An Automated Tool for Malware Analysis and Classification

Siti Rahayu Selamat[1] and Ng Thiam Tet[2]
[1,2] Faculty of Information and Communication Technology,
Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, 76100 Durian Tunggal, Melaka
Email: [1]sitirahayu@utem.edu.my, [2]ngthiamtet@gmail.com

Abstract— Malware attacks are still increasing up till today. This situation will cause many unwanted disturbances in the network or system that is being attacked by malware. Furthermore, malware is hard to identify due to the huge amount of samples and its unknown activities. Therefore, an automated tool is needed to analyze the malware samples and identify their activities. Due to that, a malware analysis will be integrated within the tool to be able to address the activities of the malware. The analysis method used is the hybrid analysis technique which, it combined both static and dynamic analysis techniques. Static analysis technique is a technique where malware is dissected, and reverse engineered to gain more information without executing the malware. Contrary to static analysis, dynamic analysis will execute the malware in a secure environment to further observe the behavior and activities carried out by the malware. In addition, a classification method via an application programming interface (API) calls made by the malware is implemented within the tool that capable to differentiate between a normal program and malware. The development of the automated tool is used Java and Python language. The result will be determined by the ability of logging and identifying the malware activities via an API call, and the ability to classify and differentiate between a malware and a normal program. In conclusion, the integration of malware analysis techniques and classification techniques will help provide more information to identify and differentiate a malware from normal programs.

Index Terms— Malware, Malware Analysis, Malware Classification, Tool.

I. Introduction

Malicious software also known as malware is utilized or made by attackers to disturb PC activity, gain access to sensitive information or access private computer systems unauthorized [1]. There are multiple types of malware throughout the history of the Internet that has been known to mankind, some of which includes the viruses, worms, Trojans, rootkits, ransomwares, key loggers and graywares.

According to Cohen [2], the definition of a virus is a program that can infect other programs by hijacking to include a potentially evolved duplicate of itself. Worm, as defined by Weaver, Paxson, Staniford, & Cunningham [3] is a program that abuses security exploits in widely used services to self-propagate through the network. Meanwhile, Trojan horse as defined by Windows [4] is a program that seems like a legit program, however it carries out some illegal action when it is run. According to Landi [5], rootkits are programs and codes which are able to be undetected either permanently or consistently on a PC. On the other hand, Brewer [6] defines ransomware is a type of malware using cryptography to encrypt files on the hard disk and prevents access to it while some also blackmails the victim by threatening to spread victim's personal information unless the ransom is paid. Ahmed, Maarof, Hassan, & Abshir [7] describes a key logger as a malware that records the client's keystroke on the keyboard. Fortinet [8] defined grayware as a term used for a variety of programs that are installed on a client's PC to track or potentially send certain data back to some other source.

Over the last decade, the number of malware has increased tremendously starting from 2007 all the way until today [9]. The increase in the amount of new malware from Q4 of year 2015 to Q3 of year 2017 and from Q4 of year 2016 to Q3 of year 2018 are represented in Figure 1 and Figure 2 respectively. Based on these figures, malware attacks are increased rapidly. Thus, there is a need to identify and analyze the malware activities or behavior in order to identify the malware and their attacks.

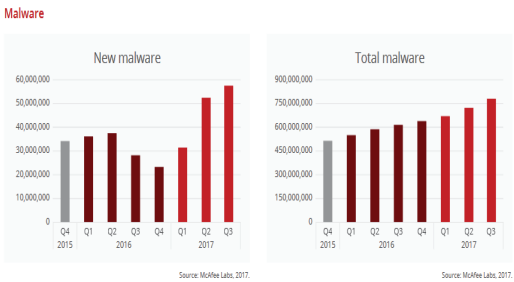

Fig. 1. Statistic Report for Number of New Malware through Q4, 2015 to Q3, 2017 [11]

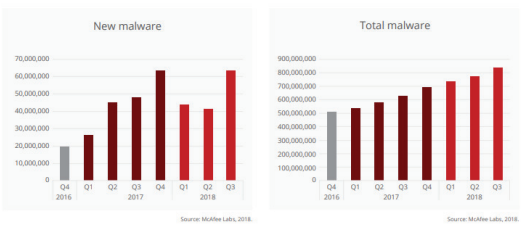

Fig. 2. Statistic Report for Number of New Malware through Q4, 2016 to Q3, 2018 [11]

II. Related Works

In this section, the malware analysis techniques and classification techniques are discussed.

