Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.5, No.12, 2014
www.iiste.org

Static Analysis Based Behavioral API for Malware Detection using Markov Chain

Proff. Abbas M. Al-Bakri[1] Hussein L. Hussein[2]
1.Kuffa university/Faculty collage of computer and mathematics
2.Baghdad university/Ibn Al-Haitham collage
* E-mail of the corresponding author: abbasmoh67@yahoo.com , husseinlefta@yahoo.com

The research is financed by Asian Development Bank. No. 2006-A171(Sponsoring information)

Abstract
Researchers employ behavior based malware detection models that depend on API tracking and analyzing features to identify suspected PE applications. Those malware behavior models become more efficient than the signature based malware detection systems for detecting unknown malwares. This is because a simple polymorphic or metamorphic malware can defeat signature based detection systems easily. The growing number of computer malwares and the detection of malware have been the concern for security researchers for a large period of time. The use of logic formulae to model the malware behaviors is one of the most encouraging recent developments in malware research, which provides alternatives to classic virus detection methods. To address the limitation of traditional AVs, we proposed a virus detection system based on extracting Application Program Interface (API) calls from virus behaviors. The proposed research uses static analysis of behavior-based detection mechanism without executing of software to detect viruses at user mod by using Markov Chain.
Keywords: Malware Detection; Markov Chain; Virus Behavior; API Calls

1. Introduction

Since the appearance of the first computer virus in 1986, and the A significant number of new viruses have appeared every year. The damage caused by malicious code has dramatically increased in the past few years. Software security assurance and malware detection are important topics of information security [13].

Malware analysis can be categorized into two main categories [11]:

- Analysis of the infected file without executing it, which is known as static analysis. In this approach, we extract low-level information such as Control Flow Graphs (CFGs), Data-Flow Graphs (DFGs) and System call analysis. This information can be gathered by disassembling or decompiling the infected file using tools like IDA Pro. Sometimes analyzing the infected file in a different environment to avoid auto execution of the malware is better. Using static analysis we get fast, safe and low false positives and we trace all paths, which helps in terms of getting a lot of information to analyze. On the other hand static analysis may fail in analyzing unknown malware that uses code obfuscation techniques [3].
- Analysis of the infected file during its execution, which is known as dynamic analysis. Dynamic analysis executes the infected file on simulated environment (a debugger or a virtual machine or an emulator) to analyze its malicious functions. The analysis environment must be invisible to the malware because the malware writer use tools like anti-virtual machine and Anti-emulation to hide their malware functions if they detect they are under analysis. Dynamic analysis fails to detect activities of interest if the target changes its behavior depending on trigger conditions such as existence of a specific file or specific day as only a single execution path may be examined for each attempt.

Techniques: There are mainly two techniques for malware detection:

- Signature-Based and Behavior-Based techniques as shown in (Table 1-2). In signature-based techniques a sequence of instructions unique to a malware is used to generate a malware signature, which is captured by researchers in a laboratory environment.

A signature should be able to identify any malware exhibiting the malicious behavior specified by the signature. Most of antivirus scanners are signature based.

- Behavior-based detection techniques focus on analyzing the behavior of known and suspected malicious code. Such behaviors include factors such as the source and destination addresses of the malware, the attachment types in which they are embedded and statistical anomalies in malware infected systems. One example of a behavior-based detection approach is the histogram-based malicious code detection technology patented by Symantec.

To overcome the limitations of signature-based detection some malware researchers apply graph ,**C**ontrol **F**low **G**raph,(**CFG**), Call Graph , Machine Learning techniques and Data Mining

techniques ,**O**bjective-**O**riented **A**ssociation, (OOA) .Other researchers apply techniques like Finite Automaton, HMM, Naïve Bayes, Support Vector Machine (SVM) and Decision tree and neural network to improve Behavior-Based Detection[3].

2. Background.

It is very important to understand application program interfaces (APIs) and their features, in order to trace the behavior of programs and to understand hidden features of malicious codes.

Therefore, an outline of API calls is provided here in order to enhance understanding of this important system service [12].

