive functionality. Malware analysis techniques are divided into three categories based on time and technique to do the analysis. A comparison between Static and Dynamic Analysis was performed by Tahir (2017) and shown in Table 1.

Table 1: Analysis comparison

Static Analysis	Dynamic Analysis
Fast and safe	Time consuming and venerable
Good in analyzing multipath malware	Difficult to analyze the multipath malware
Cannot analyze obfuscated and polymorphic malware	Cannot analyze obfuscated and polymorphic malware
Low level of false positive (Accuracy is high)	High level of false positive (Accuracy is low)

Source - Tahir, 2017, Pg 26

According to Prasad et Al. (2016) a “Static analysis” can be viewed as “reading” the source code of the malware and trying to infer the behavioral properties of the file. Static analysis can include various techniques. Chumachenko (2017) classifies static analysis in 5 techniques:

- **File Format Inspection:** A file metadata can provide useful information. For example, Windows PE (portable executable) files can provide much information on compile time, imported and exported functions.
- **String Extraction:** Refers to the examination of the software output (e.g. status or error messages) and inferring information about the malware operation.
- **Fingerprinting:** This method includes cryptographic hash computation, finding the environmental artifacts, such as hard coded username, filename, registry strings.

- **Anti-Virus Scanning:** If the inspected file is a well-known malware, most likely all anti-virus scanners will be able to detect it. Although it might seem irrelevant, this way of detection is often used by AV vendors or sandboxes to “confirm” their results.
- **Disassembly:** Refers to reversing the machine code to assembly language and inferring the software logic and intentions. This is the most common and reliable method of static analysis.

Static analysis often relies on certain tools. Beyond the simple analysis, they can provide information on protection techniques used by malware. The main advantage of static analysis is the ability to discover all possible behavioral scenarios.

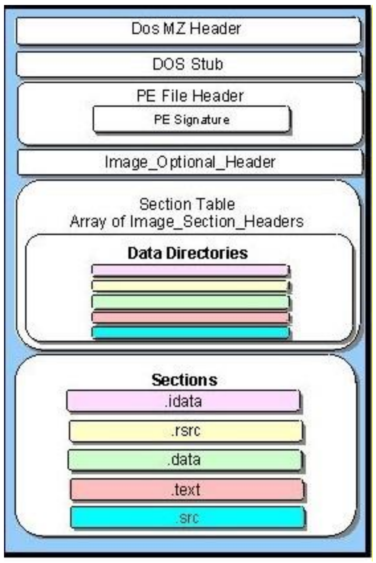
2.1.4 PE File Structure

PE file or portable executable file format is a type of format that is used in Windows (both x86 and x64). A PE executable basically contains two sections, which can be subdivided into several sections. One is Header and the other is called Section. The Figure 1 shows those sections.

According to Sikorski & Honig (2012) PE file headers can provide considerably more information than just imports. The PE file format contains a header followed by a series of sections. The header contains metadata about the file itself. Following the header there are the actual sections of the file, each of which contains useful information.

On this paper the most useful information on the PE file is the list of functions that it imports. According to Sikorski & Honig (2012) imports are functions used by one program that are stored in a different program, such as code libraries that contain functionality shared to many programs. Programmers in general re-use the imports into their programs and hence they do not need to re-implement certain functionalities in multiple programs. Knowing how and witch library code is imported is crucial for the understanding of the malware behaviour. The information that can be find in the PE file header depends on how the library code has been linked.

Figure 1: Sections of PE File



Source: Anderson et al., 2015.

2.1.5 Related Works

The work developed by Chumachenko (2017) explains the usage of machine learning to malware detection. In his work it is applied the feature of extraction on reports from cuckoo sandbox. The study performed can be useful as a base for further research in the field of malware analysis. The paper from Tahir (2017), defines the types of malware detections like static, dynamic and hybrid analysis and how each of them operates the various detection techniques.

Milosevic *et al*. (2017) presents the same problem and solution but considering a different operating system and market: the mobile android market. With the growth of mobile users and the ease of uploading a malicious code to Google Play it is necessary to develop an appropriate security method for this platform. Nath & Mehtre (2014) use static malware analysis using machine learning methods. The authors make a deep research on the static analysis techniques for malware detection to have a better understanding on how to stop zero-day attacks.

2.2 Research Methods

2.2.1 Source Gathering

To conduct the experiments, all the malware and malicious executables were gathered from different sources. The malware samples where downloaded from an open source repository known as Virusshare. The final folder contains about 285 malware samples for the Windows® platform among a total of 6 different malware families and dated from 2012 to 2019. A total of 549 malicious files were randomly selected among the existing PE files from “System32” and “Program and Files” folders. A more detailed list of files used in this work can be find at the end of the paper.

