An approach to fast malware classification with machine learning technique

Pham Van Hung

Faculty of Environment and Information Studies

Keio University

5322 Endo Fujisawa Kanagawa 252-0882 JAPAN

Submitted in partial fulfillment of the requirements

for the degree of Bachelor

Advisors:

Professor Hideyuki Tokuda

Professor Jun Murai

Associate Professor Hiroyuki Kusumoto

Professor Osamu Nakamura

Associate Professor Kazunori Takashio

Assistant Professor Rodney D. Van Meter III

Associate Professor Keisuke Uehara

Associate Professor Jin Mitsugi

Lecturer Jin Nakazawa

Professor Keiji Takeda

Abstract of Bachelor Thesis

An approach to fast malware classification with machine learning technique

With the rapid increase of malware, it is important for malware analysis to classify unknown malware files into malware families. By doing so, the behavior and characteristics of malware will be identified accurately. In this paper, an approach was introduced to perform fast malware classification based on meta-data of malware's file. A machine learning technique called decision tree algorithm, is used to classify malware rapidly and correctly. Experimental results of the malware samples show that the system successfully determined some semantic malware similarities, especially showed their inner similarities in behavior and static malware characteristic.

Keywords:
Malware, static analysis, decision tree.

Pham Van Hung

Faculty of Environment and Information Studies

Keio University

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Malware is a general word for all types of malicious software. Malware includes Virus, Trojan house, Back door, Worm, and other malicious software which are characterized by malicious code. Because of the widespread use of the Internet, computer users face many dangerous propagations of malware. The modern malware's purpose is commonly illegal profit. For example, a large number of computers are infected by keylogger and 24.3 billion USD is leveraged by e-payment system losing [1]. In addition, According to 2010 Annual Security Report, on May 2010, tens of millions computer world wide were infected by email worm such as I LOVE YOU, LOVE LETTER, LOVEBUG[2]. As a result tasks such as preventing, detecting, and removing malware are very important for network security.

1.2 Malware analysis problem

This session will be a detailed introduction on malware analysis of security professionals. Malware analysis is the process of analysing the purpose and functionally of a malware. Malware analysis purpose is understanding of characteristics that all viruses in a family have in common and create a set of signatures in order to detect malwares. In addition, the knowledge about the purpose and functionally of a malware is important for removal.

Commonly, dynamic and static malware analysis have been applied. When new malware is detected, dynamic malware analysis technique executes malware in the Virtual Machine using ProcMon, RegShot, and other tools. These tools are used to identify the general behavioral analysis techniques such as network traffic analysis, file system, and other Window features such as service,

process, and the registry.

However, the dynamic techniques are susceptible to a variety of anti-monitoring defenses, as well as *time bombs* or *logic bombs* and can be slow and tedious to identify and disable code analysis techniques to unpack the code for examination [3]. Furthermore, it takes large amount of time to prepare malware analyzing environment to analyze malware such as virtual machine environment. However, some malware can not be executed in those kind of environment.

With the static malware analysis technique, researchers perform reverse engineering using IDA Pro and Ollydbg tool to analyze malware based on its structure in order to discover its purpose and functionality but it takes a lot of time to see the malware structure.

Malware analysis is necessary to understand the behavior of malware. As a result, malware signature is created to effectively detect malware. Nevertheless, it wastes much time to find out the behavior and characteristic of malware.

With a vast amount of samples increasing day by day, it is harder for anti-virus industry and virus researchers to analyze malware without information of new malware. In order to reduce time of malware analysis, it is necessary to have an automatic malware classification system.

1.3 Approach

The thesis focused on two following issues:

- Automatically perform fast malware classification based on malware file's meta-data Use a machine learning technique, called decision tree algorithm to classify unknown malwares or subspecies rapidly and correctly.
- Help researcher to understand which family malware belongs to and detect some semantic information about malware.

For those reason, in this paper, an approach is proposed to perform fast malware classification based on malware's meta-data using machine learning technique known as decision tree.

1.4 Thesis outline

The rest of this thesis is structured as follows:

- Chapter 2 describes malware meta-data, and method in the static classification of malware. Insight is provided to understand the purpose of method given in this research.

- Chapter 3 presents the other malware static classification approach.
- Chapter 4 gives approach, design and operation of malware classification system.
- Chapter 5 mentions environment and implementation of static malware system.
- Chapter 6 shows the system evaluation method and results.
- Summary, the conclusion and future work are presented in chapter 7

Chapter 2

Background

At first, this chapter presents the growth of malware. Then, malware analysis technique is introduced. Furthermore, malware categories and the way to use Virus total service to identify malware names defined by anti-virus vendor are also introduced. Finally, there are problems of using malware name to detect semantic meaning of malware.

2.1 Growth of malware attack

Malware can self-replicate recursively. For example, Mota is a kind of worm that proliferate itself by sending spam email to addresses harvested on local machine [5]. In addition, Downadup is a malware that receives file through a peer-to-peer function[6]. Malware infects system and spreads to the other systems by communication tools such as the Internet and related technologies.

Since the rise of widespread broadband Internet access, the number of malware samples has been rapidly increasing. According to the report issued by Kaspersky's research team, 205 million pieces of malware were detected and neutralized [7]. In addition, in 2010 the number of malware samples increased by 20 millions [9].Figure 2.1 shows the increase of malware from 2003 to 2009 by Kaspersky Lab. As a result of the fast growing number of malware, there is a huge problem in Internet security and network connectivity.

Nowadays, malware is easily created by malware creation tools which is the programs which are used by attacker to generate malware[10]. In addition, unlike earlier attack tools implementing one type of attack, such tools now can be changed quickly by upgrading or replacing their code. This causes rapidly evolving attacks and, at the extreme, results in polymorphic tools that self-evolve, change with each active instance of the attack. Therefore, a large amount of malware have challenged anti-virus vendor and researcher to avoid their analysis.

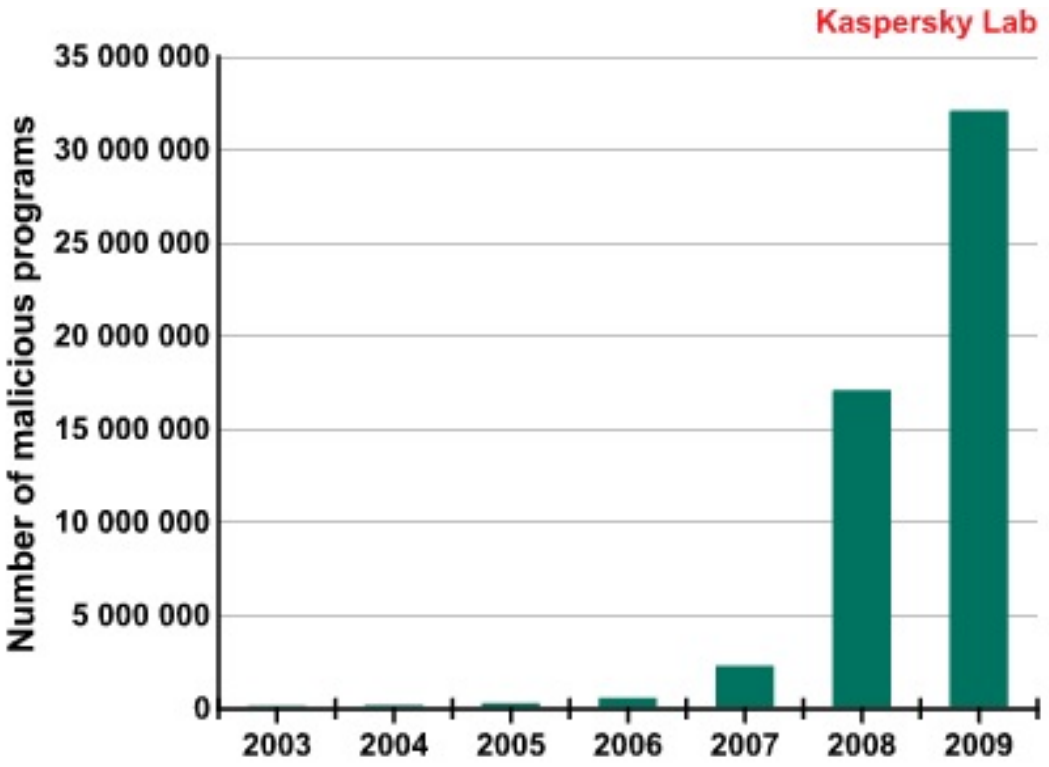

Figure 2.1: Number of malicious program

[9]

2.2 Malware avoidance technique

At present, malware is implemented with avoidance technique in order to invalidate static signature based method by anti-virus software and make analysis process more complicated. Avoidance technique can change malware signature and syntax without changing the behaviors of malware. The avoidance technique consists of code obfuscating and packing technique. Therefore, an avoidance technique causes more difficulty in malware analysis.

Code obfuscation changes the form of a malware instance to other form in order to invalidate signature based on detection technique; it consists of polymorphism and metamorphism [11]. Polymorphic technique modifies the representation of malware. Virus, worm, and other self-propagating software are often applied polymorphic technique such as encryption, data appending, and data prepending. Metamorphic malware automatically recodes itself when it distributed or executed[11]. Some simple metamorphic techniques are adding and varying lengths of NOP instructions, adding useless instruction, and looping the code segment. Advantage metamorphism techniques consist of function reordering, program flow modification, and static structure modification.

Packing malware can compress the Win32 portable execution file by several tools such as UPX,

Winpack, and ExeCryptor. According to a study carried out by Panda Research, 78% of new malware used some kinds of file packing to evade detection. PE-packer is designed to reduce the size of malware. The size of packed malware is small but it gets bigger when running in the system[12]. When uncompressed in the memory, packed malware is normally executed. UPX and some PE-packer compress malware binary files and make malware analysis harder.

For the reason that modern malware is implemented with avoidance technique, detect method by the use of static signatures is criticized for being ineffective.

