Profile Hidden Markov Model for *Malware Classification* - Usage of System call Sequence for Malware Classification

Ramandika Pranamulia, Yudistira Asnar, Riza Satria Perdana
School of Electronics Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
ramandika@sudents.itb.ac.id, yudis@stei.itb.ac.id, riza@stei.itb.ac.id

***Abstract*— Malware technology makes it difficult for malware analyst to detect same malware files with different obfuscation technique. In this paper we are trying to tackle that problem by analyzing the sequence of system call from an executable file. Malware files which actually are the same should have almost identical or at least a similar sequence of system calls. In this paper, we are going to create a model for each malware class consists of malwares from different families based on its sequence of system calls. Method/algorithm that's used in this paper is profile hidden markov model which is a very well-known tool in the biological informatics field for comparing DNA and protein sequences. Malware classes that we are going to build are trojan and worm class. Accuracy for these classes are pretty high, it's above 90% with also a high false positive rate around 37%.**

Keywords—system call, malware, profile hidden markov model, obfuscation

I. Background

Use of computer system and internet has been increasing for the last five years. It means that people are in need to use computer and internet as their daily routine. Computer and internet also contribute a lot in the field of industry, there are a lot more industries where its business procedures are automated by computer systems and internet. In contradiction with that massive usage of computer and internet which brings a lot of advantages, there are still disadvantages for this situation. The cyber-attack has been increasing as well as the usage of computer and Internet. One of cyber-attack type is malware invasion. Malware is a software program that runs on a client computer which can bring disadvantage or cause harmful to the computer client system. In this internet era where we can get anything from anywhere by just downloading or viewing something makes malware easier to spread rapidly and vast.

Indeed malware has been developed so well since its discovery for the first time. Malware creator has developed many methods to trick malware analyst and security system such as anti-malware in a personal computer. One of these methods is called obfuscation. Obfuscation in software development term usually means a deliberate act of creating source or machine code that is difficult for humans to understand. It usually uses needlessly roundabout expressions to compose statements. Programmers may deliberately obfuscate code to conceal its purpose (security through obscurity) or its logic or implicit values embedded in it, primarily, in order to prevent tampering, deterministic reverse engineering, or even recreational challenge for someone reading the source code. Obfuscating line of codes can be done manually, or using an automated tool which is the way it's done industrially.

In malware creation, obfuscation or even now packer is being used to make static analysis hard to conduct. Static analysis is usually done by inspecting the binary file of malware without executing it and studying each component. The binary file can also be disassembled (or reverse engineered) by using a disassembler tool such as IDA in windows to be translated into assembly code which can be read and understood by human; the malware analyst can then make sense of the assembly instructions and have an image of what the program is supposed to perform.

Those methods can make actually same malware files become so different in perspective of file structure and content. Static analysis will be failing to detect same malware files with different obfuscation/packer technology or even trivial settings like encryption key that is being used by packer to encrypt part of malicious code. Although we can't detect those similar malware with static analysis, we can still perform dynamic analysis. Dynamic analysis is conducted by running malwares and observe their behaviors. Behaviors that are monitored, including file system operation, registry operation, API call, and network activity.

In this paper, we use only API call as our monitored behavior. As malware will always have fingerprint of malicious activity in their sequence of system call. We think that API call sequence can provide us sufficient information for classifying a file as a malware or a benign fie. Since we consider a sequence of API calls, things that matter to us are the type of an API call and its position in the sequence. Therefore, we use a profile hidden markov model(PHMM) as a sequence mining algorithm to process those two information.

Limitation of work presented in this paper is we don't take into account other advanced technologies rather than obfuscator and packer. We also only create two classes, worm and Trojan horse, in our project just to test how good API call and profile hidden markov model can be used to detect and classify a malware instance.

This paper is organized as follows: Section II explains the underlying concepts of this works, and continues by the illustration of this approach in Section II. We explain our method to build our PHMM classifier in section III. Testing experiment is explained in section IV and compare our result and findings with related works in section V. Finally we put our conclusion of this project in section VI.

II. Basic Cocepts

A. Malware

Malware, short for malicious software, is an umbrella term used to refer to a variety of forms of hostile or intrusive software [1], including computer viruses, worms, Trojan horses, ransomware, spyware, adware, and other malicious programs. Malware has been developed with many advanced technologies such as obfuscator and packer. Obfuscating is a method to make source or machine code being difficult for humans to understand [2]. While packer is a piece of software that takes the original malware file and compresses it, thus making all the original code and data unreadable. At runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program's original code [3]. These methods have been developed with the intention of hiding the malicious part of the software so it will pass some weak defense mechanism.