2.2.2 Dissecting Malicious And Benign Executables

As it was described previously, the data set consists of about 549 malicious files and 285 malwares. The next step is to extract the features from these files: "*In a malware dissection there is two ways of generating features, with dynamic and static analysis, the static analysis the extraction is done without the execution of the malicious code whereas in dynamic analysis features are derived while running the executable. (Sewak et al., 2019)".*

In this paper the focus is on the headers and API calls of each file. The study is developed by using a python library called PEfile to extract the API calls and to build the features of the dataset. At this level of the analysis it is noticed that some calls like: WriteFile, ReadFile, CopyFile and CloseHandle are more frequent in malicious executables. These calls can offer some clarification on the attacker intent and goals on the system under attack. According to Sewak et al. (2019), some families of malwares share the same group of opcodes, but a few opcodes are more dominant in malicious files when compared to malicious executables. This characteristic makes it easier to detect a pattern by using a machine learning algorithm.

2.2.3 Dissecting Malicious And Benign Executables

The DLL of a program provides a variety of information about its functionality. For instance, Table 2 lists common DLLs and what kind of information it can provide regarding an application. Theses DLLs were selected to extract the features. The extraction resulted in an array with about 3,500 features. This array size is too big for a dataset analysis and in general it is filled with worthless information. Therefore, it was applied some feature reduction methods to remove the less important features. In that case, if the feature has 2 or less occurrences in all the 834 files, it is considered as a low-value information and hence it can be removed without a substantial impact. After the reduction, the number of features reached a value of about 2,500, which corresponds to a smaller and more manageable dataset that contains a reasonable amount of useful information.

Table 2: Common DLLs in malwares

DLL	Description
Kernel32.dll	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
Advapi32.dll	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
User32.dll	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
Gdi32.dll	This DLL contains functions for displaying and manipulating graphics.
Ntdll.dll	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by Kernel32.dll. If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
WSock32.dll and Ws2_32.dll	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
Wininet.dll	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.

Source: Sikorski & Honig, 2012

To construct a dataset, it was chosen to use a pandas data frame, to keep the information of all imports occurrence of each file. In this case, each import is a new column in the data frame. The number of columns is determined by the unique imports used by the files, thus there are no identical columns. The last step in the python script is to write and export the data frame into a comma delimited CSV file.

The **Appendix I** illustrates a flow-chart of the dataset building process, from the file gathering to the use of machine learning algorithms. The first phase is gathering the files, both benign and malicious, next the amount of each type is balanced so the dataset can have a bigger heterogeneity. The second phase is a feature extraction, in which the API call is extracted from the selected DLLs as listed in table 2. After the extraction, the reduction is applied to finalize the dataset for the machine learning algorithms.

3. Results

The study was performed in an updated Windows 10 version, an Intel I5 9600K processor with 16GB of RAM DDR4 and a 480GB Kingston SSD. The dataset extraction is coded with python language (version 3.7), and the dependencies used are PEfile, pandas, numpy and xlsxwriter to extract and register the headers of all the files to create a used imports dictionary. The result was plotted using a python library called matplotlib. The final size of the dataset used for these tests is 834 cases, divided in 634 cases for training and 200 cases for validation tests.

3.1 MLP Calibration

Table 3 shows the number of layers and neurons applied on the Multilayer Perceptron. The different sizes selected starting from 4 were incremented by 4 as well with the end at 76, (4,8,12,16….,76). In total 210 different combinations of layers and sizes were made. The training and test dataset are randomly generated, 5 independent runs and in each run the MLP is applied 5 more times, and the average result of those runs is used. This resulted in a total of 5225 runs and 4.5 hours of execution time.

