

# Metaware - An Extensible Malware Detection and Removal Toolkit

Chan Lee Yee\*, Lee Ling Chuan\*\*, Mahamod Ismail\*, Kasmiran Jumari\*

\*Department of Electrical, Electronic and System Engineering, Universiti Kebangsaan Malaysia

\*\*Malaysia Computer Emergency Response Team, CyberSecurity Malaysia

chanleeyee@gmail.com, lc.lee@cybersecurity.my, mahamod@vlsi.eng.ukm.my, kbj@vlsi.eng.ukm.my

**Abstract**— Malicious code is a threat to computer security globally. The threat is evolving and leaving challenges for security specialists to improve the detection accuracy. Hence, it is imperative to optimize the traditional manual analysis method by automatic malicious code analysis system. Automatic protocol reverse-engineering is important for many security applications, including the verification of objects and detections of malware. In this paper, we propose a new approach to computer security via automating malware analysis. This project uses combination of auto-unpacked, heuristic, disassembler and emulator techniques to find and block malicious program before the malicious software executed locally. Auto-unpacked contains self-decryption algorithms, where the script codes help quickly decipher script bodies for further analysis. Heuristic analysis is designed to analyze disassemble code contain within a suspicious program. The disassemble code of the suspicious file is compared with a known virus signature database. If the disassemble code matches with the code of the database signature, the file is flagged. The emulator is design to scans code, imitates the way they are executing, and monitoring their actions, preventing any actual damage from being dealt to the computer system or user data. Verdicts on whether or not a program poses a threat are issued based on the results of behaviour analyses. The emulator makes it possible to find malicious code that are intentionally masked to prevent detection using encryption and obfuscated code. Overall, we present our motivation for designing the system and give an overview of the system architecture.

**Keywords**— Computer System Security, Virus Detection, Malware Reverse Engineering, Disassembler, Emulator

## I. INTRODUCTION

Malicious software, known as malware – poses a major threat to the security of computer system. Malware includes such malicious code as worms, viruses, Trojan horses, and spyware. The attacks are growing in popularity, and are increasing in complexity. Malware can be in various form, however, malware binary form is almost common in Internet. The technique of malware analysis either in sophistication of samples analyzed or the tools of the trade has been evolved. One of the ways to understand the threat associated with a malicious software specimen and provides better protecting the system in our network is to begin by examining its behaviour in a controlled environment. Beside behaviour analysis, static or code analysis, network traffic analysis

techniques [1] [2] are also useful for better understand the malicious that we are facing. Competency in each of these fields requires a dedicated analyst with specific skills encompassing reading and understanding of assembly code, proficiency in operating system internals, expertise in network protocols and other deep system knowledge. These extraordinary requirements for the malware analysis and the ensuing lack of properly trained personnel capable of doing reverse engineering, has made the malware analysis's time very precious resource.

In this paper, we propose a toolkit that allows for automatic analysis of malware behaviour using machine learning. This toolkit will execute the programming commands of a questionable program or script within a specialized virtual machine or emulator [9] and then internally simulate what would happen if the suspicious file were to be executed while keeping the suspicious code isolated from the real-world machine. It then analyzes the commands as they are performed, monitoring for common viral activities such as file overwrites, replication and attempts to hide the existence of the suspicious file. If one or more virus-like actions are detected, the suspicious file is flagged as a potential virus, and the user alerted. The toolkit will also perform decompile action of the suspicious program, and then analyze the disassemble code contained within. The disassemble code of the suspicious file is compared to the code of known viruses and virus-like activities. If the code partially matches with the code of known viruses or virus-like activities, the file is flagged and the user alerted.

## II. BACKGROUND AND RELATED WORK

Software automating malware analysis is a long and difficult process. Few people possess the related knowledge and skills required to reverse engineering software binary. Even fewer people can perform reverse engineering on binaries that are purposefully obfuscated, as apply by most modern malware.