2.3 Malware analysis technique

This section describes two malware analysis techniques including: dynamic and static analysis. The detail of two techniques is described as follow:

2.3.1 Dynamic malware analysis

Dynamic malware analysis technique is technique to find out the purpose of malware by running it and making sure what will happen in system. In general, malware is executed in the Virtual Machine. After that, malware researchers use SysAnalyzer, Process Explorer, ProcMon, RegShot, and other tools to identify the general behavioral analysis techniques. For example, SysAnalyser is a great analysis tool to monitors in many aspects of system and process states such as running process, open ports, loaded drivers, injected libraries, key registry changes, APIs called by a target process, file Modifications, http, IRC, and DNS traffic. Figure 2.2 shows the example of using SysAnalyser tool to analyse malware.

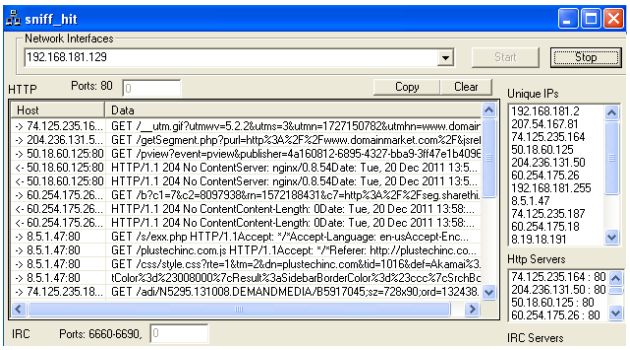

Figure 2.2: SysAnalyser tool for dynamic analysis.

The advantage of dynamic malware analysis technique is simple. The process of malware analysis is running malware and making sure what will happen in system.

However, there are some disadvantage of dynamic malware analysis technique. At first, these dynamic analysis techniques are vulnerable to a variety of anti-monitoring defenses such as *time bombs* and *logic bombs*. Additionally, these techniques can be slow and tedious to identify and disable. In addition, a dynamic analysis may fail to identify the malware if the behavior of malware is not logged during the analysis process. Furthermore, Much time is required to prepare environment for malware analysis, such as virtual machine environment or sandbox. Moreover, Some malware cannot be executed in virtual machine environment. In conclusion, dynamic technique is not enough for malware analysis.

2.3.2 Static malware analysis

Static malware analysis is a technique that identifies malware program without executing it. With the static malware analysis technique, researcher performs reverse engineering by using disassemble tools, decompile tools, source code analyzer tools such as IDA pro and Ollydbg in order to understand malware by seeing the structure of malware. Static malware analysis has an advantage that it can completely discover the purpose and functionality of malware. However, research needs time to understand the malware functionality by analysing malware structure.

In addition, most modern malwares use packer to modify itself. Packer is a program that modify other program files to compress their content. When a packer compress, encrypts, or modifies an executable program, the program looks much different. For analysing malware by Ollydbg or IDA pro, malware must be unpacked. PEid is a program that can find the signature of a known compiler or packer. Figure 2.3 shows that PEiD is applied to identify identifies the compiler and packer used by the malware author.

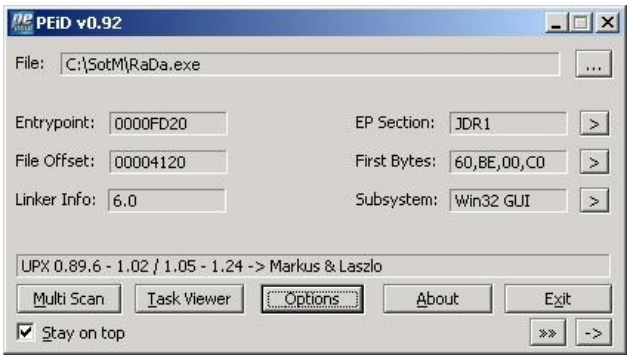

Figure 2.3: PEid identify the packer used by the malware author.

Ollydbg is a debugger that shows the assembly instruction of a program as they execute. Breakpoints can be setted on specific instructions to pause execution. Single step through the program

one instruction at a time, or let the program run normally. Figure 2.4 show the example of using OllyDbg tool. There are four main windows in Ollydbg debugger: the code window in the center, the register window on the right, the stack window in the bottom right hand corner and the memory dump window in the bottom left corner.

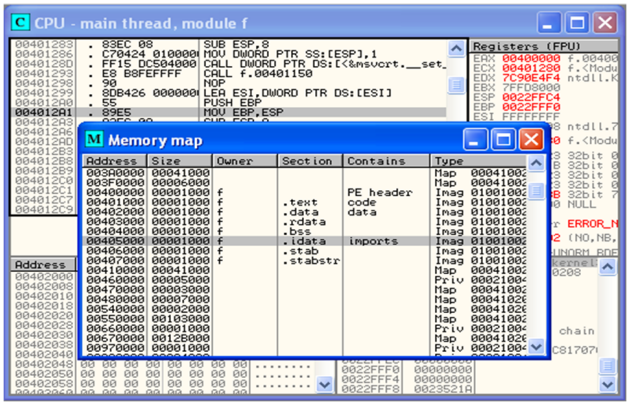

Figure 2.4: Ollydbg tool for static analysis.

IDA pro is a disassembly tool used by many researchers in different areas or reverse engineering. In addition to the disassembly listing, IDA pro also provides control-flow graphs. Figure 2.5 shows a control-flow graph provided by IDA pro.

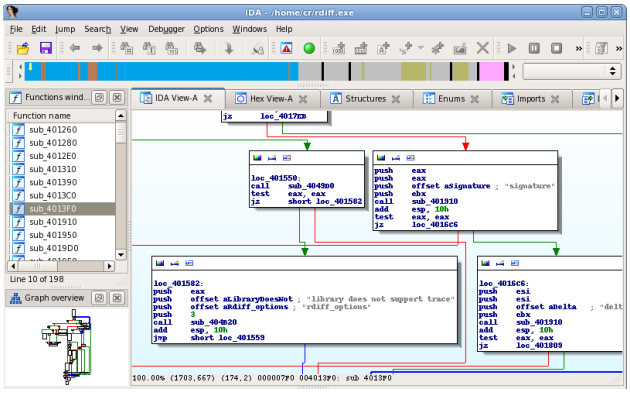

Figure 2.5: IDA pro for static analysis.

2.4 Malware categories

In general, malware is classified into a few categories or types based on behaviors, method of infection, and the resulting propagation of malware. For example, there are some malware categories: virus, trojan, worm, spyware, and rookit. Each category have specific types of malware threats:

- Virus :"Software which infects other application and use them as a spreading medium"[11].
- Trojan :"A malicious application which present itself as something else"[11].
- Worm :"Code with ability to spread from computer to computer by means of different network protocols"[11].
- Spyware :"Application aiming to havest personal information"[11].
- Rookit :"Hidden tools providing stealth services to its writer"[11].

However, the differences between the categories are a bit fuzzy, and the classes are obviously not exclusive. In addition, depending on purpose of virus industry, a unique malware can belong to rookit, virus, or spyware. The detail of malware categories is presented in the next section.

2.4.1 Use virus total to detect the name of categories.

In this thesis, MD5 hash is used to detect malware name provided by many anti-virus vendor. An MD5 hash is generated by MD5 Message-Digest Algorithm, a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. An MD5 hash is typically expressed as a 32-digit hexadecimal number, and MD5 is not collision resistant[29].

2.4.2 Using virus total to getting vendor name

In this paper, MD5 hash of malware is used to search the name of malware by virus total. Virustotal is a web service that analyzes malware files and facilitates in order to quick detect of viruses, worms, trojans, and all kinds of malware provided by antivirus engines. Malware's name is provided by various anti-virus vendor, but there are many names for unique malware. The Figure 2.6 reveals malware name detected by antivirus engines.

2.5 Problems of malware name

As the malware name detected by anti-virus engines given in Figure 2.6, unique malware is not classified into unique name. For each anti-virus vendor, the name of the malware detected different to the other anti-virus vendor. The classification of each anti-virus engine is unlike others. Therefore, malware detected by anti-virus engines cannot provide meaningful characterization of malware for virus researcher. In order to overcome this problem, this paper is proposed to classify the malware into families based on specific target and its operation behavior. As a result, the new malware families are required for detect meaningful characterization of malware.

Antivirus	Version	Last Update	Result
AhnLab-V3	2011.08.13.00	2011.08.13	Win-Trojan/Downloader.6144.QR
AntiVir	7.11.13.37	2011.08.12	TR/Crypt.XPACK.Gen
Antiy-AVL	2.0.3.7	2011.08.13	Trojan/Win32.Agent.gen
Avast	4.8.1351.0	2011.08.14	Win32:Trojan-gen
Avast5	5.0.677.0	2011.08.14	Win32:Trojan-gen
AVG	10.0.0.1190	2011.08.14	Win32/Heur
BitDefender	7.2	2011.08.14	Trojan.Generic.304464
CAT-QuickHeal	11.00	2011.08.13	TrojanDownloader.Agent.pkw
ClamAV	0.97.0.0	2011.08.13	Trojan.Downloader-38041
Commtouch	5.3.2.6	2011.08.13	W32/Downldr2.BXKD
Comodo	9739	2011.08.14	TrojWare.Win32.TrojanDownloader.Small.OBW
DrWeb	5.0.2.03300	2011.08.14	Trojan.DownLoader.61643
Emsisoft	5.1.0.8	2011.08.14	Trojan-Dropper.Agent!IK
eSafe	7.0.17.0	2011.08.10	Win32.Agent.pkw
eTrust-Vet	36.1.8499	2011.08.12	-
F-Prot	4.6.2.117	2011.08.13	W32/Downldr2.BXKD
F-Secure	9.0.16440.0	2011.08.14	Trojan.Generic.304464
Fortinet	4.2.257.0	2011.08.14	W32/Agent.PKW!tr.dldr
GData	22	2011.08.14	Trojan.Generic.304464
Ikarus	T3.1.1.107.0	2011.08.14	Trojan-Dropper.Agent
Jiangmin	13.0.900	2011.08.13	TrojanDownloader.Agent.aavi
K7AntiVirus	9.109.5010	2011.08.12	Trojan-Downloader
Kaspersky	9.0.0.837	2011.08.14	Trojan-Downloader.Win32.Agent.pkw

Figure 2.6: Malware name is detected by antivirus engines.