Table 3: Sizes and Hidden Layers Example

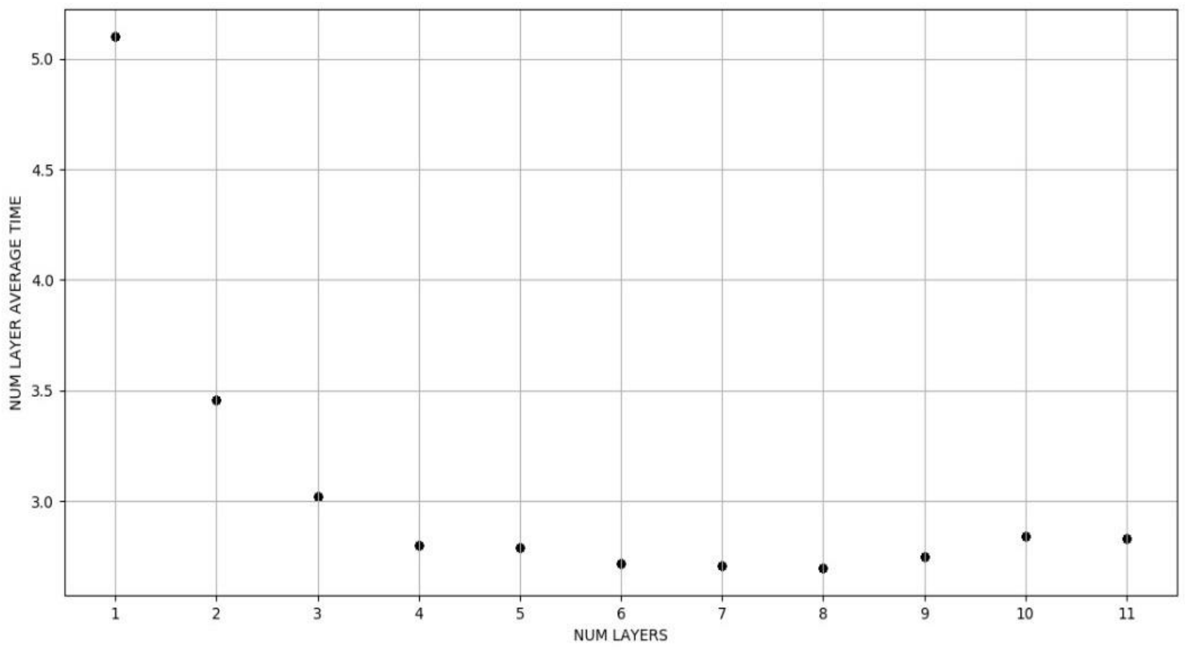
Number of Layers	Total neurons in each layer	Example
1	4	4
2	4	4,4
3	4	4,4,4
4	4	4,4,4,4
5	4	4,4,4,4,4
6	4	4,4,4,4,4,4
7	4	4,4,4,4,4,4,4
8	4	4,4,4,4,4,4,4,4
9	4	4,4,4,4,4,4,4,4,4
10	4	4,4,4,4,4,4,4,4,4,4
11	4	4,4,4,4,4,4,4,4,4,4,4

Source: Authors, 2020.

Figure 2 shows the average execution time by the number of hidden layers with all sizes. The worst processing time occurs with just one layer and it was about 5.3 seconds. As the number of hidden layers increases the average time drops to about 2.5 seconds and stabilizes. Based on that it is possible to conclude that the best number of hidden layers in this dataset is 3 or 4.

By analysing the size of the hidden layers, figure 4 is a plot of the average processing time by hidden layer size. The best size for this situation is about 24 neurons in each layer. Figure 5 shows an average score of each layer size, and analysing the plot is possible to note that after 12 neurons, the size does not have a significant impact on the average score.

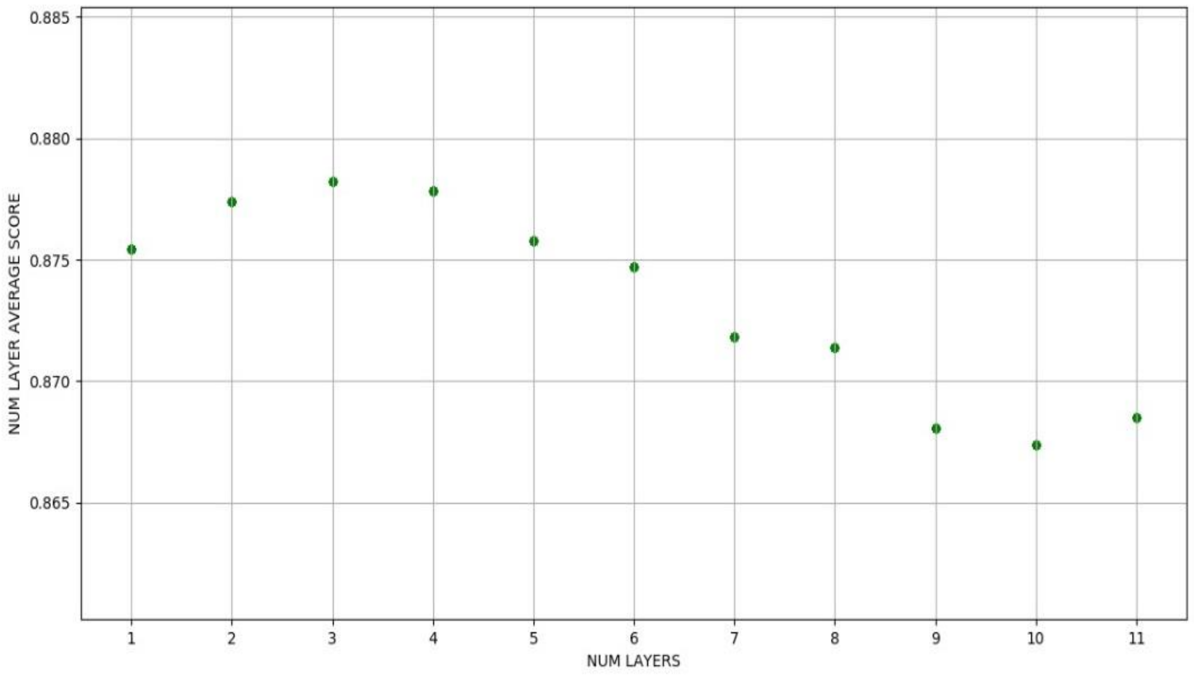
Figure 2: Execution Time/Layer Evaluation