Clam AntiVirus [3] also known as ClamAV is an open source (GPL), cross-platform antivirus software toolkit that capable to detect many types of malicious software. It provides a number of utilities including common line scanner, automatic database updater and a flexible and scalable multi-threaded daemon. The ClamAV virus database is updated very

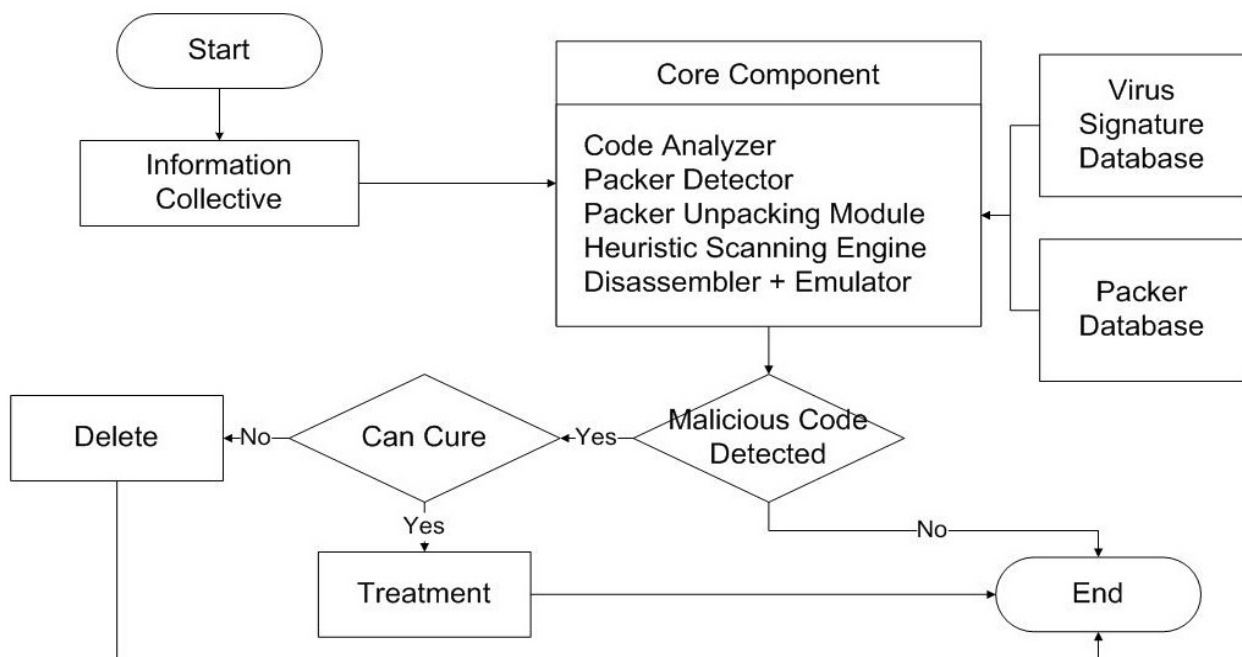


Figure 1. Flow Chart of Metaware Architecture

frequently and as of 20 October 2010 contained 866,266 virus signatures [3][4].

Norman Sandbox[5] is a malware-scanning tool that executes malware on virtualized hardware. This service allows user to upload a malicious executable to the Norman website; however, rather than simply checking the file against a database of known signatures, the service attempts to automatically determine what actions the executable would take on the infected system. After scanning the file, the service e-mails user a report that includes basic behavioral information, such as what files and registry entries the malicious program might create and some network-level activity the specimen may exhibit.

Bitblaze[6] is an innovative service in that it attempts to automatically unpack (dump) the Windows executable user upload via its web interface. The part of the service that extract packed code from malicious executable is called Renovo [7]. This module “monitors program execution and memory writes at run-time, determines if the codes under execution is newly generated, and then extract the hidden code of the executable.

### III.SYSTEM ARCHITECTURE

The goal of the Metaware malware analysis toolkits project is to design and develop in a modular fashion in order to make it flexibility, accuracy and extensibility. Accuracy plays an important role in motivating us to build precise, formal models of instructions that allow the tool to accurately model the program execution behaviour symbolically. As the complexity of binary malware analysis, we would like to develop core utilities which can be re-used and easily

extended to enable other more sophisticated analysis on binaries.

In this project, we propose using combination of heuristic, disassemble [14] and emulator technique [9] to detect and cure malicious scripts before malicious software executed locally. Figure 1 shows the overall flow chart of the system architecture. In order to prevent any obfuscation code poses obstacles to the scanning and detection module, an automating process for identifying and extracting the hidden-code bodies is proposed. We devise an algorithm for identifying if a program is applied any obfuscation mechanism. A module for automating the process of extracting the hidden-code bodies of the program is executed if any packer is detected [8].

The idea of heuristic scanning [10] [11] is to identify new malicious software. This is to ensure the user able to locate new virus infected files. The initialize phase of heuristic analysis can only perform searching of known virus signatures. The matching of all virus signatures for each traversed emulator instructions will take a very long time, especially if the database contains a large number of records. In this case, the signature is compared to full, and if matched with at least 80%, the binary will flag as infection.