2.6 Malware families are used in this paper

This paper uses malware families which are reported by Information-technology Promotion Agency [14]. The table indicates malware families used in this paper such as Win32/Virut, Win32/Autorun, Win32/IRCbot, Win32/Gaobot, Win32/Waledac, Win32/Downadup, Win32/Sality, and W32.Mota.

Malware familes	Summary
Win32/Virut	"Win32/Virut is a family of file infecting viruses that target and infect .EXE and .SCR files accessed on infected systems. Win32/Virut also opens a backdoor by connecting to an IRC server, allowing a remote attacker to download and run files on the infected computer." [16]
Win32/Autorun	"Win32/Autorun is a family of worms that spreads by copying itself to the mapped drives of an infected computer. The mapped drives may include network or removable drives." [15]
Win32/IRCbot	"Win32/IRCbot is a large family of backdoor Trojans that targets computers running Microsoft Windows. The Trojan drops other malicious software and opens a backdoor on the infected computer to connect to IRC servers. The Trojan can maintain multiple IRC server connections simultaneously to receive commands from attackers." [17]
Win32/Gaobot	"The Win32/Gaobot worm family spreads using different methods, depending on the variant. Some variants spread to machines with weak passwords. Others exploit vulnerabilities to infect machines. Once a machine is infected, the worm connects to an IRC server to receive commands." [18]
Win32/Waledac	"Win32/Waledac is a trojan that is used to send spam. It also has the ability to download and execute arbitrary files, harvest email addresses from the local machine, perform denial of service attacks, proxy network traffic and sniff passwords." [19]
Win32/Downadup	"Win32/Downadup attempts to spread to network shares by brute-forcing commonly used network passwords and by copying itself to removable drives." [6]
Win32/Sality	"Virus:Win32/Sality is a family of polymorphic file infectors that target Windows executable files with the extensions .SCR or .EXE. They may execute a damaging payload that deletes files with certain extensions and terminates security-related processes and services." [20]
W32.Mota	W32.Mota is a worm that propagates by sending itself to email addresses gathered from the computer." [5]

Table 2.1: Malware families

Chapter 3

Related research

This chapter presents the recent approach for automatically classifying malware into malware families. For the reason that the detection based on static signature is no longer effective in chapter 2, new malware classification method is required to detect malware using avoidance technique.

3.1 Flow graph

Another approach is using emulator to automatically unpack the packing malware, then from reverse code produce flowgraph, flowgraph matching to perform classification[8]. The system is created by the approach as the followings:

- Fist, System automatically unpacks malware based on application level emulation, and then uses entropy analysis to detect.
- Second, Produce control followgraph by using graph invaiant based on signature in order to measure similarity between malware.
- Next, Genetic string based on control flow signature in order to be able to use string edit distancce. Automatically unpack malware based on application level emulation.
- Finally, Malware classification uses a set of similarity functions and a set of similarity search algorithms to identify benign and malicious classes.

The disadvantage in flowgraph approach is high cost of runtime complexity. Firstly, runtime complexity of malware classification is $O(N \log M)$ where M is the number of control follow graphs in the database, and N is the number of control follow graph input binary [8]. In addition, to

identify two flowgraphs is N^3, considering N is the number of nodes in each graph. Further more, needs to unpack the sample if it was packed with an executable packer. Therefore, this approach is ineffective in malware classification system with a large number of instances.

In addition, malware classification based on flowgraph approach cannot accurately detect metamorphic malware. As presented in chapter 2, metamorphic malware can recode itself with program follow modification, function reordering. As a result, it is hard to classify malware based a set similarity between follow graph.

3.2 Optimizing decision tree in malware classification system using Genetic Algorithm

Malware classification system can use machine learning classifier. The main idea of machine learning technique classifier is a search algorithm that is learnt from externally supplied instances to produce a concise model of the distribution, which then makes prediction about new instances. Current machine learning classifier in malware classification include: Naive Bayse, Suport Vector machine, Decision tree, K-nearest Neighbor [25]. This approach uses combining Genetic algorithms with Decision tree. The data set is separated in training data set and testing data set. The data set includes:

```
// Malware classes for chromosome
hash_map<MalwareClass*,int> _Dataclasses;
hash_map<MalwareClass*,int> _Appsclasses;
hash_map<MalwareClass*,int> _Sysclasses;
hash_map<MalwareClass*,int> _Dosclasses;
```

Genetic algorithm is a method analogous to the process of natural evolution. Genetic algorithm is used to optimize decision tree for accurately classifying malware. Malware classification system, which is implemented by the combining generic algorithm with decision tree algorithm approach, is to classify malware into two classes: benign program and malicious program, not to detect semantic characterization of malware by classifying malware into families. Therefore, the combining generic algorithm with decision tree algorithm approach cannot be used to detect semantic characterization of malware.

3.3 Conclusion

To achieve fast malware classification, two approaches described as above can not be used in the system. New approach for fast malware classification system will be introduced in the next chapter.

Chapter 4

Classification based on malware's meta-data using decision tree approach

This chapter describes Win32 executive files structure, known as PE files structure, which is used in classification system as malware meta-data. Afterwards, the machine learning technique called decision tree method is mentioned in classification malware system. Finally, we will propose the classification approach based on malware's meta-data using decision tree technique.

4.1 PE file format[13]

4.1.1 The PE File Format

The Portable Executable (PE) file format has been designed to be used by all Win32 based system. The general layout of a PE file is given in Figure 4.1 . PE header consists of four parts: a MS-DOS session including DosMZ header and Dos stub; PE header; a section table; and images pages consisting of several different regions such as .text session, and .data session.

The PE file format starts with DOS MZ header. The first two bytes - "MZ" of the header is the DOS EXE signature. The first few hundred bytes of the typical PE file are taken up by the MS-DOS stub. This stub is a tiny program that displays a string as "This program cannot be run in MS-DOS mode.".

The following part is PE header. The fields of this structure contain only the most basic

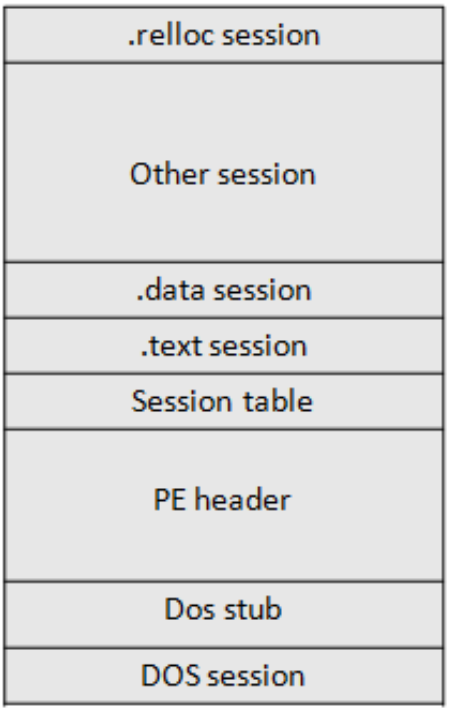

Figure 4.1: The general layout of a PE file.

information about the file such as the locations and sizes of the code and the data areas. These data structures of PE header will be explained shortly.

Following the PE header is the session table. The section table is an array of IMAGE_SECTION_HEADERs structures. An IMAGE_SECTION_HEADER provides information about its associated section such as location, length, and characteristics. The number of elements in this array is given in the PE header, known as *NumberOfSections* field. There are commonly at least two sections in a PE file: code and data. Figure 4.2 shows a section table for a typical PE file.

That is all about the physical layout of the PE file format. The major steps in loading a PE file into memory is described as follows:

- When the PE file is running, the PE loader examines the DOS MZ header for the offset of the PE header. If it is found, it would skip to the PE header.
- The PE loader checks if the PE header is valid. Then, it goes to the end of the PE header if the PE header is valid.
- Following the PE header is the section table. The PE header reads information about the sections and maps those sections into memory using the file mapping.

```
01 .text     VirtSize: 00005AFA  VirtAddr:  00001000
    raw data offs:   00000400  raw data size: 00005C00
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00009220  line #'s:      0000020C
    characteristics: 60000020
      CODE  MEM_EXECUTE  MEM_READ

  02 .bss      VirtSize: 00001438  VirtAddr:  00007000
    raw data offs:   00000000  raw data size: 00001600
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00000000  line #'s:      00000000
    characteristics: C0000080
      UNINITIALIZED_DATA  MEM_READ  MEM_WRITE

  03 .rdata    VirtSize: 0000015C  VirtAddr:  00009000
    raw data offs:   00006000  raw data size: 00000200
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00000000  line #'s:      00000000
    characteristics: 40000040
      INITIALIZED_DATA  MEM_READ

  04 .data     VirtSize: 0000239C  VirtAddr:  0000A000
    raw data offs:   00006200  raw data size: 00002400
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00000000  line #'s:      00000000
    characteristics: C0000040
      INITIALIZED_DATA  MEM_READ  MEM_WRITE

  05 .idata    VirtSize: 0000033E  VirtAddr:  0000D000
    raw data offs:   00008600  raw data size: 00000400
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00000000  line #'s:      00000000
    characteristics: C0000040
      INITIALIZED_DATA  MEM_READ  MEM_WRITE

  06 .reloc    VirtSize: 000006CE  VirtAddr:  0000E000
    raw data offs:   00008A00  raw data size: 00000800
    relocation offs: 00000000  relocations:   00000000
    line # offs:     00000000  line #'s:      00000000
    characteristics: 42000040
      INITIALIZED_DATA  MEM_DISCARDABLE  MEM_READ
```

Figure 4.2: A section table for a typical PE file.

- After the PE file is mapped into memory, the PE loader concerns itself with the logical parts of the PE file, such as the import table.