Malwares are also divided into some classes like worm and Trojan horse based on their behavior. Worm is a class of malwares which can replicate their self individually without help of other actors such as computer users, unlike virus which has to be executed first before it can replicate itself. Trojan horse on the other side is a class of malware which act like a functional software from user's perspective but running some malicious code behind like deleting all files, downloading other malware, or opening a backdoor.

B. Antimalware

Antimalware or antivirus is a software used to defend a system against malicious activity and recover the system from attacks [4]. Antimalware can combat malware in two ways

1. It can provide the real time protection against malware installation on a computer by monitoring all incoming networks and all installation files.
2. It can uninstall malware that has already been installed in a computer system by iterating through all files in the operating system.

Antimalware can also be divided into two category by its detection methodology, static and dynamic. Static antimalware analyze a file without running it, while dynamic antimalware run a file and monitor its behavior. In static method file hashing and string comparison are the main methods used. Static antimalware search for malicious files as well as partition code in a file that's malicious by file hashing and string comparison. In the dynamic method, behavior of the file is monitored such as activity of changing registry values, activity of modifying file systems, and network activity.

C. Profile Hidden Markov Model

Profile Hidden Markov Model(PHMM) is a type of Hidden Markov Model(HMM) which is a statistical model in form of probabilistic finite state machine. States in HMM are categorized into hidden state and emitting state. Hidden state is an unobservable state that has probability to emit emitting states which are observable. As we can see in Figure 1 below rainy and sunny are hidden states while walk, shop, and clean are emitting states. Each hidden state has probability to go to other hidden states, including itself like P(rainy→sunny) = 0.3 and P(rainy→rainy)=0.7. Hidden state also has probability to emit a specific emitting state like P(rainy→shop) = 0.4. There is also a start state and probability of being rainy or sunny from that start state, P(start→rainy) = 0.6 and P(start→sunny)=0.4. These two probabilities are known as initial probability, which is a probability of a sequence started with a give state in this case rainy or sunny.

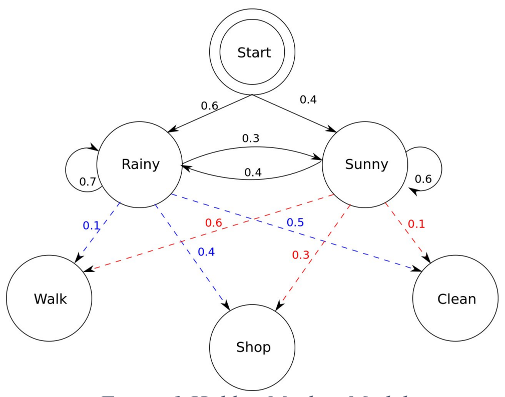


Figure 1 Hidden Markov Model

PHMM has a little bit different structure from HMM that can be seen on Figure 2. PHMM has three types of hidden state that are matched, insertion, and deletion state. Each state is a representation of each column sequence after being aligned. Also PHMM has no cycle since every hidden state represents each column sequence uniquely, because there are no two columns has the same predecessor and successor and emitting the same emitting states with similar probabilities.

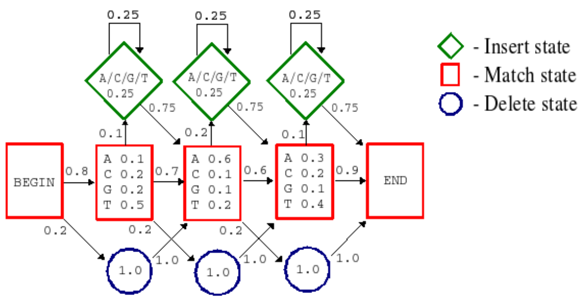


Figure 2 Profile Hidden Markov Model

PHMM is widely used for Bioinformatics studies in DNA sequence and protein sequence comparison. We implement this method to malware classification by changing the ATCG information or protein name with system call sequence. We encode system call name into an alphabet. We align sequences from malware instances with the same malware class and give the alignment result to PHMM.

III. Malware Classifier Designed

The program that is created for the purpose of malware classification in this paper consists of two different modules. The first module is the windows API call interceptor and the second one is the code that is used for processing the dataset to fit the HMMER framework. The architecture of this program can be seen at **Error! Reference source not found.** below in data flow diagram representation. We build malware models from a raw database by parse the raw data into a fasta file format which consists of system call sequences from malwares with the same malware class. These sequences must be aligned first before HMMER tool can process it, therefore we process them with clustal omega beforehand. A hooking program is created to capture system call sequence of a running malware program and compare it to the models using HMMER query. We run these malwares in a virtual environment using virtualbox. Specifications of implementation environment are as follows.