A. Malware Analysis Techniques

Malware analysis is a technique which malware analysts use to identify the functions of a malware. There are three types of analysis technique which includes, static analysis, dynamic analysis and dynamic analysis. Static analysis is the technique of investigating an application without executing it [12]. This is made possible through program analyzers, debuggers and disassemblers. As for dynamic analysis, it is referred to as

analyzing the activities performed by a program while it is being executed in a safe environment [12]. Hybrid analysis is the combination of both static and dynamic analysis.

Static analysis can be divided into two approaches which are basic and advanced static analysis. Basic static analysis comprises of analyzing the executable file without looking at the real codes which includes antivirus scanning, hashing for fingerprinting, finding strings that may bring significant meaning, checking if the file is packed, performing YARA rule checking and checking the imported linked libraries and functions [13].

Advanced static analysis comprises of figuring out the malware's codes to find out what the program does by reversing the application using a disassembler. Be that as it may, advanced static analysis requires in-depth knowledge of disassembly, code constructs, Windows API, Registry, Networking API and kernels as well as performing reverse engineering using IDA or other disassemblers [14]. With more experience, advanced static analysis will be easier.

Dynamic analysis can be divided into two approaches which are basic and advanced dynamic analysis. Usually, dynamic analysis is done after the static analysis has been done on the application. Basic dynamic analysis is the act of running the malware and watching its action on the system keeping in mind that the end goal is to clean the system, deliver useful trace, or both. Nonetheless, before running a malware, it is required to set up a safe environment to analyses the running malware without danger of harming the host system [15]. Furthermore, the basic dynamic analysis includes using a sandbox, running the malware, monitoring the process with Process Monitor and Process Explorer, comparing registry snapshots for change in the registry and packet sniffing using Wireshark to determine traffic flow [16].

On the other hand, advanced dynamic analysis utilizes a debugger to look at the code of a running malware. A debugger is a type of application utilized to perform checks or to study the execution of another program. Advanced dynamic analysis methods make it easier to extract information regarding an executable as the data is gained during runtime. This is to ensure any obfuscation of code has been undone and is in plain assembly for easier readability. Advanced dynamic analysis requires knowledge of debugging, using a debugger and having the knowledge of kernels. Furthermore, there are different types of debugger which are source-level debugger and assembly debugger and two modes for debugging which are kernel-mode and user-mode debugging [17].

B. Malware Classification Techniques

Malware classification is a method to categorize malware based on different aspects such as how they get executed, how they spread, and/or what they do. There are many ways to classify malware such as using the machine learning engine, comparing the binary content and frequency of instruction.

Classification with machine learning utilizes an arrangement of pre-characterized samples to create a model which can be utilized for future classification. The classifier will learn with the use of training samples to create a classifier model [18]. Perdisci, Lanzi, & Lee [19] proposed an extension of this method, where malwares are to be unpacked before classification. The extension's method was to have a classifier to identify if the executable is packed or not, having a universal unpacked and a classifier to differentiate if the code is malicious or not. Rieck, Trinius, Willems, & Holz [20] utilized machine learning in their proposed system to classify malware base on scalable clustering and classification, incremental analysis of malware behaviour and extensive evaluation with real malware.

Binary content comparison for malware classification uses 4 methods as mentioned by Tabish, Shafiq, & Farooq [21]. The methods are generating block of codes, extracting features from the block, data mining and comparison and correlation. Firstly, the block generator module separates the byte-level code of a given file into blocks. Then the feature extraction module will use a total of 13 different features which are Simpson's Index, Canberra Distance, Minkowski Distance of Order, Manhattan Distance, Chebyshev Distance, Bray Curtis Distance, Angular Separation, Correlation Coefficient, Entropy, Kullback - Leibler Divergence, Jensen-Shannon Divergence, Itakura-Saito Divergence and Total Variation. Next, after the features are extracted, the data mining module will utilise the decision tree and Boosting (Ada-BoostM1) which is implemented in WEKA to classify the input file. Finally, the comparison and correlation module will find the similarity between blocks and classify whether the input file is malicious or benign. Kim, Kang, & Im [22] stated that it is possible to classify malware by dividing the codes into smaller segment and analyse the divided code. There are four modules which are the disassembler, block divider, major block selector, and similarity calculator. The disassembler will change the input file into a set of instructions while the block divider will separate the instructions from the disassembler into many blocks using the *retn* instruction as the end of a block. Next, the major block selector will scan all blocks for function call instructions which use the call or jump instruction. Finally, the similarity calculator will compare dataset from a database with the major block from the major block selector section.