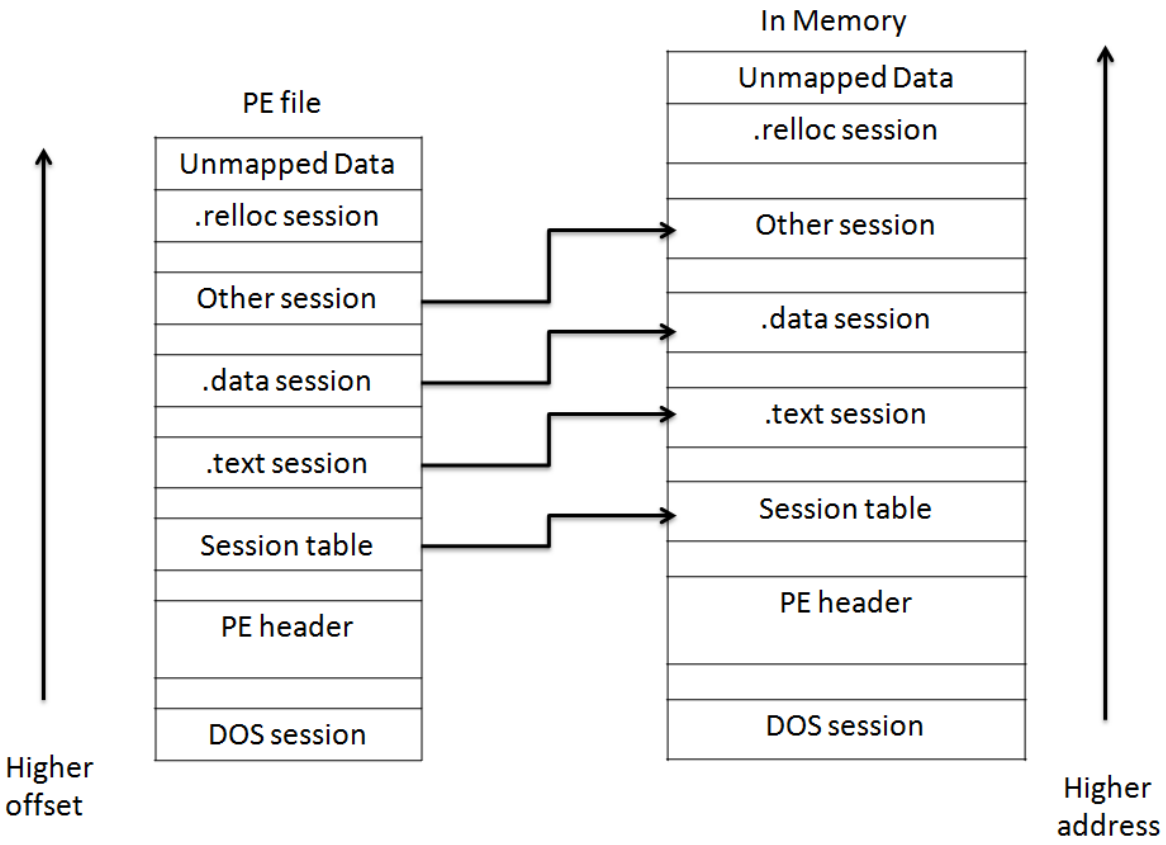

Figure 4.3: PE file format.

4.1.2 The PE Header

The PE header is the second part of the PE files structure. The PE header starts with two characters "PE" known the PE header file signature. The main PE header is a structure of type IMAGE_NT_HEADERS, which is defined in WINNT.H. The structure of PE header consists of a DWORD and two substructures composed of FileHeader and OptionalHeader. IMAGE_NT_HEADERS contains the following three fields:

```
DWORD Signature;
IMAGE_FILE_HEADER FileHeader;
IMAGE_OPTIONAL_HEADER OptionalHeader;
```

Following the PE signature DWORD in the PE header is a structure of type IMAGE_FILE_HEADER.

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
```

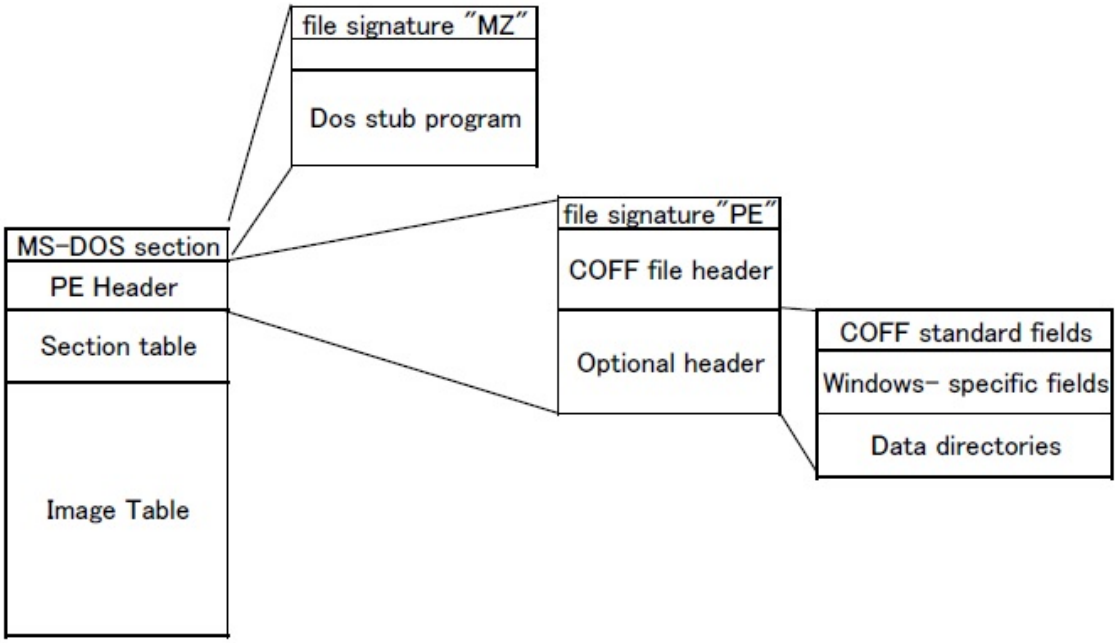

Figure 4.4: Layout a file in the PE header format.

```
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

The fields of this structure contain only the most basic information about the file. The fields of the IMAGE_FILE_HEADER include:

- Machine: The CPU that this file is intended for.
- NumberOfSections: The number of the members in section
- TimeDateStamp: The time that the linker
- PointerToSymbolTable: The file offset of the COFF symbol table.
- NumberOfSymbols: The number of symbols in the COFF symbol table.
- SizeOfOptionalHeader: The size of an optional header that can follow this structure.

- Characteristics: The size of an optional header can follow this structure. If there are no relocations in this file, characteristics will be 0x0001. Otherwise, the file is an executable image, characteristics will be 0x0002. If the file is a dynamic-link library, not a program, characteristics will be 0x2000.

Following the file header is the optional header which consists of three parts: COFF standard fields, Windows specific fields, and data directories. The first part includes the COFF standard field which contains the sizes of various parts of code, and the *AddressOfEntryPoint* which indicates the location of the entry point of the application and the location of the end of the Import Address Table(IAT). Then, the second part contains the information of OS. Finally, the Data directories field contains the information of Session table.