Source: Authors, 2020.

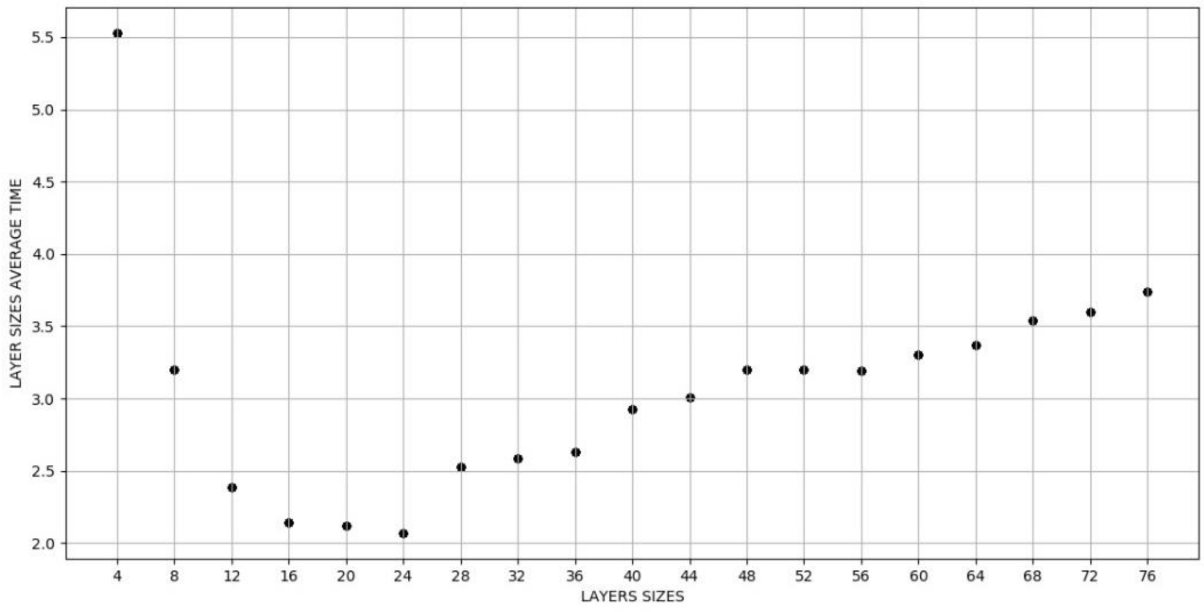
Figure 3 compares average score with number of layers. It is possible to notice that the MLP performed the best result with 3 hidden layers, narrowing the previous options of 3 or 4 to only 3.