Emulator is design to emulate the execution of a program (e.g. virus, worm etc.) in a special “environment”, also known as buffer emulation or “sandbox”. The emulator is design to scans code, imitates the way they are executing, and monitoring their actions, preventing any actual damage from being dealt to the computer system or user data. If the emulator is infected by a polymorphic virus file [13], the decrypted virus body in emulator will be ready for detection.

### **A. Information Collective**

With the help of GET\_COMMAND function, it able receives an instruction of scanning option. The information collective module calls GetCurrentDirectory, FindFirstFile, FindNextFile etc. Win32 API functions to collect information about the scanning target. This includes file size, file type or extension of the binary file and file permissions. Information collective is an important step for the system before core component module function is called.

### **B. Code Analyzer**

There are two different methods for developing emulator. First, it involves an ordinary trace the program, where the test code is loading into memory by using debug interrupt (int1). This method has been applied by most debuggers; the disadvantage is that this method easily breaks off the process of tracing. However, this method has successfully detected the family SMEG (Simulated Metamorphic Encryption Engine) [12] virus. The second method is direct emulation. A piece of code is read into the antivirus buffer; understand the instructions and their execution.

Typically, two main procedures are used during code analysis. First, collect the initial information about the area code before emulator function is called. Second is to identify the characteristics of parse code inside the emulation code.

Indeed, the idea of framework emulator is design to simulate the performance of small section of code. If the virus body is encrypted, the packer unpacking decrypts it and code analyzer scans the decrypted code for all the known virus signatures. Below shows the algorithms of parsing and analysis (code analyzer) technique, which the procedures that are used in the core component:

**Definition:** Design to reflect characteristics of the test (emulated) code and identify the characteristics of parsed code.

#### **Begin:**

- 1) Initialization of code analyzer
- 2) Execute of test code variables (resets the values for virtual registers, setting up the stack)
- 3) Pass malicious binary to emulator (the size of the test code, the memory location of the test code).
- 4) Start cycle code emulation:
  - a) Start heuristic code analyzer. If heuristical masks unable to detect, go to Step (C).
  - b) Search for virus signatures from current area code. If the process succeeds, the emulator is completed. The detection of virus entry will transfer to back to the toolkit.
  - c) Search for the value which the difference between the initial position of the original code to the memory and the current position.

- d) Analysis of disassembler instruction. If encounter any error (e.g. unknown instruction), exit the emulator.
- e) Analysis the instructions (the size of instructions, the usage of registers, the type of execution simulation) which received from the disassembler..
- f) Translation of a pointer to the next instruction.
- g) If the instructions of emulator go smoothly, and the desired portion of code has not been fully explored, move to the top of the cycle.

In the initialization of code analyser section, the required variable values such as the value of the time limitation of the emulator, the infected mark, setting the value of a virtual stage and virtual register etc. will be collected and subsequently transfer to the emulator process.

### **C. Packer Detector and Packer Unpacking**

Code packing or packer is one technique commonly used to hinder malware code analysis through reverse engineering. Our approach is, no matter what packing methods or how many hidden layers are applied, the original program code and data should eventually be present in memory to be executed. The instruction pointer should jump to the original entry point (OEP) of the restored program code which has been written in memory at run-time. The OEP is the first hidden instruction being executed when the program control flow is transferred from the restoration routine to the hidden code. Taking advantage of this inevitable nature of packed executables, we propose a module to extract the hidden original code from the packed executable. Packer detector function is built on top of scanning core components and it is develop to analysing an executable, and determines if the binary packed with any packer. If so, it will pass to packer remover to extracts the hidden code and obtains the hidden-code bodies.

### **D. Heuristic Scanning Engine**

The implementation of virus signatures detection described in this section is fairly conventional. Heuristic scanning engine uses a specific viral signature database to determine if a binary contains malicious program. The virus signature database is referring to common short signatures, which are presented in most viruses (also known as "suspicious" command). Our approach is to select two different partial code segments of the common short signatures and saved it into database. If the virus signatures are matching, the scanning file is flagged as virus. Figure 2 shows an example of heuristic scanning and detection of Virus.Win32.Kriz.4029 with pre-defined virus signature database. According to the virus signature database, the selected virus signatures are "060h, 0BEh" and "000h, 0F0h". The value after "\*" symbol represents total ignoring opcode in byte from the last detected opcode. As shown in figure, the scanning will start at code segment opcode "0E00 0050" until the detection of opcode "60BE". The heuristic scanning will jump 6 byte after "\*" symbol is detected. 6 byte after opcode "60BE" is "00F0", which is exactly match with the pre-define database.