- OS: Windows 7 32 bit
- IDE: Microsoft Visual Studio 2015
- Programming language: Visual C++
- Hooking language : Easyhook Framework
- Other Tools: ClustalOmega(Online Tools)

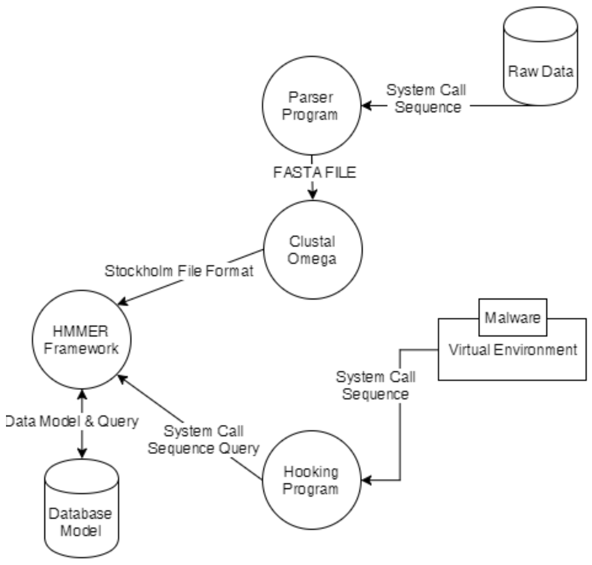


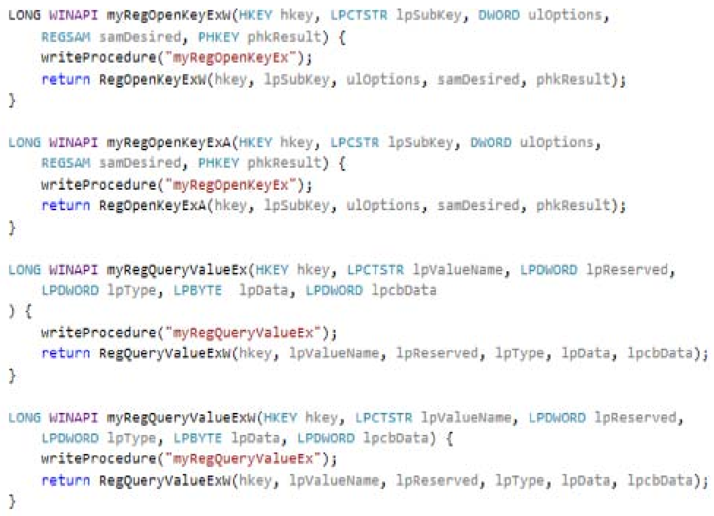
Figure 3 Malware Classifier Data Flow Diagram

A. Windows API Call Hooking

In windows system, API calls hooking is not as easy as in UNIX system where we can just use strace command to hook all system call that is being called by a process and all its child. First, we have to know that in the Windows API call are categorized as a dynamic link library (.DLL) files. We can have two different types of DLL files, native and extended. Native DLL means DLL that are directly call machine codes in Windows while extended DLL call native DLL. The API calls that we want to hook are the native one.

We use a framework name an easyhook framework to done the job in this paper [4]. Easyhook is available in two languages, VC++ and C#. We are using a VC++ framework in this experiment since we don't have to translate the variable and procedure type that we are going to hook. If we are using C# we have to translate the procedure type and variable type to type that is exist in C# language.