The following is the structure of optional header.

```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD                 Magic;
  BYTE                 MajorLinkerVersion;
  BYTE                 MinorLinkerVersion;
  DWORD                SizeOfCode;
  DWORD                SizeOfInitializedData;
  DWORD                SizeOfUninitializedData;
  DWORD                AddressOfEntryPoint;
  DWORD                BaseOfCode;
  DWORD                BaseOfData;
  DWORD                ImageBase;
  DWORD                SectionAlignment;
  DWORD                FileAlignment;
  WORD                 MajorOperatingSystemVersion;
  WORD                 MinorOperatingSystemVersion;
  WORD                 MajorImageVersion;
  WORD                 MinorImageVersion;
  WORD                 MajorSubsystemVersion;
  WORD                 MinorSubsystemVersion;
```

```
  DWORD                 Win32VersionValue;
  DWORD                 SizeOfImage;
  DWORD                 SizeOfHeaders;
  DWORD                 CheckSum;
  WORD                  Subsystem;
  WORD                  DllCharacteristics;
  DWORD                 SizeOfStackReserve;
  DWORD                 SizeOfStackCommit;
  DWORD                 SizeOfHeapReserve;
  DWORD                 SizeOfHeapCommit;
  DWORD                 LoaderFlags;
  DWORD                 NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

The fields of Optional header are described as follow:

- Magic: appears to be set to 0x010B.
- MajorLinkerVersion: "The major version number of the linker."
- MinorLinkerVersion: "The minor version number of the linker."
- SizeOfCode: "The size of the code section, in bytes, or the sum of all such sections if there are multiple code sections."
- SizeOfInitializedData: "The size of the initialized data section, in bytes, or the sum of all such sections if there are multiple initialized data sections."
- SizeOfUninitializedData: "The size of the uninitialized data section, in bytes, or the sum of all such sections if there are multiple uninitialized data sections."
- AddressOfEntryPoint: "A pointer to the entry point function, relative to the image base address. For executable files, this is the starting address. For device drivers, this is the address of the initialization function. The entry point function is optional for DLLs. When no entry point is present, this member is zero."

- BaseOfCode: "A pointer to the beginning of the code section, relative to the image base."
- BaseOfData: "A pointer to the beginning of the data section, relative to the image base."
- ImageBase: "The preferred address of the first byte of the image when it is loaded in memory."
- SectionAlignment: "The alignment of sections loaded in memory, in bytes."
- FileAlignment: "The alignment of the raw data of sections in the image file, in bytes."
- MajorOperatingSystemVersion: "The major version number of the required operating system."
- MinorOperatingSystemVersion: "The minor version number of the required operating system."
- MajorImageVersion: "The major version number of the image."
- MinorImageVersion: "The minor version number of the image."
- MajorSubsystemVersion: "The major version number of the subsystem."
- MinorSubsystemVersion: "The minor version number of the subsystem."
- Reserved1: "This member is reserved and must be 0."
- SizeOfImage: "The size of the image, in bytes, including all headers."
- SizeOfHeaders: "The combined size of the MS-DOS stub, the PE header, and the section headers, rounded to a multiple of the value specified in the FileAlignment member."
- CheckSum: "The image file checksum."
- Subsystem: "The subsystem required to run this image."
- DllCharacteristics: "The DLL characteristics of the image. "
- SizeOfStackReserve:"The number of bytes to reserve for the stack."
- SizeOfStackCommit: "The number of bytes to commit for the stack."
- SizeOfHeapReserve: "The number of bytes to reserve for the local heap."
- SizeOfHeapCommit: "The number of bytes to commit for the local heap."
- LoaderFlags: "This member is obsolete."

Condition	Class	Attribute 1	Attribute	
1	A	0	0	...
2	B	1	0	...
3	C	1	1	...
...	...	...	...	...

Table 4.1: The data table.

- NumberOfRvaAndSizes: "The number of directory entries in the remainder of the optional header. Each entry describes a location and size."
- DataDirectory: "A pointer to the first IMAGE_DATA_DIRECTORY structure in the data directory."

4.2 Decision tree[32]

Decision tree is a machine learning technique for building knowledge-based systems by inductive inference from examples. Decision tree is to create a model that predicts the value of a target variable based on several input variables. An example is shown on the Figure 4.5 from the data Table 4.1.Each interior node corresponds to one of the input variables. There are edges to children for each of the possible values of that input variable. Each leaf shows a value of the target variable.

In data mining, the decision tree technique can be described also as the combination of mathematical and computational techniques to aid the description, categorization and generalization of a given set of data.

Data comes in records of the form

$$(x, Y) = (x_1, x_2, x_3, ..., x_k, Y) \tag{4.1}$$

The dependent variable, Y which is the target variable is analysed to understand, classify or generalise. The vector x is composed of the input variables, x_1, x_2, x_3 that are used for that task.

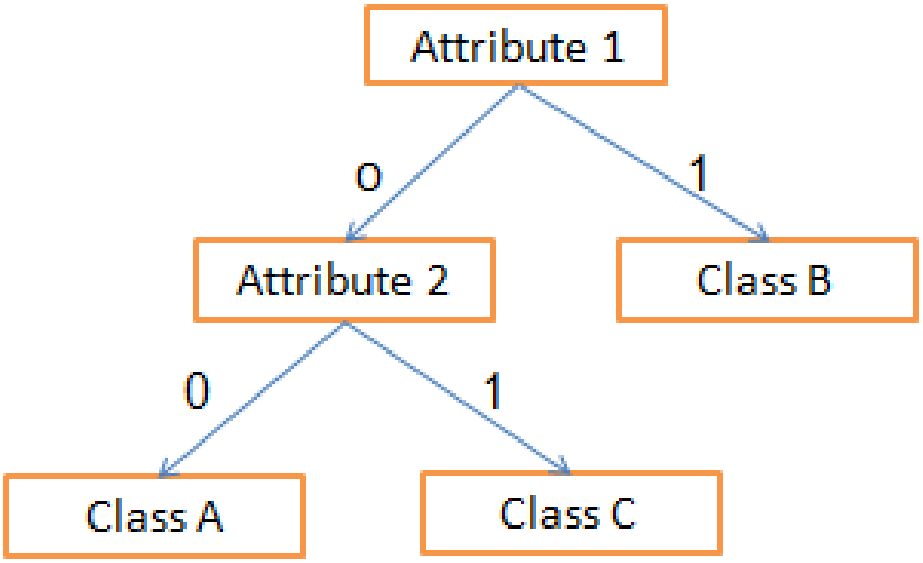

Figure 4.5: Decision tree sample.

4.3 Classification based on malware's meta-data using decision tree approach

To make malware classification rapid and correct, decision tree algorithm is used in malware classification system.

As the previous section, data comes in records of the form:

$$(x, Y) = (x_1, x_2, x_3, ..., x_k, Y) \tag{4.2}$$

In malware classification system, Y is malware families name, x_1, x_2,...,x_3 is malware file's meta-data. Malware name is provided by anti-virus engines that presented in chapter 2 cannot used in this system. The new malware families is required to detect meaningful characterization of malware. This classification system use famous malware families reported in Information technology agency such as netsky, mydoom, autorun, mytob, and other famous malware families. Therefore, when new malware is classified into famous malware families, researcher detect some meaningful characterize of malware, and reduce time for malware analysis.

4.4 Conclusion

In this chapter, our approach is proposed with the technique used in malware classification system. In the next chapter, propose the system design and implementation will be proposed.

Chapter 5

Implementation

In this chapter, the work of implementing malware classification system will be described. Starting with the implementing environment, this chapter will go through introduction of the meta-data used in this system, and finally system software implementation which is written in PHP language.

5.1 Environment

In proposing system, operator system, Database, and are all operated in one computer. Technical information of employed computer is shown as in Table 5.1. DecisionTree 1.5 is good choice to this implementation, with these following reasons:

- The library constructs a decision tree from multidimensional training data.
- The library use the decision tree thus constructed for classifying unlabeled data. [31].

5.2 Overview

The system uses machine learning technique, named decision tree algorithm for malware classification. The goal is classifying malware fast and decision tree algorithm is used to achieve. Classification is a form of supervised learning, which requires training data, with known input/output, to form knowledge [37]. As mentioned in Figure 5.1, the system contains three following parts.

- First part: Read binary file to take meta data and input to database.
- Second part: Manually cluster malware loaded from database.
- Third part: Use decision tree algorithm to classify malware.

Element	Attribute	Environment
OS	Computer	Ubuntu 11.04
Database	Mysql	Mysql 5.0
Programming language	C	gcc
	PHP	PHP 5.0
	Python	Python 2.7
Library	DecisionTree 1.5	ID3

Table 5.1: Implementation environment of the malware classification system.

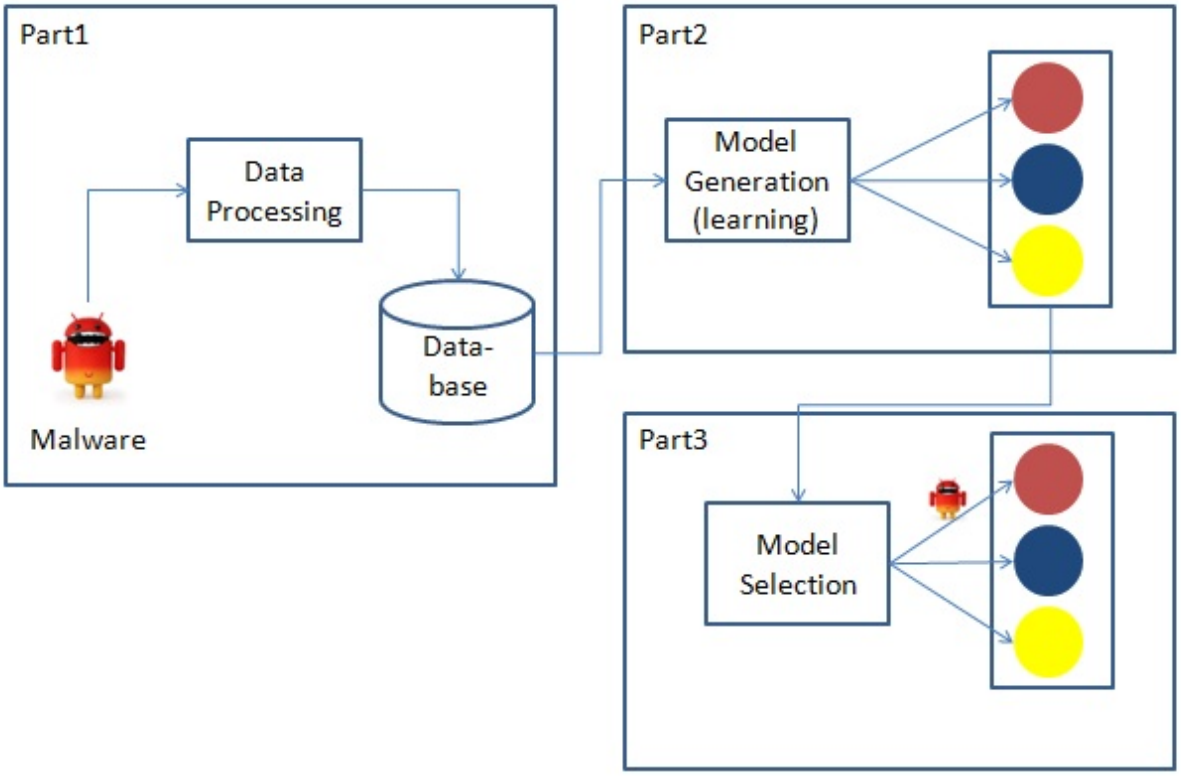

Figure 5.1: The system architecture.

A vast amount of malware need analyzing, and the system uses database to store malware file's meta-data, to easily control a large number of data and to share them in the internet among web browsers. The flow of data is shown in the Figure 5.1. At first, binary file meta-data are read, and all of the meta-data are input into database. In the second part, the system exports meta-data in database as the training data to create the decision tree for malware classifying. Finally, the

Meta-data classes	fields
PE header	Magic, MajorLinkerVersion, SizeOfCode, SizeOfInitializedData, SizeOfUninitializedData, AddressOfEntryPoint, BaseOfCode, BaseOfData, ImageBase,...
Dll	wininet1dll, ole32dll, user321dll, gdi32dll, advapi321dll, crtdlldll, ntdlldll, shell32dll,...
Session	UPX0, UPX1, UPX2, text, data, rdata, rsrc, bss, idata, rsrc1, aspack, packed, q5p6, q1rk2j, 4895f, t623ai, weitl, edata, ndata,..
API	WritePrivateProfileStringA, GetPrivateProfileStringA, WriteFile, ReadFile, MulDiv, FindNextFileA, FindFirstFileA, DeleteFileA, CopyFileA,...

Table 5.2: The data table.

decision tree algorithm created in the second step is utilized to class unknown malware into the different malware families.

5.3 Classification based on machine learning technique

5.3.1 Meta-data

Meta-data is simply data about data. The malware file's meta-datas are all simply information about malware file. PE header includes: meta-data of MS-DOS section; COFF file header; Optional header; Section header; and other types of data in sections such as API import and export table. Meta-datas contain many data about malware. The malware classification system based on can identify malware changed on signature and syntax by using code reordering, code rewriting, and instruction insertion. Additionally, malware's meta-data can be easily received. Therefore, in order to create a fast and high accuracy malware classification system, PE header's meta-data is good choice to classification malware system.

Malware file's meta-data are as shown in Table 5.2:

At first, there is a problem that PE file's meta data value is too large to detect unknown malwares which have meta-data's differences from the training data of the system. The meta-data

of PE header file are separated to give semantic information for malware classification. For example, normally *ImageBase* field of PE header file have value 400000h [4] so that the vaue of ImageBase of malware file in our system is 0 if it has 40000h and it is 1 if it has other value, and in this research that value is called as semantic value.

5.3.2 Create training data

In this system use decision tree technique. The attributes of decision tree described in previous subsection. In here, we mentions the method of identifying malware family name based on malware name provided by anti-virus vendors.

This research applies virustotal service to take the name of malware provided by the vendors and uses it to cluster malware into malware families with semantic information.

Figure 5.2 indicates HTTP post technique to automatically get name from Virus total service and cluster malware into malware families which given in chapter 2.

Malware families in this classification system are as follows: Win32/Virut, Win32/Autorun, Win32/IRCbot, Win32/Gaobot, Win32/Waledac, Win32/Downadup, Win32/Sality, and W32.Mota.

The disadvantage of using Virustotal to identify malware's families is that a unique malware has many different names provided by many antis-virus vendors. Therefore, if unique malware has two or more families name provided by anti-virus vendor, unique malware will be classed into the most reported malware family.

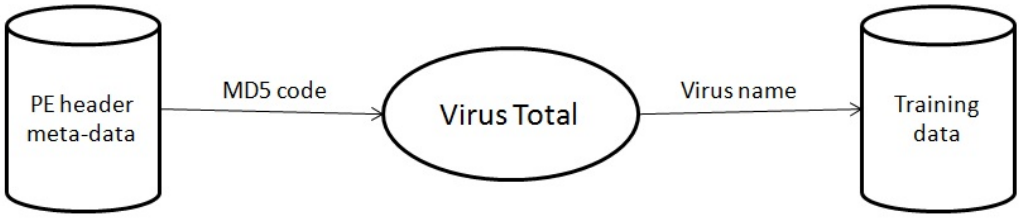

Figure 5.2: Clustering method.

5.3.3 Classification

To make malware classification rapid and correct, decision tree algorithm is utilized. Thus, it is easy to update a new malware family, the classification uses decision tree algorithm to determine each

malware family. Decision tree 1.5 library is good choice for implementing malware classification system.

Here is sample code:

Listing 5.1: Python Implementation of Decision tree 1.5