Figure 3: Score/Layer Evaluation



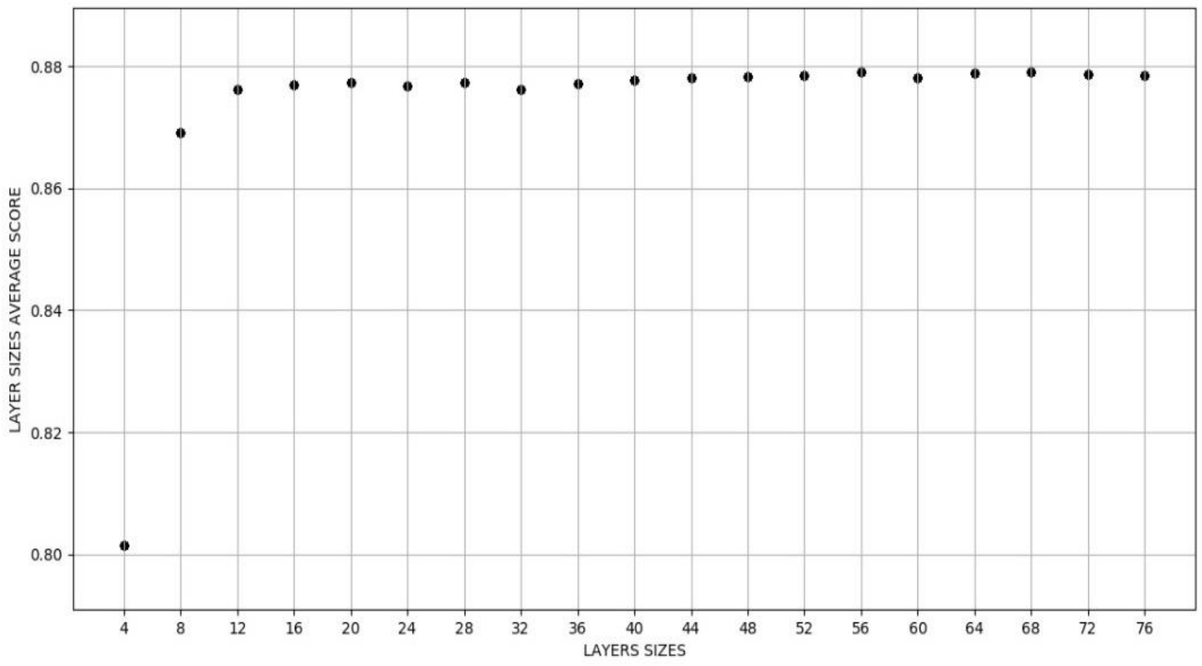
Source: Authors, 2020.

Figure 4: Execution Time versus Size Evaluation (milliseconds)



Source: Authors, 2020.

Figure 5: Score versus Size Evaluation



Source: Authors, 2020.

3.2 Algorithms Experiments

With the MLP properly configurated, the dataset was applied in multiple machine learning algorithms. Table 4 shows the maximum, average and minimum score of each algorithm for 100 tests each. From the results in table 4, its concluded that the MLP had the best performance at 95% of accuracy with an average of 88.33% when compared to the other algorithms. Naive bayes algorithm had the worst performance at 77% with an average of 70%.

For a better visualization of the analysis, Figure 13 shows a box plot comparison of the four algorithms. In a box plot, the middle line inside the box represents the median score while the box size corresponds to its variation. Therefore, a smaller box equals a smaller range of scores. It can be noted that there is difference between the algorithms performance for this model of malware detection.

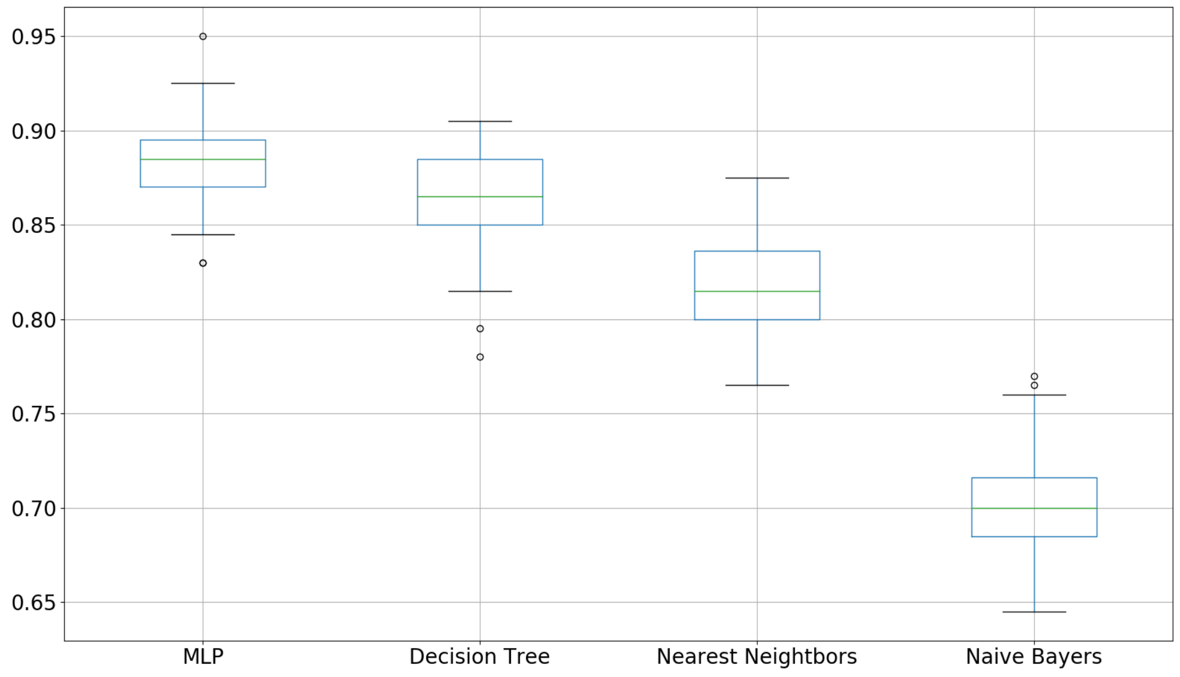
Table 4: Scores of 100 runs for each Algorithm