Instruction frequency-based malware classification utilizes information from both malware and legit files [22]. The classification is made by identifying the frequency of instructions from a set of instructions that are malicious. This allows the classifier to differentiate between a malware and a legit file. Bilar [23] proposed using opcodes to classify malware where a dataset is created by using PEiD to collect PE header information of the malware and legit file. Next, a list of opcodes was taken from the samples and added along with the list in Wikipedia to a total of 398 IA-32 opcodes. From the extraction of opcodes from the samples, the result of extraction was compared to the list of opcodes and broken down into the most frequent opcodes for malware. Finally, from the list of most frequent opcodes, statistics were made to obtain a result that shows the difference between malware and clean files. Han, Kang, & Im [24] added to the approach of using instruction frequency to classify malwares. The approach stated that the instructions of a specific malware are made from opcodes like *'push,' 'mov,' 'sub'* and more. The approach proposed a design to utilize a simple binary comparison method using instruction frequencies. First, the method will compare a malware with a normal program and map each instruction that is like each other. Then, the instructions from two malware of similar family are compared and mapped to show the instruction frequency difference. From the result of comparison, it will be easy and fast to classify a new input, whereby, if the new input has low instruction frequency difference, it will be categorized under the same family as the malware it is used to compare with.

III. Methodology And Implementation

The methodology of the overall analysis process illustrated in Figure 3 is started by performing analysis on the file followed by classification process. The analysis approach is divided into three parts which are static analysis, dynamic analysis and hybrid analysis as depicted in Figure 3 while the classification is a standalone process.

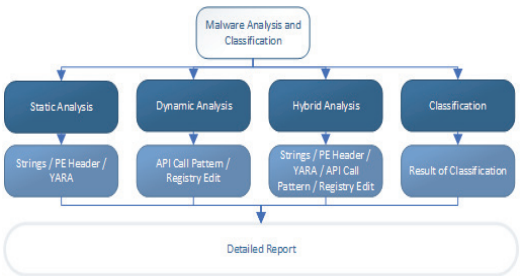

Fig. 3. Proposed Malware Analysis and Classification Tool Approach

For the implementation, an application development environment setup is created properly as it plays an important role in the implementation phase. The lack in development environment setup may lead to unexpected errors or events which could affect the implementation result. The architecture of the setup is illustrated in Figure 4.

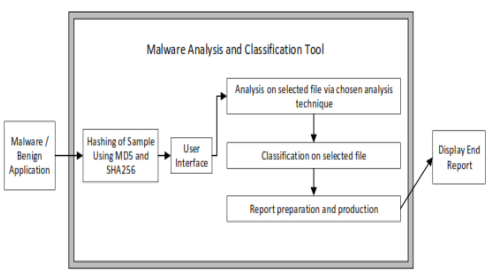

Fig. 4. Malware Analysis and Classification System Architecture

Based on Figure 4, there are 4 modules involved in the implementation of the application which are hashing of sample using Message Digest and SHA, performing analysis, classification and report preparation. Firstly, to be able to fully utilize the tool, the user will need to provide a file to be analyzed by the tool. The objective of this process is to ensure that there is something provided for the tool to carry out its functions. Once the user has selected a file, the user is then allowed to interact with the user interface of the tool where four other modules which are the static analysis, dynamic analysis, hybrid analysis and classification modules.

A. Implementation of Static Analysis

For static analysis, this research analyses the file via three methods which is printable string extraction, portable executable (PE) header information extraction and YARA rules checking.

a) YARA Rules Check: The first step of this process is to load PEiD rules, packer rules and anti-virtual machine/anti debugging rules. Upon completion of loading the rules, the file will be tested against these rules to find any matching array of bytes, string or reference. The PEiD rules are used to determine if the file has any PEiD recognizable packers. In addition, the packer rules are used to further determine if the file has been packed and what type of packer has been used. As for the anti-virtual machine/anti debugging rules, it is used to determine if the file has any anti-virtual machine/anti debugging properties that may disrupt further analysis. All matching rules will be displayed.

b) String Extraction: The first step of this process is to load the selected file into memory. Once the file is loaded, a regular expression pattern is used to search for all printable UTF-16 strings from within the region of memory of the file. Next, all strings that exist will be appended and displayed.

c) PE Header Information Extraction: The first step of this process is to load the selected file into memory. Once the file is loaded, extraction of the section information, header information and import information are executed. After all that, all information extracted will be displayed in a report. The information section will display information such as the section name, virtual address of the section, virtual size of the section, and the size of raw data. The header information will dis-play the image base, entry point address, and the number of sections as well as the section data. For the import information, the DLL, the imported function name and the imported function address will be displayed.