```
dt = DecisionTree( training_datafile = "training.dat", debug1 = 1 )
 dt.get_training_data()
 dt.show_training_data()
 root_node =        dt.construct_decision_tree_classifier()
 root_node.display_decision_tree("   ")
```

Malware' meta-data is used as input data for each decision tree, and the decision tree determines the malware family. Lists of malware PE header file's meta-data are Magic, MajorLinkerVersion, MinorLinkerVersion, SizeOfCode, SizeOfInitializedData, SizeOfUninitializedData, AddressOfEntryPoint, BaseOfCode, BaseOfData, ImageBase, SectionAlignment, FileAlignment, MajorOperatingSystemVersion, MinorOperatingSystemVersion, MajorImageVersion, MinorImageVersion, MajorSubsystemVersion, MinorSubsystemVersion, Reserved1, SizeOfImage, SizeOfHeaders, CheckSum, Subsystem, DllCharacteristics, SizeOfStackReserve, SizeOfStackCommit, SizeOfHeapReserve, SizeOfHeapCommit, LoaderFlags, NumberOfRvaAndSizes, Machine, NumberOfSections, TimeDateStamp, PointerToSymbolTable, NumberOfSymbols, SizeOfOptionalHeader, Characteristics, and other fields. The value of each field in PE header has semantics for decision tree algorithm if it is appeared at 10% in malware training data. With this approach, the semantic value list of meta-data malware is made. With this approach, the list semantic value of meta-data of malware is generated. For example, this is list meta-data created:1, 2, 6, 4, 0, 1, 1, 4, 0, 200, 4, 0, 0, 0, 4, 0, 0, 1.00E+00, 200, 0, 2, 0, 100000, 4000, 100000, 1000, 0, 10, 14c, 2, 1000, 8, 0, e0, 3. This is value before generating :35328, 10b, 2, 0, 6200, 0, 200, b32e, b000, 9000, 400000, 1000, 200, 3, 0, 0, 0, 4, 0, 0, d000, 400, afe8, 2, 0, 1000, 1000, 10000, 0, 0, 10, 14c, 6, 0, 0, 0, e0, 10e.

The decision tree exported by system is as the following code. Probabilities of each node is the probabilities of each families. The most probable family is the family of malware sample.

```
NODE 0:
   Branch features and values to this node: []
   Class probabilities at current node: [0.28916876574307304, 0.03853904282115869,
       0.020151133501259445, 0.0619647355163728, 0.01057934508816121,
       0.5531486146095718, 0.011083123425692695, 0.015365239294710327]
   Entropy at current node: 1.76730915987
```