Algorithm	Score (%)		
	Min	Average	Max
MLP	83%	88.33%	95%
Decision Tree	78%	86.36%	90.5%
Naive Bayes	64.5%	70%	77%
K-Nearest Neighbour	76.5%	81.8%	87.5%

Source: Authors, 2020

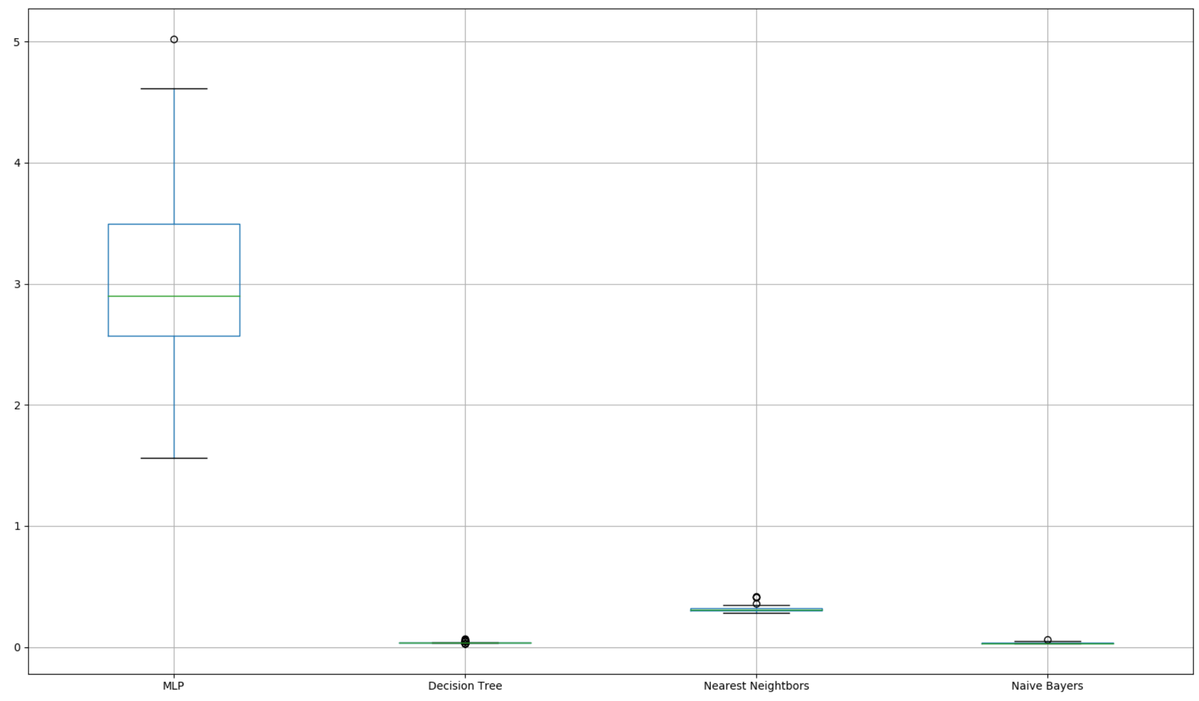
Figure 6 is another boxplot, showing the execution time of each algorithm. From that figure it is concluded that the MLP execution time is significantly bigger when compared to decision tree, naive bayes and K-nearest neighbour. Excluding the MLP from the analysis, it can be seen that the time does not have a significant variation among the 100 execution, thus the smaller box. The Figure 8 is a scatter comparison between time and score of all algorithms in 100 executions each. According to figures 6-8 presented, it is possible to infer that MLP, despite needing calibrations and a considerably better processing time, presented an accuracy, both in average and in maximum values, superior to the other algorithms

Figure 6: Algorithms Accuracy (100 Executions)