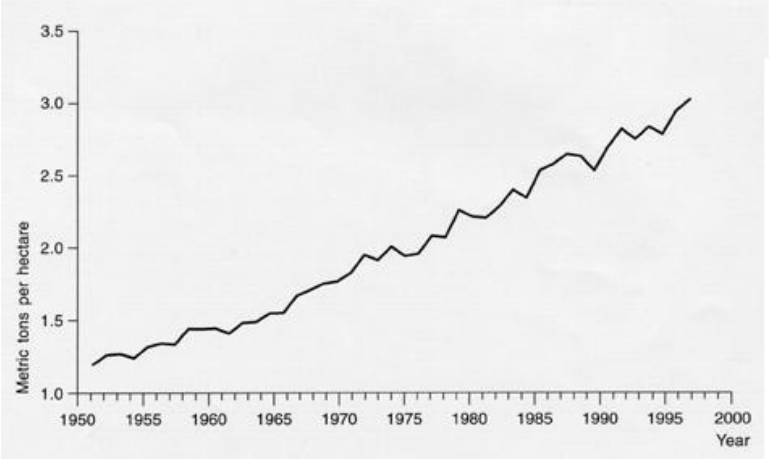

Figure 1. The Trend of Economic Development
Description for the above figure.

2.1. Windows Application Program Interface (API) system calls.

In 1995, Microsoft released Windows 95 and at the same time introduced a set of system calls known as Win32 API, which represented a 32-bit application program interface. The new APIs had the advantage of higher system speeds because they provided a set of optimized system operations. User applications in the Windows operating system (OS) based on these API function calls are stored in **D**ynamic **L**ink **L**ibraries (**DLL**s) such as User32.ddl, Kernel32.dll, Advqpi.dll, and Gui32.dll, in order to gain access to system resources involving registry and network information, processes and files. Each Win32 API call has its own memory address place in the **I**mport **A**ddress **T**able (**IAT**) which every process in the system has and which each process will consult when it makes an API call. A Win32 API call is normally called from a process running at the user level, and then the called API will be handled by the system and converted to its equivalent function, known as a native API call, which will be understood by the kernel of the OS. A service in the kernel will handle the requested operation and its outcome will return to the original user application that made the call.

The majority of systems services run in the kernel and need privileges in order to access it. Native API calls, which can be directly called by any process at the kernel level, are dealt with in the dynamic link library (ntdll.dll) in order to have the kernel provide the requester service. The complete list of kernel mode functions is stored by memory location addresses in the **S**ystem **S**ervice **D**ispatch **T**able (**SSDT**), which is accessed each time a native API routine is called. The parameters are then passed to the memory location and the function continues with its execution.

Windows API calls play an important role in exploiting the power of Windows, allowing virus writers to use API calls to gain more security privileges and perform malicious actions. Windows APIs issue calls to perform several actions, such as user interfaces, system services and network connections, which can be utilized for good or evil.

Because API calls will give a full and complete description of a particular program, the analysis of its API calls will lead directly to the understanding of its behavior [12].

2.2. Polymorphic and Metamorphic malware

Software security assurance and malware (Trojans, worms, and viruses, etc.) detection are important topics of information security. Software obfuscation, a general technique that is useful for protecting software from reverse engineering, can also be used by hackers to circumvent the malware detection tools. Current static malware detection techniques have serious limitations, and sandbox testing also fails to provide a complete solution due to time constraint [1].

A classification of malware based on payload, enabling vulnerability, and propagation mechanism gives three generations [1].

First generation

- Malware that shares the properties of a virus
- Requires human action to trigger replication and spreading
- Propagates via email and file sharing
- Examples: Melissa, LoveLetter.

Second generation

- Malware that shares the properties of a worm
- Does not require human intervention for replication and spreading.
- Automatic scanning of victims for vulnerabilities
- Hybrid in nature, blended with viruses and Trojans
- Propagates via Internet
- Examples: Slapper worm, SQL Slammer worm, and Blaster worm.

Third generation

- Pre-compiled vulnerable targets
- Exploits known and unknown vulnerabilities
- Employs multiple attack vectors
- Geographical region- or organization-specific Malware.
- Attacks security technologies and products.