```
   Best feature test at current node: 209
NODE 1:
   Branch features and values to this node: ['209=>0']
   Class probabilities at current node: [0.38588640275387265, 0.04819277108433735,
       0.0006884681583476765, 0.08296041308089501, 0.013769363166953529,
       0.45197934595524963, 0.00963855421686747, 0.006884681583476764]
   Entropy at current node: 1.56845724671
   Best feature test at current node: 44
NODE 2:
   Branch features and values to this node: ['209=>0', '44=>0']
   Class probabilities at current node: [0.4968425922724381, 0.054748822372797955,
       0.0005797736588481511, 0.11087157799969649, 0.013693701656603949,
       0.31032084114450614, 0.00944503996959897, 0.0034976509255101565]
   Entropy at current node: 1.1010285852
   Best feature test at current node: 32
NODE 3:
   Branch features and values to this node: ['209=>0', '44=>0', '32=>0']
   Class probabilities at current node: [0.5693425835153504, 0.06312750788349168,
       0.0006773557567694956, 0.12900589385502, 0.014474831409967637,
       0.21330422996733156, 0.00852684536125038, 0.0015407522508188685]
   Entropy at current node: 0.793704327232
   Best feature test at current node: 208
NODE 4:
   Branch features and values to this node: ['209=>0', '44=>0', '32=>1']
   Class probabilities at current node: [0.06609160326169744, 0.004967753320681031,
        0.0, 0.0031284685758396417, 0.009052702284896878, 0.8867347206306493,
       0.014900395603026747, 0.015124356323208915]
   Entropy at current node: 0.793704327232
   Best feature test at current node: 12
.............
.............
```

The virtualization decision tree is shown in Figure 5.3 . Each interior corresponds to one of the meta-data of new malware.

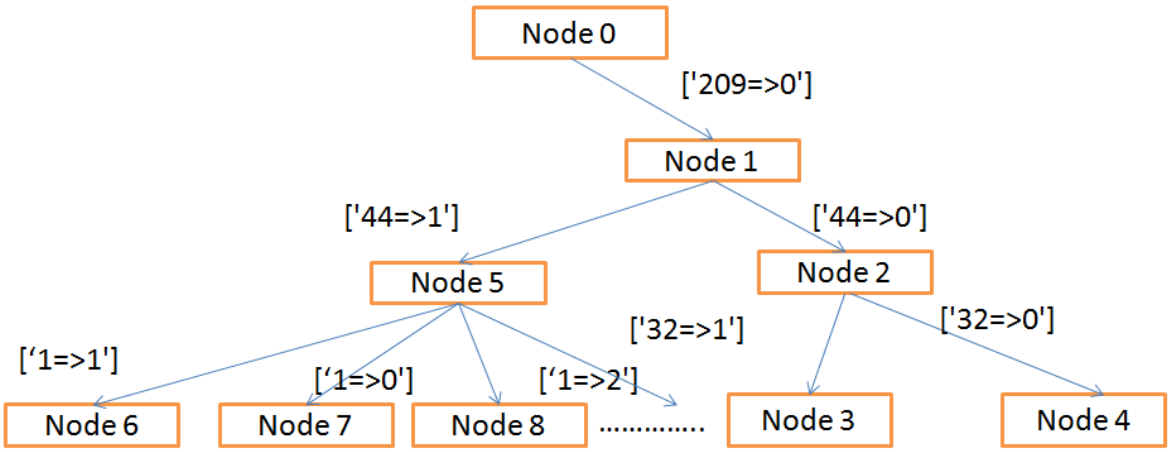

Figure 5.3: The virtualization decision tree.

Chapter 6

Evaluation

6.1 Accuracy evaluation

In 4436 malwares obtained, 4181 of them are to construct a decision trees. The number of each malware families is shown in Table 6.1. Then, 255 remaining malwares meta-data are kept to test experimental result the system.

In the decision tree, 255 malwares are chosen to check the experimental result of system.

The number of each malware families is shown in Table 6.2 Table **??** reveals that result. Some of the data in axis show these total number of malware in that family, and that number is separated into groups that these malwares has been classified by the system. For example, there are four malwares in Win32/Virut family, and three of them are successfully sorted into Win32/Virut family and, while one malware is in other family. Therefore, the system recognizes the trojan agent family with 75 % of accuracy.

6.2 Efficiency of classification

The accuracy of this classification system:

$$Accuracy = \frac{98 + 95 + 9 + 2}{355} = 57\% \tag{6.1}$$

The other classification is often classify malware into benign and malicious software. The system implemented in this thesis classify malware into the malware family. Therefore, system The system is useful to help virus researcher determine the malware family that unknown malware belongs to. The malware family contains Win32/Virut, Win32/Autorun, Win32/IRCbot, Win32/Gaobot, Win32/Waledac, Win32/Downadup, Win32/Sality, W32.Mota, known as famous malware family.

Malware	Number
Virut	2243
IRCbot	1169
Gaobot	265
Autorun	195
Downadup	103
Mota	79
Sality	66
WaledacMota	61

Table 6.1: Training data.

Malware	Number
Virut	200
IRCbot	100
Gaobot	20
Autorun	10
Downadup	10
Mota	5
Sality	5
WaledacMota	5

Table 6.2: Test data.

Malware	Virut	IRCbot	Gaobot	Autorun	Down-adup	Mota	Sality	Waledac	Acuracy	Training data
Virut	98	102	0	0	0	0	0	0	49%	2243
IRCbot	5	95	0	0	0	0	0	0	95%	1169
Gaobot	0	20	0	0	0	0	0	0	0%	265
Autorun	10	10	0	0	0	0	0	0	0%	195
Downadup	0	1	0	0	9	0	0	0	90%	103
Mota	2	3	0	0	0	0	0	0	0%	79
Sality	1	1	0	0	1	2	0	0	40%	66
Waledac	2	3	0	0	0	0	0	0	0%	61

Figure 6.1: Accuracy evaluation.

Virus researchers who know the family of malware can easily find out some semantic similarities between malwares and shows their inner similarity in behavior and static malware characteristics.

6.3 Efficiency of classification

The evaluation was performed on a 2.4 HZ core i3 laptop with 4G memory, running in ubuntu 11.4.

To evaluate the speed of the malware classification system, 2999 malware samples were evaluate the classification efficiency of the system. Figure 6.2 show the processing time of our approach. 1606 samples required approximately 0 seconds. 1392 required 0.01 second. Only one sample required 0.02 seconds. The median time samples to perform classification for each samples is approximately 0.05 seconds. 6.2. The median time compared to the median time of folowgraphs appoach introduced in chapter 3. The evaluation of folowgraphs appoach was performed on a 2.4 GHz Quad Core Desktop PC with 4G of memory, running 32-bit Windows Vista Home Premium with Service Pack 1[8]. The result was shown in figure 6.3. The median time to perform classification was 0.25 seconds. The slowest sample that is required 5.12 seconds. Only 6 samples required more than 2 seconds. Processing time of followgraph approach was shown in figure **??**.

According to the median time to perform classification, the system implemented by our approach is faster than the malware classification based on the followgraphs approach. The processing time of two approach is shown in Figure 6.4.

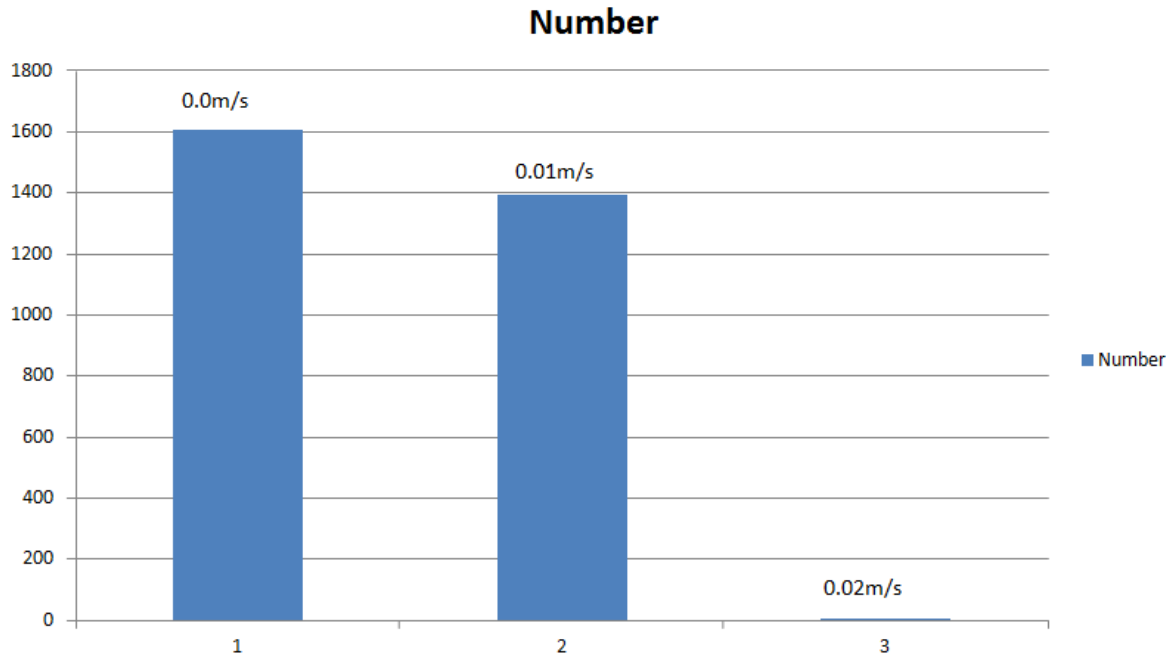

Figure 6.2: Processing time of our approach.

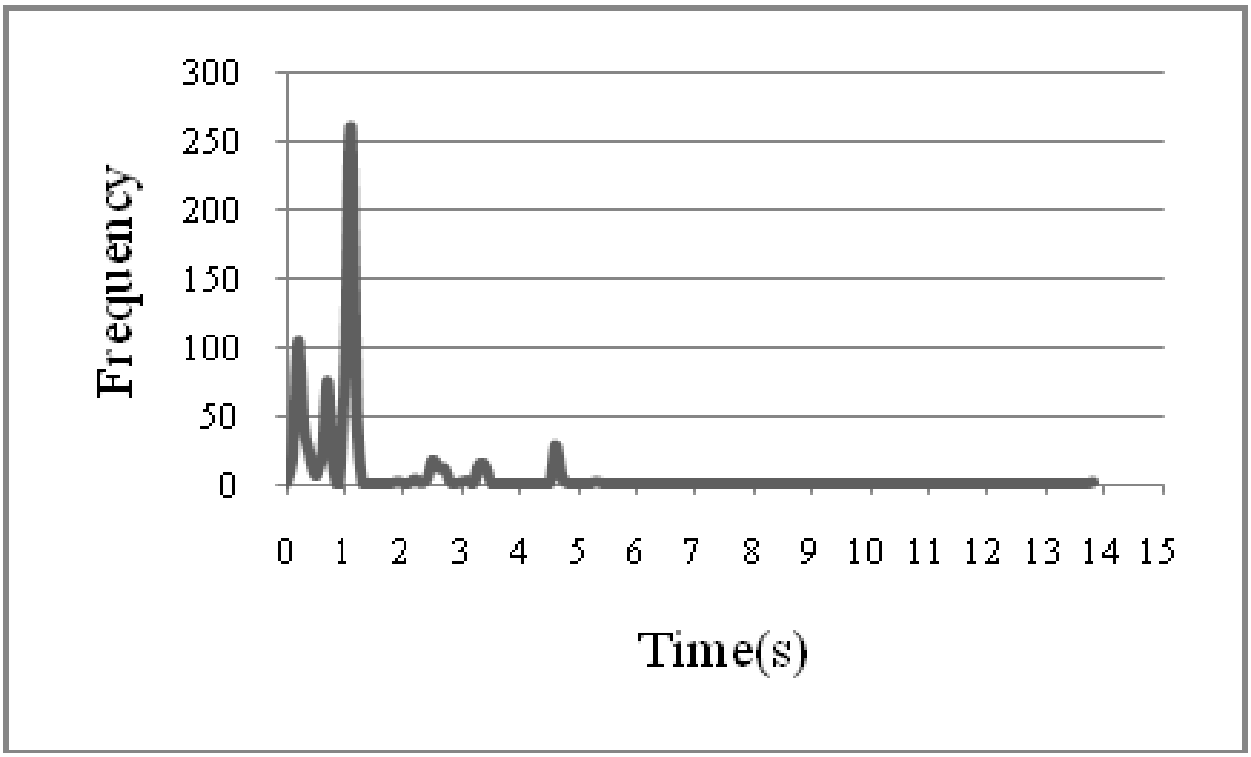

Figure 6.3: Processing time of followgraphs approach.

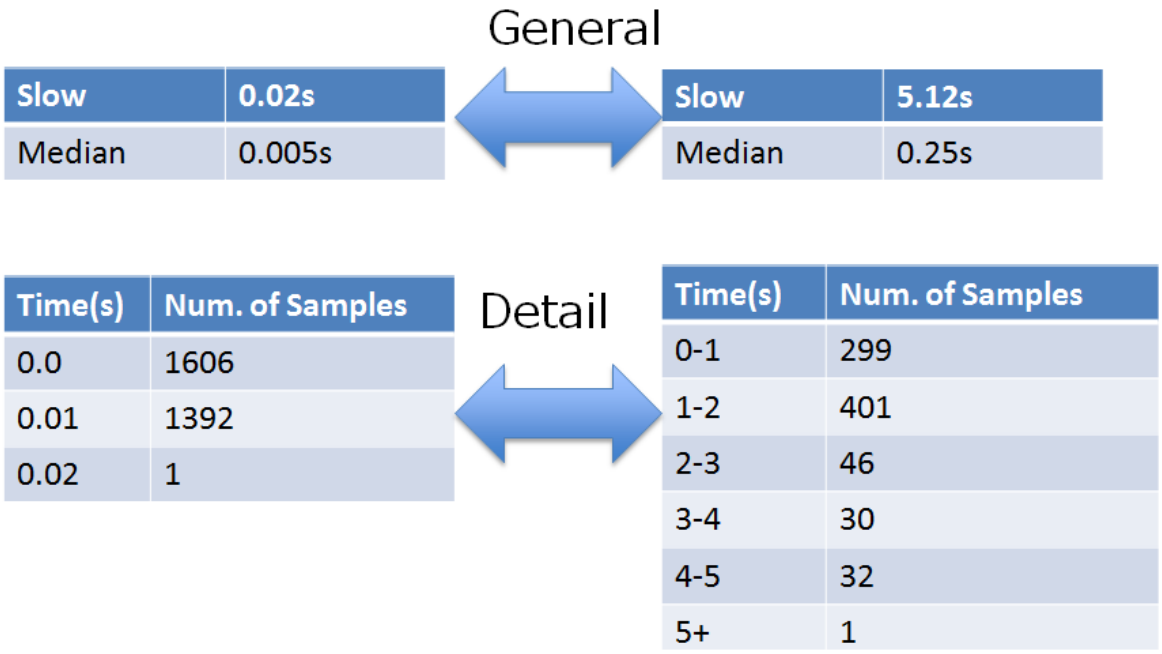

Figure 6.4: Processing time of followgraphs approach.

6.4 Discussion

Accuracy evaluation

Obviously, in experiment we obtained 4181 malware samples for constructing decision tree and 255 malware samples for evaluating system. The accuracy of our malware classification system is 57%. However, the accuracy of Mota, Waledac, Sality samples are too low because the number of those samples is less than Virut, or IRC bot samples. There are 2243 Virut, and 1169 IRC bot samples. Otherwise, there is only 79 Mota, 61 Waledac, 66 Sality samples.

Moreover, all of Gaobot samples is classified into IRC bot because the Gaobot binary file is nearly similar to IRC bot.

The number of Downadup samples in training data are small, but the accuracy of Downadup samples are higher than another. The reasons is Downadup binary files are different to other families so that decision tree easily identify Downadup samples.

Efficiency of classification

1606 malware samples required approximately 0 seconds. With our approach, malware samples use meta-data to compare with each node of decision tree. In some case, malware sample is only

correspond to one to two node in decision tree so that the result is exported instantly.

The result of processing time shows that our approach is faster than follow graphs approach. Motivation of our approach is fast classify malware into families. Only meta-data is used on our system. Additionally, our system is implemented by the decision tree technique which is faster than other classification technique. Therefore, the processing time of classification in our system is fast.

Chapter 7

Conclusion

7.1 Conclusion