B. Implementation of Dynamic Analysis

For dynamic analysis, this research analyses the file by logging and visualizing the application programming interface (API) calls and checking on specific registry entries.

a) API Call Logging and Visualization: The first step of this process is to identify the type of file that has been loaded. If the file is a dynamic link library (DLL) file, the process will launch the target application which will host the DLL file. In this research, the target application is putty.exe. Once the target application is executed, the application will place a debugger hook into the target application. Next, the application will copy the DLL file to SysWOW64 directory and proceed with DLL injection into the target application. After injecting, API calls related to files and registry will be tracked and logged. However, if the file is an executable (EXE) file, the process will launch the EXE file and immediately place a debugger hook onto the newly launched EXE file. API calls related to files and registry will be tracked and logged. Finally, upon completion of API call logging, the log will be used to generate a graph to visualize the API call made by the DLL/EXE file.

b) Registry Entry Check: The first step of this process is to read the selected registry entry. After reading the selected registry entry, the application will make a backup of the said registry entry for comparison purposes after executing the malware. Once the malware has finished running, the selected registry will be read once more, and the initial registry will be loaded and compared to the current registry to find the differences that the malware potentially has changed. The registry will be displayed.

C. Implementation of Hybrid Analysis

For hybrid analysis, this research combines the methods from both static and dynamic analysis to analyze the file.

a) YARA Rules Check: The first step of this process is to load PEiD rules, packer rules and anti-virtual machine/anti debugging rules. Upon completion of loading the rules, the file will be tested against these rules to find any matching array of bytes, string or reference. The PEiD rules are used to determine if the file has any PEiD recognizable packers. In addition, the packer rules are used to further determine if the file has been packed and what type of packer has been used. As for the anti-virtual machine/anti debugging rules, it is used to determine if the file has any anti-virtual machine/anti debugging properties that may disrupt further analysis. All matching rules will be displayed.

b) String Extraction: The first step of this process is to load the selected file into memory. Once the file is loaded, a regular expression pattern is used to search for all printable UTF-16 strings from within the region of memory of the file. Next, all strings that exist will be appended and displayed.

c) PE Header Information Extraction: The first step of this process is to load the selected file into memory. Once the file is loaded, extraction of the section information, header information and import information are executed. After all that, all information extracted will be displayed in a report. The information section will display information such as the section name, virtual address of the section, virtual size of the section, and the size of raw data. The header information will dis-play the image base, entry point address, and the number of sections as well as the section data. For the import information, the DLL, the imported function name and the imported function address will be displayed.

d) API Call Logging and Visualization: The first step of this process is to identify the type of file that has been loaded. If the file is a dynamic link library (DLL) file, the process will launch the target application which will host the DLL file. In this research, the target application is putty.exe. Once the target application is executed, the application will place a debugger hook into the target application. Next, the application will copy the DLL file to SysWOW64 directory and proceed with DLL injection into the target application. After injecting, API calls related to files and registry will be tracked and logged. However, if the file is an executable (EXE) file, the process will launch the EXE file and immediately place a debugger hook onto the newly launched EXE file. API calls related to files and registry will be tracked and logged. Finally, upon completion of API call logging, the log will be used to generate a graph to visualize the API call made by the DLL/EXE file.

e) Registry Entry Check: The first step of this process is to read the selected registry entry. After reading the selected registry entry, the application will make a backup of the said registry entry for comparison purposes after executing the malware. Once the malware has finished running, the selected registry will be read once more, and the initial registry will be loaded and compared to the current registry to find the differences that the malware potentially has changed. The registry will be displayed.