Existing AntiVirus (AV) products provide detection techniques which are based on signatures that have been collected from previous seen viruses and then added to an AV database. Prior to the arriving of a virus to the system, its signature will be compared with those stored in the database and if there is a match, the virus will be detected; otherwise, the system will run normally. Thus, zero day viruses will not be detected by traditional detection systems unless this new virus is received by the antivirus database. Signature-based detection systems need databases in order to store the signatures. As the number of viruses increases every day, ever larger databases are needed to store all their signatures, so that more storage space will be needed in the near future. The large database will also affect the speed of searching for signatures, and, thus, affect the performance of the system. These disadvantages mean that the signature-based detection techniques will soon be inadequate to protect computer systems [12].

There is a growing need for behavioral specification to be used in detecting attacks, providing a robust and manageable detection technique. The present research proposes to build a detection technique using temporal logic specifications that have been inferred from the analysis of Windows and native API calls that represent virus behaviors [1].

3. Related work

Skormin et al [2] have designed an approach that intercepts API calls while a program is running. They detect any attempt of a malware to self-replicate at run-time. Their methodology was to trace the behavior of normal processes and analyses API calls along with their input, output arguments and the execution results. The replication of a process was modelled by the Gene of Self Replication (GSR) based upon building blocks. Each block in the GSR is considered as a portion of the self-replication process which includes seeking for files and directories, writing to files, reading from files, and closing and opening a file. This approach has detected several viruses from different classes but on the other hand, they used to hook native API calls only in the kernel. As said by [9,7] native APIs are not fully documented that gives some viruses the ability to use some of these undocumented API to attack the system.

Alazab et al [6] have used a static analysis to track API calls using existing tools. They analyze malware to classify program executable as normal or malicious. They have used the IDA Pro [5] with their own Python program to automatically extract API calls. They had examined six groups of virus steps such as search, copy, delete, read and write. They have found that read and write files were the most API calls used by malwares to infect the program. Lists of Win32 API calls have been extracted at the user level. However, there are some viruses that might not be detected by [6] because they directly call the kernel by using native API calls as mentioned by [4,9].

Veeramani and Rai [8] have used a statistical analysis for Windows API calls to describe the behavior of programs. They used an automated framework for analyzing and categorizing executables rely on their relevant API calls. They try to increase the detection rate by using Document Class wise Frequency feature Selection (DCFS) measure by getting the information related to malware from the extracted API calls.

4. Stochastic process in Markov Chain

A stochastic process is a mathematical model that evolves over time in a probabilistic manner .The special kind of stochastic process, called *Markov chain*, where the outcome of an experiment depends only on the outcome of the previous experiment. In other words, the next state of the system is depending only on the present state, not on preceding state [10].

4.1. Transition Matrix

The entries in the first row of the matrix P in table (3) represent the probabilities for the various kinds of virus steps behaviors probability. Similarly, the entries in the other rows represent the probabilities for the various kinds of virus steps behaviors, respectively. Such a square array is called the matrix of **transition probabilities**, or the transition matrix.We consider the question of determining the probability that, given the chain is in state i behavior, it will be in state j from now. We denote this probability by p_{ij} [10].

5. Experimental results

The proposed malware detection system is evaluated using the Malware Detection System (MDS). Accuracy is defined as the product of number of malicious process rightly classified as malicious transitions table (3).

$P(\text{malware}) = P_{12}*P_{23}*P_{34}*P_{45.}$ If $P(\text{malware}) > 0$ Malware

Else Benign

Where, P (malware) is the Probability of malware.

Windows executable files which are recognized as benign and malicious executables are collected from vxheavens.com for malware and download.com for benign. The malicious executables mainly consists of backdoors, worms, and Trojan horses and the benign executables are gathered from [10]. The proposed algorithms are implemented using Mat lab.

5. Research Motivation

In this paper, we address the problem of static slicing on binary executables for the purposes of the malicious code detection in user API. Static slicing is useful to extract those code fragments that are critical from the security standpoint. The flowchart of research motivation steps shown in figure 1.