Analyzing a great number of new malware samples each day is a difficult problem. The fast malware classification system is successful to classify unknown malware into malware families which have semantic specifications. It is strongly believed that the system is useful in malware analysis to determine malware behavior and semantic malware characteristics, and to more easily examine a large number of malwares.

However, there is a problem that the system only uses malware meta-data and cannot detect the malware family with same program structure.

Surprisingly, the system cannot be used to classify W32 malware which does not have the signature of PE file signature.

7.2 Future work

The anonymous malwares in the system are classified into malware family, and in the future work, the malwares shall be automatically unpacked before classifying malwares.

Acknowledgement

First and foremost, I am sincerely grateful to faculty members in Murai and Tokuda Laboratory, Professor Hideyuki Tokuda, Professor Jun Murai, Associate Professor Hiroyuki Kusumoto, Professor Osamu Nakamura, Associate Professor Kazunori Takashio, Assistant Professor Rodney D. Van Meter III, Associate Professor Keisuke Uehara, Associate Professor Jin Mitsugi, Lecturer Jin Nakazawa.

I would be extremely thankful to Professor Keiji Takeda for his endless help and valuable comments to my thesis. Without his help, I cannot even think of writing this thesis.

I give my special thank to Doctor Masayoshi Mizutani for his sincerely help and many advices since the time I first came to the lab.

I would like to thank to Mr.Kunihiko Shigematsu, Mr.Yuki Uehara, my friend Toshinori Usui, for listening and giving me many useful discussion points. Their gentleness and encouragement helped me overcome hard periods in this one year.

I would like to thank my friend Takuya Shibuta and Nguyen Tien Thanh, for their shape comments for the thesis structure and my English writing.

I would also like to thank all ISC group members, Mr.Toshinori Usui, Mr.Naoto Somi, Mr.Sekine Fuyuki, Mr.Vu Xuan Duong, Mr.Tomonori Yamamoto, Mr.Hiroki Yoshihara, Mr.Daido Yoshihara, Mr.Takao Oya, Mr.Reimon Arima, Mr.Hiroaki Kono, Mr.Takuya Yui, Mr.Makoto Komatsu, Mr.Do Trung Kien, Mr.Nguyen Anh Tien, Ms.Asuka Nakajima, Ms.Akane Mitsuki, Mr.Kouki Nakayasu, Mr.Kohei Tsuyuki, Mr.Yu Yoshida, Mr.Akifumi Fukaya, Ms.Urara Ozawa, who have given me numerous support and always cheer me up in my hard time.

I owe my deep gratitude to members in Murai and Tokuda Laboratory who have been so friendly and kindly supporting me with my research.

Lastly and most importantly, I am heartily thankful to my family, including my grandparents, my parents, my older sister, and my older brother. Especially Ms.Nghiem Thi Hang for her greatest support in order to finish this thesis.

Bibliography

[1] John Bambenek. *http://handlers.dshield.org/jbambenek/keylogger.html*. 13 Sep 2005.

[2] Symantec Announces MessageLabs Intelligence. *http://www.symantec.com/about/news/release/article.jsp?prid=2010120701*. 12 Dec 2011.

[3] Georg Wicherski, *peHash: A Novel Approach to Fast Malware Clustering*. December 7, 2008.

[4] Goppit,*PORTABLE EXECUTABLE FILE FORMAT*. 2011.

[5] Symantec.*http://www.symantec.com/security_response/writeup.jsp?docid=2004070412024899*. 01 Jan 2012.

[6] Symantec.*http://www.symantec.com/security_response/writeup.jsp?docid=2008112203240899*. 01 Jan 2012.

[7] *http://news.softpedia.com/news/Kaspersky-Report-161-Million-Network-Attacks-Blocked-232812.shtml*. 14 December 2011.

[8] Silvio CESARE, *Fast Automated Unpacking and Classificationof Malware*. Masters Thesis,Central Queensland University, 2010.

[9] Eugene Kaspersky. *http://www.kaspersky.co.in/news?id=207576357*. 14 Jun 2011.

[10] Malware protection center. *http://www.microsoft.com/security/portal/Threat/Encyclopedia/Glossary.aspx*. 21 Dec 2011.

[11] MalwareChet Hosmer, Chief Scientis.*Polymorphic & Metamorphic Malware*. Black Hat, 2008.

[12] Paul Craig.*PE Packers Used in Malicious Software*. Ruxcon, 2006.

[13] *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*. Microsoft Systems Journal. March 1994.

[14] Information-technology Promotion Agency, Japan.*http://www.ipa.go.jp/security/txt/list.html*. 01 Jan 2012.

[15] Malware Protection Center.*http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32%2fAutorun*. 01 Jan 2012.

[16] Malware Protection Center.*http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32%2fVirut*. 01 Jan 2012.

[17] Malware Protection Center.*http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?name=Win32%2FIRCBot#symptoms_link*. 01 Jan 2012.

[18] Malware Protection Center.*http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?=Win322fGaobotsummarylink*. 01 Jan 2012.

[19] Malware Protection Center.*http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32%2fWaledac*. 01 Jan 2012.

[20] Malware Protection Center.*http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32%2fSality*. 01 Jan 2012.

[21] David Zimmer. *http://www.woodmann.com/collaborative/tools/index.php/SysAnalyzer*. 21 March 2011.

[22] Wikipedia. *http://en.wikipedia.org/wiki/Decision-tree-learning*. 21 Dec 2011.

[23] Wikipedia. *http://en.wikipedia.org/wiki/Antivirus-software*. 14 Jun 2011.

[24] Qinghua Zhang, Douglas S. Reeve.*MetaAware: Identifying Metamorphic Malware*. 23rd Annual Computer Security Applications Conference 2007.

[25] Mohd Najwadi Yusoff and Aman Jantan.*Optimizing Decision Tree in Malware Classification System by using Genetic Algorithm*. The Society of Digital Information and Wireless Communications, 2011.

[26] S. B. Kotsiantis.*Supervised Machine Learning: A Review of Classification Techniques*. Informatica 31, 2007.

[27] Panda Security Internacional.*http://www.pandasecurity.com*. 01 Jan 2012.

[28] *http://win32assembly.online.fr/pe-tut1.html*. 14 December 2011.

[29] Wikipedia. *http://en.wikipedia.org/wiki/MD5*. 14 December 2011. Perdram amini, Ero Carrera *Reverse engineering on windows: Application in malicious code analysis*. Masters Thesis,Central Queensland University. 2010.

[30] *http://win32assembly.online.fr/petut1.html*. 17 Feb 2010.

[31] *http://pypi.python.org/pypi/DecisionTree/1.5*. 14 December 2011.

[32] *Induction of decision trees*. Machine learning 1, 2007. 17 Feb 2010.

[33] Jingjing Yao, Qiang Hou, *Malicious executables classification based on behavioral factor analysis*. International Conference on e-Education, e-Business, e-Management and e-Learning, 2010.

[34] Yuhei Kawakoya, Makoto Iwamura, Mitsutaka Itoh, *Analyzing Malware with Stealth Debugger*. Information Processing Society of Japan, 2008.

[35] Kevin Coogan, Saumya Debray, Tasneem Kaochar, Gregg Townsend, *Automatic Static Unpacking of Malware Binaries*. 16th Working Conference on Reverse Engineering, 2009.

[36] Tony Abou-Assaleh, Nick Cercone, Vlado Ke?selj, Ray Sweidan , *N-gram-based Detection of New Malicious Code*. Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004 .

[37] Tony Lee, Jigar J. Mody, *Behavioral Classification*. Presented at the EICAR Conference, May 2006.

[38] Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, *IMDS: Intelligent Malware Detection System*. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, 2007.

[39] Robin Sharp, *An Introduction to Malware*. Spring 2009.