D. Implementation of Classification

For classification, this research performs classification by instruction frequency.

a) Instruction Extraction: The first step of this process is to create an *objfile* of the malware file. This can be done by using the *objdump* utility and the output is the file information, address, opcode and assembly instruction. Once the *objfile* is created, the assembly instruction is extracted with the utilization of *sed* and *cut* utilities. The extracted instructions are kept for instruction calculation and classification.

b) Instruction Calculation and Classification: The first step of this process is to load the previously saved file which contains the extracted instructions. Once this has been done, the application will iterate through the file and count the frequency of each individual instruction available as well as the total instruction. With the individual instruction frequency ready at hand, the application will divide each of the 8 instructions (*mov, add, push, pop, call, jmp, xor, cmp*) over the total instruction to gain the percentage of the instructions. Finally, this percentage will be used to perform classification.

IV. Result

The analysis of the four main processes which are static analysis process, dynamic analysis process, hybrid analysis process and classification process is performed in an emulated environment that runs a Microsoft Windows 7 (64-bit). The Conficker worm provided by MyCERT from CyberSecurity Malaysia is used in this research to generate the results.

A. Static Analysis Module

In static analysis module, there are 3 routines which are processed.

a) YARA Rule Check: PEiD, packer and anti-virtual machine/anti-debugging rules are loaded, and the malware is checked against the rules. The details of the matches will be displayed as depicted in Figure 4.

b) String Extraction: The malware is loaded into the memory and a regular expression pattern is used to display all printable strings found within the malware. The list of printable strings will be displayed as depicted in Figure 5.

c) PE Header Information Extraction:

The malware is loaded into the memory and the extraction of the section information, header information and import information are executed. Result from the information extraction is displayed as depicted in Figure 5.

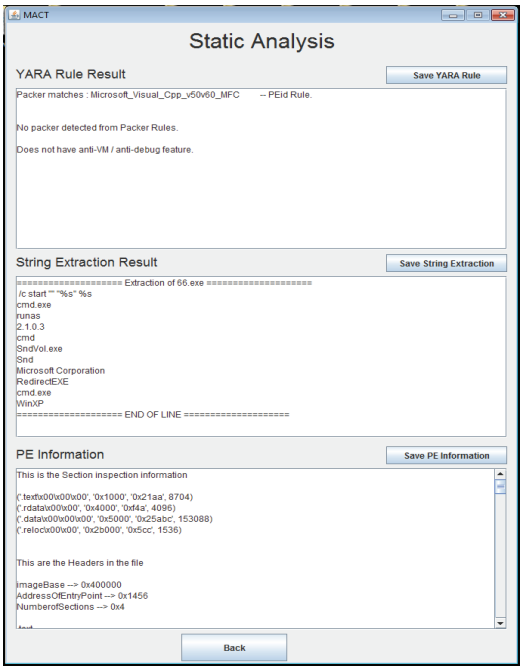

Fig. 5. Static Analysis Result

B. Dynamic Analysis Module

In dynamic analysis module, there are 2 routines which are processed.

a) API Call Logging and Visualization: The malware is executed and the graph for the API call is generated as shown in Figure 6.

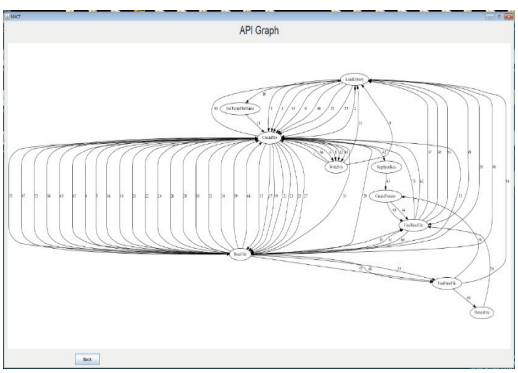

Fig. 6. API Call Visualization Result

b) Registry Entry Check: The registry changes that are made by the malware that has been executed towards specific registry is displayed in Figure 7.

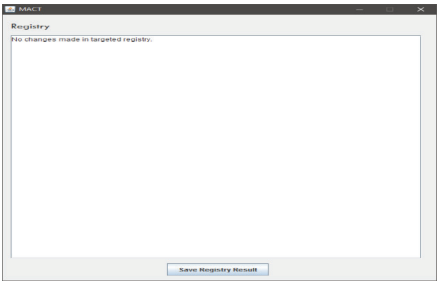

Fig. 7. Registry Entry Check Result

C. Hybrid Analysis Module

In hybrid analysis module, there are 5 routines which are processed.

a) YARA Rule Check: PEiD, packer and anti-virtual machine/anti-debugging rules are loaded, and the malware is checked against the rules. The details of the matches will be displayed as depicted in Figure 7.

b) String Extraction: The malware is loaded into the memory and a regular expression pattern is used to display all printable strings found within the malware. The list of printable strings will be displayed as depicted in Figure 7.

c) PE Header Information Extraction: The malware is loaded into the memory and the extraction of the section information, header information and import information are executed. Result from the information extraction is displayed as depicted in Figure 8.