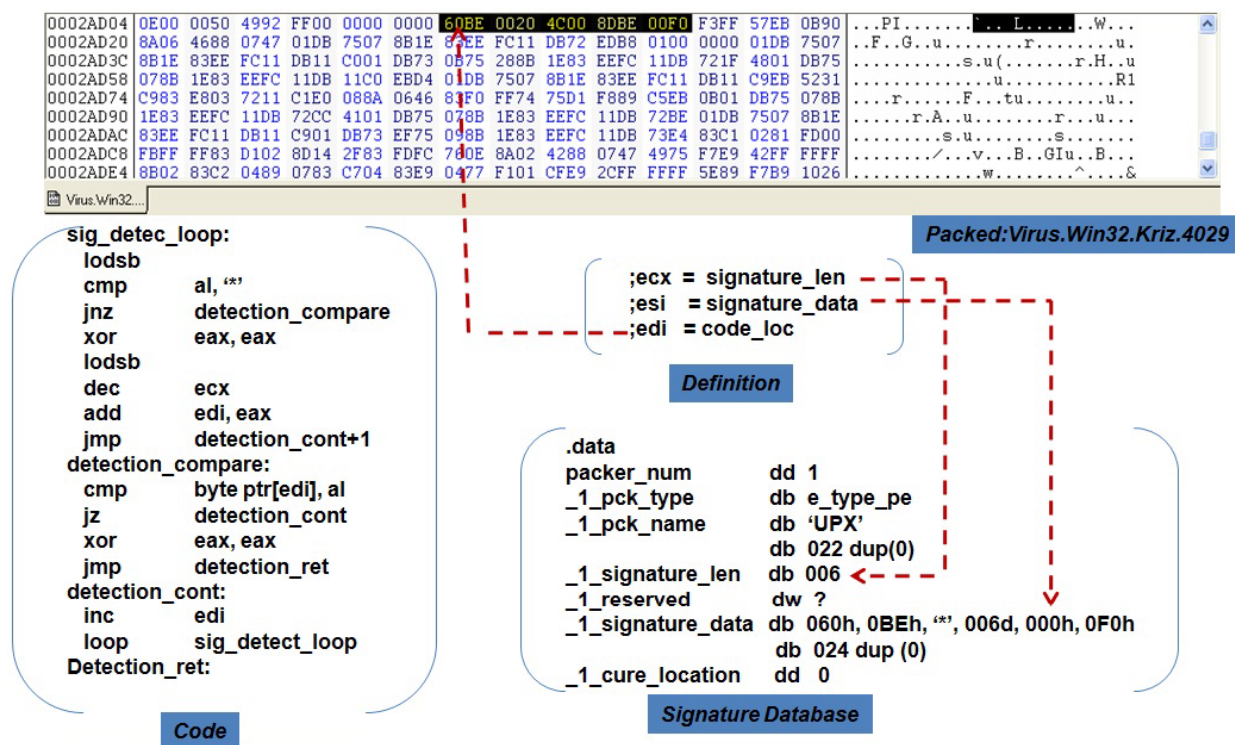


Figure 2. Heuristic Scanning of Virus.Win32.Kriz.4029

### E. Disassembler and Emulator

Emulator in this project can define as research programs for the content of code segments that perform virus-like or destructive acts and search for permanent code sections (signature) characteristics of certain viruses, which can be arranged in any order. The challenge of developing emulator is not only emulate the instruction, but also the challenges of dealing with operating system. The task of developing emulator is laborious as the emulator must constantly monitor every single action.

Imitation refers the process of implementation of various analogous actions manually. This mock execution required the modification shift of the execution program drawn from the current buffer to a design buffer named "dreg". The disassembler parses further instructions and fills special instructions parameters inside the buffer "dreg".

Before emulating the example test code, there are main parameters that should be used to ensure the emulator stable and "successful" work. The knowledge of the executable file's type such as COM, EXE, PE etc is one of the necessary and important options for universal emulator. This allows the emulator emulates the executable files with a particular way depending on the type of executable file. Original location of instructions in memory is another necessary option for emulator. In this case, the original location in memory refers to the entry point of the test program. Besides that, the emulator may need to know the location of the instruction

buffer, or the scanning buffer. The depth of analysis and the maximum time for the emulator program are also the required parameter for the emulator.