We create two files for hooking procedure. The first file defines all API calls that we want to hook and the second one define the injection process to an executable target. The first file will be compiled as a .DLL file and the second one (injector) will be compiled as a command line program. The snippet of the first file can be seen in **Error! Reference source not found.** and the second one in Table 1.



```
LONG WINAPI myRegOpenKeyExW(HKEY hkey, LPCTSTR lpSubKey, DWORD ulOptions,
    REGSAM samDesired, PHKEY phkResult) {
    writeProcedure("myRegOpenKeyEx");
    return RegOpenKeyExW(hkey, lpSubKey, ulOptions, samDesired, phkResult);
}

LONG WINAPI myRegOpenKeyExA(HKEY hkey, LPCSTR lpSubKey, DWORD ulOptions,
    REGSAM samDesired, PHKEY phkResult) {
    writeProcedure("myRegOpenKeyEx");
    return RegOpenKeyExA(hkey, lpSubKey, ulOptions, samDesired, phkResult);
}

LONG WINAPI myRegQueryValueEx(HKEY hkey, LPCTSTR lpValueName, LPDWORD lpReserved,
    LPDWORD lpType, LPBYTE  lpData, LPDWORD lpcbData
) {
    writeProcedure("myRegQueryValueEx");
    return RegQueryValueExW(hkey, lpValueName, lpReserved, lpType, lpData, lpcbData);
}

LONG WINAPI myRegQueryValueExW(HKEY hkey, LPCTSTR lpValueName, LPDWORD lpReserved,
    LPDWORD lpType, LPBYTE lpData, LPDWORD lpcbData) {
    writeProcedure("myRegQueryValueEx");
    return RegQueryValueExW(hkey, lpValueName, lpReserved, lpType, lpData, lpcbData);
}
```

Figure 4 Hooked API

Table 1 Easyhook Injector Function

```
NTSTATUS nt = RhCreateAndInject(
argv[2],NULL,  NULL, EASYHOOK_INJECT_DEFAULT,
dllToInject, NULL, NULL, NULL, pulong);
```

B. Dataset Analysis

Dataset that is used for building model can be downloaded here https://www.sec.cs.tu-bs.de/data/malheur/. This dataset consists of malware specific to family category. Information about the dataset can be seen in Table 2 below.

Table 2 Dataset Information

Family Name	Class Name	Total Malware
ADULTBROWSER	DIALER/TROJAN	262
ALLAPLE	WORM	300
BANCOS	TROJAN	48
CASINO	TROJAN/ADWARE	140
DORFDO	TROJANDOWNLOADER	65
EJIK	TROJAN	168
FLYSTUDIO	TROJAN/WORM	33
LDPINCH	TROJAN/BACKDOOR	43
LOOPER	TROJAN	209
MAGICCASINO	ADWARE	174
PODNUHA	TROJAN/ROOTKIT	300
POISON	BACKDOOR/TROJAN	26
PORNDIALER	DIALER/TROJAN	97
RBOT	VIRUS/VIRUT	101
ROTATOR	ADWARE	300
SALITY	VIRUS	84
SPYGAMES	VIRUS/VIRUT	139
SWIZZOR	TROJAN	78
VAPSUP	TROJAN/ADWARE	45
VIKING_DLL	WORM	158
VIKING_DZ	WORM	68
VIRUT	VIRUS	202
WOIKOINER	TROJAN	50
ZHELATIN	WORM/TROJAN	41

We are going to create two malware classes from available dataset. The classes are worm and trojan. Worm class is built using viking_dll, flystudio, viking_dz, and zhelatin family. While trojan class is buit using bancos, flystudio, looper, podnuha, and swizzor family. The information that we are going to extract from the dataset is the API call sequence for each individual malware from each classes. We just take 15 API call types that are the most widely used by allaple and ejik family as a representation of worm and trojan class respectively, since HMMER doesn't support protein code more than 20 codes. The most widely used API calls for allaple and ejik family can be seen at Table 3. Then to fulfill 15 system calls we have to find some other system calls that are specific to both classes. Information about the behavior of allaple and ejik are retrieved from public and private malware analysis community such as Microsoft msdn web page. We conclude the complete 15 system calls that are being used for this analysis are in Table 4.

C. HMMER Tools

HMMER is a tool commonly used by biologist to compare DNA sequence between individuals to get the common functionality between these individuals. HMMER can build a model for individual belong to the same class given their sequence of DNA. In this malware analysis case, we are going to substitute DNA with system call sequence to build the model for each malware class. HMMER ask a stockholm file format (.sto) as an input to build a model and fasta file format (.fa) to query a model.

Table 3 Top 8 System call used by Allaple and Ejik Family

RANK	System call
1	VirtualProtect
2	FindFile
3	RegOpenKeyEx
4	RegQueryValueEx
5	OpenFile
6	RegSetValueEx
7	RegCreatKeyEx
8	GetSystemDirectory

Table 4 Complete Systemcacall

RANK	System call
1	VirtualProtect
2	FindFile
3	RegOpenKeyEx
4	RegQueryValueEx
5	OpenFile
6	RegSetValueEx
7	RegCreatKeyEx
8	GetSystemDirectory
9	Ping
10	SetFileAttributes
11	CreateSocket
12	ConnectSocket
13	EnumWindow
14	CreateWindow
15	SetWindowsHook

We build a model by giving some sequences of trojan class and worm class in stockholm format. Then, HMMER will create a database with .hmm format for these two classes separately in a text file. We can then query to each model (.hmm) using a fasta file for a malware with unknown class. We will then take a model with greater score as the class for the analyzed malware. If the score in both classes doesn't pass the default threshold, then the program will be classified as a benign program.