Once extracted API by using APIMonitor software, this software will extract API functions and store them in Excel sheet. Then coded these APIs by using API Data Base to convert these APIs to codes, the code ranged from 1 to 3044.

After that the research will extract APIs code that mapping the codes of virus behavior API that matching the five steps of virus detection behaviors [4], as shown in table (4).This procedure one after coded the APIs of the software want to be examined and the APIs in table (4) will be coded and converted to integer values ranged from 0 to 5 as shown in table (5) .

5.1. Extracting API calls.

Viruses are just like normal programs and can be distinguished by tracking their API calls that lead to malicious actions. Therefore, this research concentrates on tracing API calls in order to understand virus behavior. More than one tool was utilized to trace API calls in static and runtime environments. Most researchers rely on just one tool, which runs either statically or dynamically, but this research uses static analysis.

Such a system is able to examine API calls issued in the user mode and if a file is detected as a virus, no further examination is needed. If, however, it is not considered to be a virus, the detection system will examine it at the kernel level by observing its native API calls. In this research we extract API from the application which we want to examine it by using API monitor which extract the API from the application without need for execute.

5.2. Convert API to its code.

We create an Excel database sheet to collect all APIs windows Microsoft .This Excel database sheet contain all API sorted in alphabetic order and each API has its own index, which it's unique and not shared with any other API windows Microsoft.

In this step we convert each API in the examined software by matching API with each API in the database sheet and extract its index when found , this index number will be in the range from 1 to 3041 according to the database sheet indexes.

5.3. Extract API features.

In this step we use "Virus category" in table (3) which explains the five steps to generate virus.

In these steps we use the APIs codes generates by previous step and covert its APIs to APIs code , as

shown in API Function Calls code fie field , then covert these codes to another values from 0 to 5 as explained ,Step Code, in table (3) as follows:

0 : For each API not found in virus behaviors.
1 : For each API found in First step (Find to infect) virus behaviors.
2 : For each API found in Second step (Get information) virus behaviors.
3 : For each API found in Third step (Read and/or copy) virus behaviors.
4 : For each API found in Fourth step (Write and/or delete) virus behaviors.
5 : For each API found in Fifth step (Set information) virus behaviors.

5.4. Markov chain

The Markov chain is used when we need to evaluate the probability of action occurs depending on the previous action only. This idea is used in this research to evaluate of probability for malware occurs according to the graph translating which we designed and constructed for the transition graph and table of probability of six steps for transition graph and 6*6 matrix for probability transition table(3).For the experiment we use application which has (942) API and after the coding and transformation we calculate the following transition probability "Step Code " in table (5),and transition graph figure 2.To find if the tested application is malware or benign, we can use this system to extract the above transition matrix for each application, and by examining the items $P_{12,}P_{23},P_{34},P_{45,}$ explained the behavioral virus analysis including our API extraction mechanism. The transition in reverse order is not important then it's ignored in transition graph representation.

6. Conclusions

The problem of accurate discrimination between benign and malware (API) function calls have been explained in this paper. This problem challenges the most malware behavior detection models that depend on (API) tracing and analyzing. A Markov chain (i.e. 5-grams) or longer is used to model the API call. This composite feature set is provided as an input to the malware detection system to raise the final alarm. Association mining based classification is used because it yields higher detection accuracy than previous data mining based detection systems.

The number of APIs is reduced, by removing the redundant APIs not used as virus behavior, to make the malware analysis efficient.

The performance of the proposed detection system is evaluated for accuracy of the malware detection system and compared with the existing data mining based detection systems. It is inferred that the proposed malware detection system outperforms the existing malware detection systems.

Table 1. Summary of the advantages and disadvantage of static and dynamic analysis

Malware analysis	Advantage	Disadvantage
Static analysis	-Fast and safe. -Low level of false positives. -Good in analyzing multipath malware.	Difficulty analyzing unknown malware.
Dynamic analysis	Good in detecting unknown malware.	-Neither fast nor safe. -Difficulty analyzing multipath malware.

Table 2. Summary of the advantages and disadvantages of signature based and behavior-based techniques.