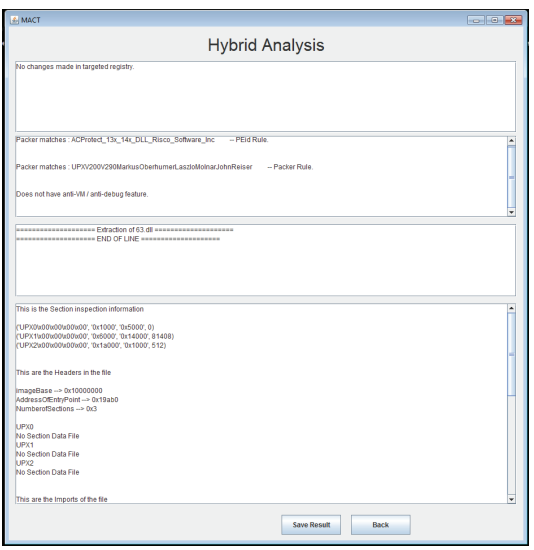

Fig. 8. Dynamic Analysis Result

d) API Call Logging and Visualization: The malware is executed and the graph for the API call is generated as shown in Figure 9.

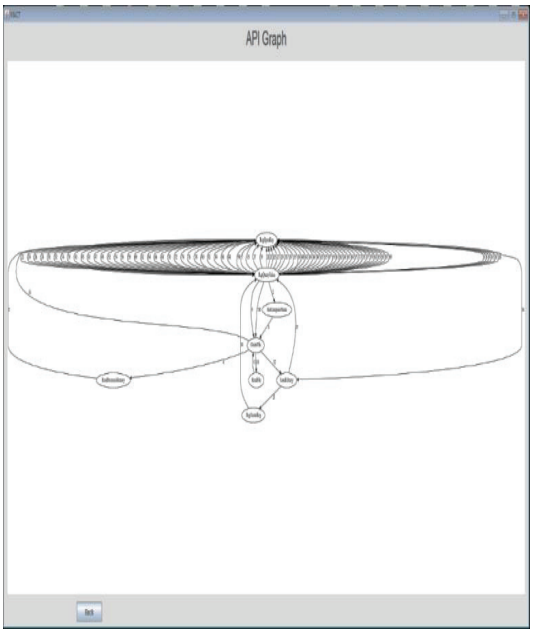

Fig. 9. API Call Visualization Result

e) Registry Entry Check: The registry changes that are made by the malware that has been executed towards specific registry is displayed in Figure 7.

D. Classification Module

In classification module, there are 2 routines which are processed.

a) Instruction Extraction: The *objfile* of the malware file is created by using the *objdump* utility. The assembly instruction is extracted from the *objfile* with the utilization of *sed* and *cut* utilities. This list of assembly instruction will be used for the calculation and classification process.

b) Instruction Calculation and Classification: The file which contains the extracted instructions is loaded and will be iterated to count the frequency of each individual instruction available as well as the total amount of instructions. With the individual instruction frequency ready at hand, 8 instructions (*mov, add, push, pop, call, jmp, xor, cmp*) will be divided over the total instruction to gain the percentage of the instructions. Finally, this percentage will be used to perform classification. The result of the classification is displayed as depicted in Figure 10.

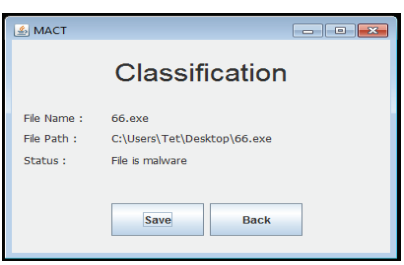

Fig. 10. Classification Result

V. Conclusion

In this paper, the researchers have utilized static and dynamic malware analysis in order to identify the behavior of the malware. The research combines both static and dynamic malware analysis techniques to gain more de-tails understanding on how the malware behave and the type of API called in the Windows environment. The output of the analysis is used to visualize the API call made by the malware. This research is a preliminary worked for automated malware analysis. The finding will contribute ideas in finding an effective method to automate malware analysis.