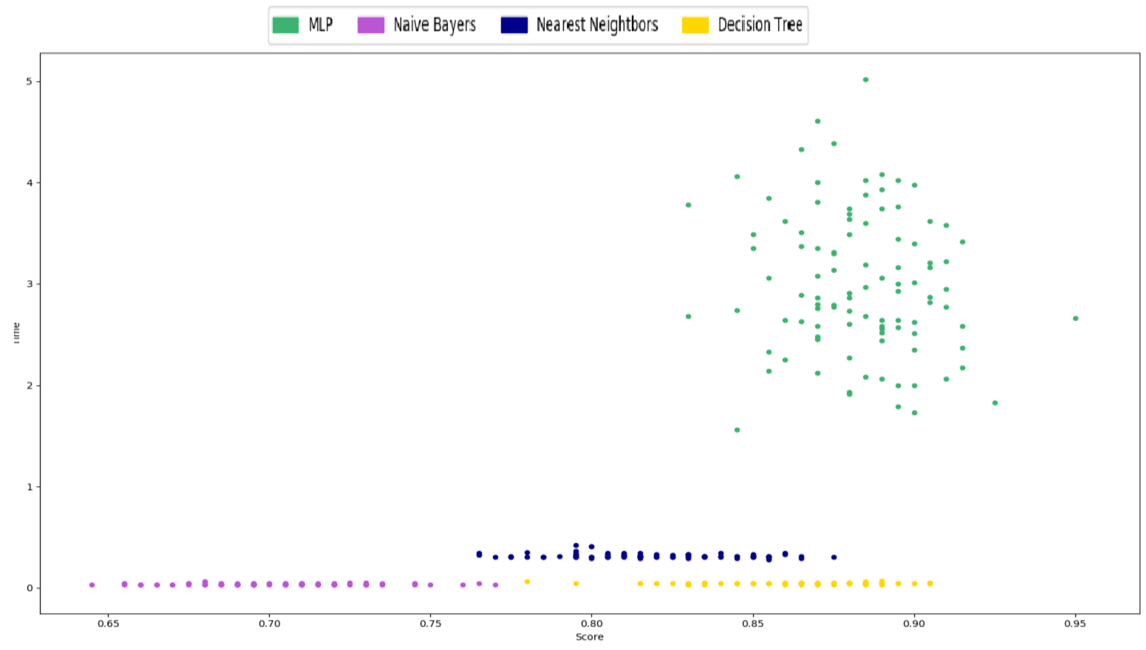
Source: Authors, 2020

Figure 7: Algorithms Time (100 Executions – In seconds)



Source: Authors, 2020

Figure 8: Algorithms Time versus Accuracy (100 Executions – In seconds)



Source: Authors, 2020

4. Conclusion

The most common method of detection used today is signature-based detection, which is an inefficient method when applied to zero-day attacks and less known malwares. The objective of this work was to propose a new malware detection method based in pattern recognition of API calls using machine learning algorithms.

This study concluded that it is plausible the usage of machine learning algorithms to differentiate a malware when the file is not obfuscated from a malicious file with an 88.33% accuracy, using a multilayer perceptron, based only in the API calls of each file.

The next steps of future work on this field is to collect information and build a dataset of a dynamic malware analysis, trying to find a pattern on the file behaviour, for example, the quantity of internet requests, files moved, created or deleted, which folder and processes the malware is trying to access.

References

[1] Alexandrou, A. (2019). 10 Cybercrime. *International and Transnational Crime and Justice*. Cambridge University Press,578 p.

[2] Rothrock, R. (2018). Digital Resilience: Is Your Company Ready for the Next Cyber Threat?. *Amacom*.

[3] Savage, K, Coogan, P and Lau, H (2015) The evolution of ransomware. *Symantec, Mountain View*

[4] Lallie, H S, Shepherd, L A, Nurse, J R, Erola, A, Epiphaniou, G, Maple, C and Bellekens, X (2020). Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic. *arXiv preprint arXiv:2006.11929*.

[5] Santos P H J, Costa Junior C A D F, Rodrigues, D S S, Carvalho L M, Souza F H B (2020) Information Security and Machine Learning: Encryption Allocation Based on Recognizing of Text Patterns. *In Proceedings of International Joint Conference on Industrial Engineering and Operations Management- ABEPRO-ADINGOR-IISE-AIMASEM (IJCIEOM 2020)*.

[6] Chaurasia, R. (2018). Ransomware: The Cyber Extortionist. In Handbook of Research on Information and Cyber Security in the Fourth Industrial Revolution (pp. 64-111). *IGI Global*.

[7] Idika, N. and Mathur, A. P. (2007). A survey of malware detection techniques. *Purdue University*, 48, 2007-2.