In order to emulate a test program but not to corrupt the data stored inside the stack, a special buffer, which is called a virtual stack, is given. All the manipulations produced together with stack in the simulation execution program will occur in this buffer. Prior process the emulation program, the values of virtual registers EAX and ESP must set. The EAX register must set to the value of the entry point of the program, and act as a parameter passed to the framework emulator. While installing the ESP register, buffer for the stack must be allocated. Virtual register ESP must contain the offset, indicating the end of the buffer. Before start the process of emulation program, the buffer under the stack must be defined and allocate variables under virtual registers. The virtual ESP register must contain the end of the buffer offset allocated under the stack.

The term anti-emulation technique refers to steps that a code author may take to deceive emulator. This tactic will fool the emulator, as it make the emulator enters the eternal cycle. As a result, the scanning analysis process just hangs and cause the emulator fail to decode the virus body correctly. The duration of scanning time may be prolonged if no limited value time is set on the scanning analysis. Consequently, the scanning time should have a minimum value for the entire studied program.

#### IV. CONCLUSIONS

The analysis and detection of malware can be a very time-consuming and challenging task. In this paper we have described Metaware Malware Analysis toolkits, an approach to automatically identifying the malicious code of an execution. In order to prevent any obfuscation code poses obstacles to the scanning and detection module, an automating process for identifying and extracting the hidden code bodies is proposed. Our malware scanning and detection method is based on combination of heuristic, disassembler and emulation concept. Inside this paper, the overall concept and algorithm of core component has been described.

#### REFERENCES

- [1] Jia Zhang, YunTao Guan, XiaoXin Jiang, Haixin Duan, Jianping Wu, "AMCAS: An Automatic Malicious Code Analysis System," IEEE Web-Age Information Management. WAIM'08, Zhangjiajie Hunan, pp. 501-507, 2008
- [2] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa, K. Nakao, "Malware Behavior Analysis In Isolated Miniature Network for Revealing Malware's Network Activity," Communications, 2008. ICC'08. IEEE International Conference, Beijing, pp. 1715-1721, 2008
- [3] (2010) Clam AntiVirus website [online]. Available: <http://www.clamav.net/lang/en/>
- [4] Creating Signatures for ClamAV. Clam AntiVirus [online]. Available: <http://www.clamav.net/doc/latest/signatures.pdf>
- [5] M. Bernhard Hämmerli, Robin Sommer, "Detection of Intrusion and Malware," 4<sup>th</sup> International Conference, DIMVA 2007, Lucerne, Switzerland, 2007, pp. 231-250
- [6] R. Sekar, A.K. Pujari, "BitBlaze: A New Approach to Computer Security via Binary Analysis," Springer-Verlag Berlin Heidelberg, 2008, pp. 1-25
- [7] Min Gyung Kang, Pongsin Poosankam, Heng Yin, "Renovo: A Hidden Code Extractor for Packed Executables," In Proceeding of the 5<sup>th</sup> ACM Workshop on Recurring Malcode (MORM), October 2007.
- [8] P. Royal, M. Halpin, D. Dagon, R. Edmonds, Wenke Lee, "Polyunpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware," Computer Security Applications Conference, ACSAC'06 22<sup>nd</sup> Annual, Miami Beach, FL, USA, 2006, pp. 289-300
- [9] D. Inoue, K. Yoshioka, M. Eto, Y. Hoshizawa, K. Nakao, "Malware Behavior Analysis in Isolated Miniature Network for Revealing Malware's Network Activity," Communications, ICC'08. IEEE International Conference, Beijing, 2008, pp. 1715-1721
- [10] Y.W. Cheng, "Fast Virus Signature matching Based on the High Performance Computing of GPU," *Communication Software and Networks*, pp. 513-515, March. 2010
- [11] W.Chen, T. Li, Y. Shi and Y. Fu, "A Worm Detection Model Based on Artificial Immunology," *Intelligent Information Technology Application*, vol. 1, pp. 64-67, Dec. 2009
- [12] (2010) VX heavens, [online]. Available: <http://vx.netlux.org/vx.php?id=es06>
- [13] M. Christodorescu and S.Jha. "Testing Malware Detectors," *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'04)*, 2004
- [14] Justin Ferguson, Dan Kaminsky, Jason Larsen, Luis Miras, and Walter Pearce. "Reverse Engineering Code with IDA Pro" ISBN 13:978-1-59749-237-9, 2008