IV. EXPERIMENT

The experiment is conducted by building a model for each malware class, in this experiment model is created for worm and trojan class. We don't build a model for benign class, so we are going to determine an instance as benign by looking at its threshold regarding to classes that are created. We classify an unknown instance belongs to a class if it has the highest score in that class and will be classified as benign if it doesn't pass threshold at any class.

Models will be tested to know how good sequence of system call and profile hidden markov model for malware classification. Aspects that we are going to measure for each model are accuracy and false negative rate. The experiment is conducted under these environment circumstances. These environments are set up within a virtualbox version 5.1.26 running on windows 10 host.

- OS: Windows 7 32 bit
- RAM: 4GB
- Processor: Intel(R) i7-6700HQ CPU @2.60GHz

As described earlier malware classes are built from malwares from different families. We take 10 malware instances from each family to build malware classes related to the family. We test 53 worms, 50 trojans, and 49 benign executable files to these two malware class models that have been created. Results are shown in Table 5 below.

Table 5 Multiclass Malware Classifier Confusion Matrix

	Worm(53 data)	Trojan(50 data)	Benign (49 data)
Worm Detected	51	1	6
Trojan Detected	0	49	14
Benign Detected	2	0	34

Based on data in the confusion matrix above we can calculate accuracy for each malware class and false negative for benign class. Results are shown below.

V. RELATED WORKS

Early work on sequence of system call analysis is proposed by Forrest in her paper titled A Sense of Self for Unix Processes [5]. In her paper Forrest try to build an anomaly detection system for UNIX application program name sendmail. She tried to capture sequences of system calls of sendmail under normal and abnormal/attacked circumstances. She built a database for the normal one and compared some unknown sequences of sendmail to the normal database. She created a threshold on which a sequence will be categorized as abnormal if it deviates threshold parameter. She concluded that her work has a high accuracy for detecting abnormal event.

Another recent work on sequence of system call is proposed by Canzanese in his paper titled Detection and Classification of Malicious Processes Using System Call Analysis [6], which is published on May 2015. Canzanese work his project on windows operating system which is well known for its vulnerability to many malwares. He focuses more on efficient online detection system using system call rather on algorithm to detect anomalies. He used a default Event Tracing for Windows (ETW) system to get a list of kernel system calls that is made by a process. In this paper, we are going to use user space system calls instead of kernel system calls.

Our work is basically only comparable with Canzanese's work, since Forrest's work is anomaly in the unix process instead of malware analysis. Canzanese work precision is very good; for worm class is around 89-100%, depending on the family of the worm with more than 2000 worm instances in his dataset. While trojan class accuracy is around 89-100% for most trojan families. In our work, precision for worm is about 96% and trojan is about 98%. Therefore, comparing our work to Canzanese work we have the same quality in precision.

VI. CONCLUSION & FURTHER STUDIES

Based on testing results, here are some conclusions that we can get from the experiment we have conducted.

1. Sequence of system call is reliable as a fingerprint to detect malicious software.
2. Profile hidden markov model is a reliable method for classifying malware instances based on their system call sequences. False negative rate is pretty high since we don't build a class for benign instances.
3. User space windows API can be used for malware classification instead of kernel space API.

Suggestions for further studies in aspect of enhancing accuracy and performance are

1. Use more system call type and find out how many system call is suitable for highest accuracy but minimal number of system call.
2. Expand malware classes and look for most common system call and sequence of system call from those malware classes
3. Using k-mers to determine selection of system call.

VII. REFERENCES

[1] "Defining Malware: FAQ". *technet.microsoft.com*. Retrieved 10 September 2009.

[2] https://en.wikipedia.org/wiki/Obfuscation_(software),accessed September 19, 2017.

[3] https://blog.malwarebytes.com/threat-analysis/2013/03/obfuscation-malwares-best-friend/, accessed September 19, 2017.

[4] https://easyhook.github.io, accessed September 19, 2017.

[5] S. Forrest et al, "A sense of Self for Unix Processes," Proc. 1996 IEEE Symp. Security and Privacy, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 120-128.

[6] Raymond J. Canzanese, Detection and Classification of Malicious Process Using System Call Analysis. Doctor Thesis, Drexel University, Drexel, 143 p.