[8] Tahir, R (2018) A study on malware and malware detection techniques. *International Journal of Education and Management Engineering,* v8, n.2, p.20.

[9] Prasad, B. J., Annangi, H., and Pendyala, K. S. (2016). Basic static malware analysis using open source tools.

[10] Chumachenko, K. (2017). Machine learning methods for malware detection and classification. Bachelor’s Thesis. Information Technology.

[11] Sikorski, M, and Honig, A (2012). Practical malware analysis: the hands-on guide to dissecting malicious software. *no starch press*.

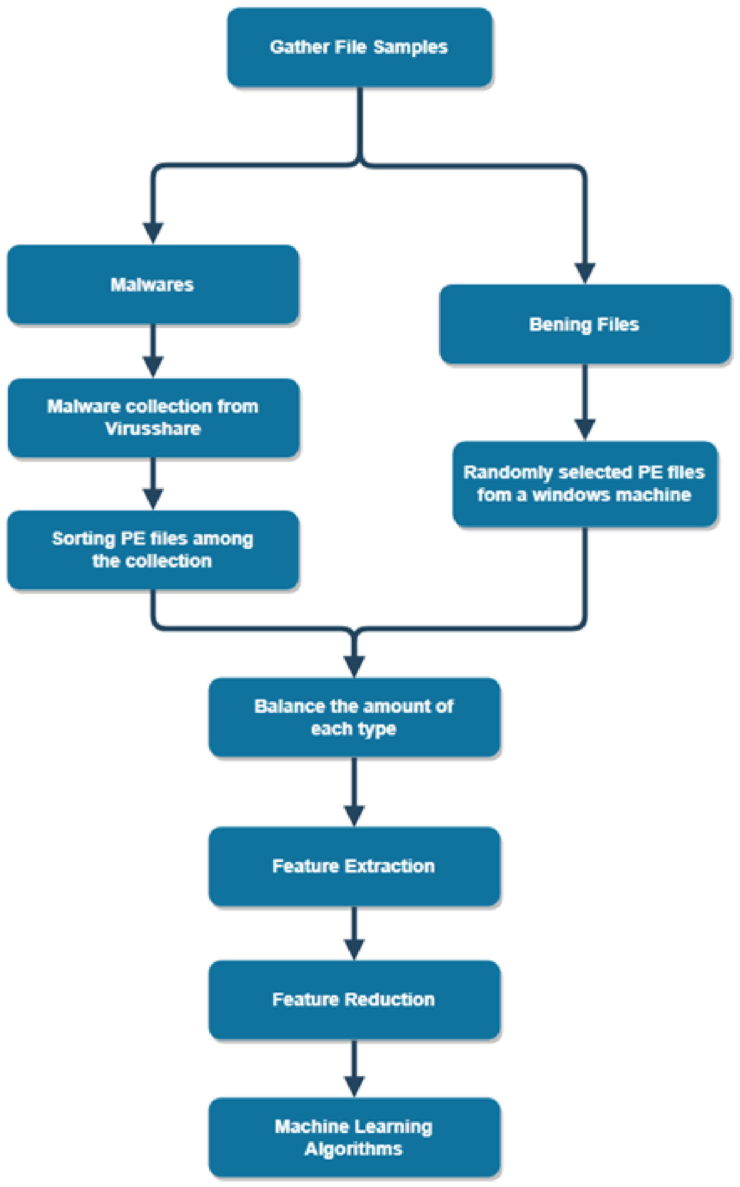
[12] Anderson, H. S., Kharkar, A., Filar, B. and Roth, P (2017). Evading Machine Learning Malware Detection, *blackhat.com*.

[13] Milosevic, N., Dehghantanha, A. and Choo, K. K. R. (2017). Machine learning aided Android malware classification. *Computers & Electrical Engineering*, v. 61, 266-274.

[14] Nath, H. V. and Mehtre, B. M. (2014). Static malware analysis using machine learning methods. *In International Conference on Security in Computer Networks and Distributed Systems*, pp. 440-450, Springer, Berlin, Heidelberg.

[15] Sewak, M., Sahay, S. K. and Rathore, H. (2018). An investigation of a deep learning based malware detection system. *In Proceedings of the 13th International Conference on Availability, Reliability and Security,* pp. 1-5.

Appendix I - Methodology Flow-chart


Gather File Samples
Malwares
Bening Files
Malware collection from Virusshare
Randomly selected PE files fom a windows machine
Sorting PE files among the collection
Balance the amount of each type
Feature Extraction
Feature Reduction
Machine Learning Algorithms

Source: Authors, 2020