Acknowledgment

The authors would like to express the appreciation to INSFORNET Research Group, C-ACT and Universiti Teknikal Malaysia Melaka in encouraging the authors to perform this research and develop the tool.

References

[1] Milošević, N. (2013). History of malware. arXiv Preprint arXiv:1302.5392, pp. 1–11.
[2] Cohen, F. (1987). Computer viruses. Computers & Security, Vol. 6, No. 1, pp. 22–35.
[3] Weaver, N., Paxson, V., Staniford, S., & Cunningham, R. (2003). A taxonomy of computer worms. Proceedings of the 2003 ACM Workshop on Rapid Malcode - WORM'03, pp. 11.
[4] Windows, N. (2001). Backdoors and Trojan Horses. Information Security Technical Report, Vol. 6, No. 4, pp. 31–57.
[5] Landi, M. (2009). The methods of windows rootkits. Journal of Applied Security Research, Vol. 4, No. 3, pp. 389–426.
[6] Brewer, R. (2016). Ransomware attacks: detection, prevention and cure. Network Security, Vol. 9, pp. 5-9.
[7] Ahmed, Y. A., Maarof, M. A., Hassan, F. M., & Abshir, M. M. (2014). Survey of Key logger Technologies. International Journal of Computer Science and Telecommunications, Vol. 5, No. 2, pp. 31.
[8] Fortinet. (n.d.). PROTECTING NETWORKS AGAINST SPYWARE, ADWARE AND OTHER FORMS OF "GRAYWARE". California: Fortinet. Retrieved from http://www.boll.ch/fortinet/assets/Grayware.pdf
[9] McAfee . (2010). McAfee Threats Report: Third Quarter 2010. California: McAfee .
[10] McAfee. (2016). McAfee Labs Threats Report September 2016. California: McAfee.
[11] McAfee. (2018). McAfee Labs Threat Report December 2018. California: McAfee.
[12] Gandotra, E., Bansal, D., & Sofat, S. (2014). Malware Analysis and Classification: A Survey. Journal of Information Security, Vol. 5, No. 2, pp. 56–64.
[13] Sikorski, M., & Honig, A. (2012). Basic Static Analysis. In M. Sikorski, & A. Honig, PRACTICAL MALWARE ANALYSIS (The Hands-On Guide to Dissecting Malicious Software) (p. 2). San Francisco: No Starch Press.
[14] Kakisim, A. G., Nar, M., Carkaci, N., & Sogukpinar, I. (2018, November). Analysis and Evaluation of Dynamic Feature-Based Malware Detection Methods. In International Conference on Security for Information Technology and Communications, pp. 247-258. Springer, Cham.
[15] Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2012). A Survey on Automated Dynamic Malware-Analysis. ACM Computing Surveys, Vol. 44, No. 2, pp. 6-12.
[16] Bulazel, A., & Yener, B. (2017, November). A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium (p. 2). ACM.
[17] Sikorski, M., & Honig, A. (2012). Advanced Dynamic Analysis. In M. Sikorski, & A. Honig, PRACTICAL MALWARE ANALYSIS (The Hands-On Guide to Dissecting Malicious Software) (p. 3). San Francisco: No Starch Press.
[18] Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., & Nicholas, C. (2018). An investigation of byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques, Vol. 14, No. 1, pp. 1-2
[19] Perdisci, R., Lanzi, A., & Lee, W. (2008, December). McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. In 2008 Annual Computer Security Applications Conference (ACSAC) (pp. 301-310). IEEE.

[20] Rieck, K., Trinius, P., Willems, C., & Holz_aff2n3, T. (2011). Automatic Analysis of Malware Behavior Using Machine Learning. Journal of Computer Security, Vol. 19, No. 4, pp. 639–668.

[21] Tabish, S. M., Shafiq, M. Z., & Farooq, M. (2009). Malware detection using statistical analysis of byte-level file content. Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics - CSI-KDD '09, pp. 23–31.

[22] Kim, T. G., Kang, B., & Im, E. G. (2013). Malware classification method via binary content comparison. Information (Japan), Vol. 16, No. 8A, pp. 5773–5788.

[23] Bilar, D. (2007). Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics, Vol 1, No. 2, pp. 156.