Malware based	Advantage	Disadvantage
Signature-based	-Less scanning time. -Few false positives.	-Unknown malware can easily evade detection. -Cannot deal with simple obfuscation.
Behavior-based	Best results in detecting of polymorphic malware.	Not able to detect a lot of polymorphic viruses present (Packers).

Table 3. Transition Matrix Probability

steps	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0.9251	0.0081	0.0142	0.0445	0.0040	0.0040
P_1	0	0.8750	0.1250	0	0	0
P_2	0.2128	0	0.3830	0.3936	0.0106	0
P_3	0.0769	0	0.4327	0.4231	0.0673	0
P_4	0.0205	0	0	0.0051	0.9436	0.0308
P_5	0.2273	0	0.0909	0	0.0455	0.6364

Table 4. Five steps of virus behavior

Step	**Virus category**	**API Function Calls**
First	Find to infect	FindFirstStream,FindFirstFileTransacted,FindFirstStreamTransactedW,FindFirstStreamW,FindClose,FindNextFile,FindFirstName,FindFirstFileEx,FindFirstFile,FindFirstNameW,FindNextFileName,FindNextFileNameW,FindFirstNameTransactedW,FindNextStreamW,FindNextStream
Second	Get information	GetFileAttributesEx,GetFileAttributesTransacted,GetFileAttributes,GetFileInformationByHandle,GetFileBandwidthReservation,GetCompressedFileSizeTransacted,GetFileInformationByHandleEx,GetCompressedFileSize,GetBinaryType,GetFileSizeEx,GetFileSize,GetFileType,GetTempFileName,GetTemppath,GetFinalPathNameByHanle,GetLongPathNameTransacted,GetFullPthNameTransacted,GetFullPthName,GetLongPathName,GetShortPathName.
Third	Read and/or copy	ReadFile,ReadFileW,OpenFile,OpenFileByld,ReopenFile,CreateHardLinkTransacted,CreateHardLink,CreateSymbolicLinkTransacted,CreateSymbolicLink,CopyFileEx,CopyFile,CreateFileW,CreateFile,CopyFileTransacted , CreateFileTransacted.
Fourth	Write and/or delete	ReplaceFile,WriteFile,DeleteFileTransacted,DeleteFileW, DeleteFile, CloseHandle
fifth	Set information	SetFileInformationByHandle,SetFileValidData,SetFileBandwidthReservation,SetFileShortName,SetFileAttributesTransacted,SetFileApisToOEM,SetFileAttributes,SetFileApisToANSI.

Table 5. Five steps codes of virus behaviors

Step Code	Virus category
1	For each API found in First step (Find to infect) virus behaviors.
2	For each API found in Second step (Get information) virus behaviors.
3	For each API found in Third step (Read and/or copy) virus behaviors.
4	For each API found in Fourth step (Write and/or delete) virus behaviors.
5	For each API found in Fifth step (Set information) virus behaviors

Table 6. Five steps of virus behavior codes

Step	Step code	Virus category	API Function Calls code
other	0	None of the virus steps	0
First	1	Find to infect	732,738,740,745,747,748,749,751,752,753,761, 764,764,767,768
Second	2	Get information	847,898,900,1031,1033,1036,1039,1041,1042,1043,1044,1046,1050,1057,1059,1104,1106,1273,1324, 1327
Third	3	Read and/or copy	236,238,241,293,300,301,306,308,370,371,1931,932, 2135,2138,2289
Fourth	4	Write and/or delete	181 , 452 ,454 , 455,2290 , 3005
fifth	5	Set information	2465,2466,2467,2469,2471,2473 , 2477 , 2479

Table 7 transition matrix probability

steps	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0.9251	0.0081	0.0142	0.0445	0.0040	0.0040
P_1	0	0.8750	0.1250	0	0	0
P_2	0.2128	0	0.3830	0.3936	0.0106	0
P_3	0.0769	0	0.4327	0.4231	0.0673	0
P_4	0.0205	0	0	0.0051	0.9436	0.0308
P_5	0.2273	0	0.0909	0	0.0455	0.6364

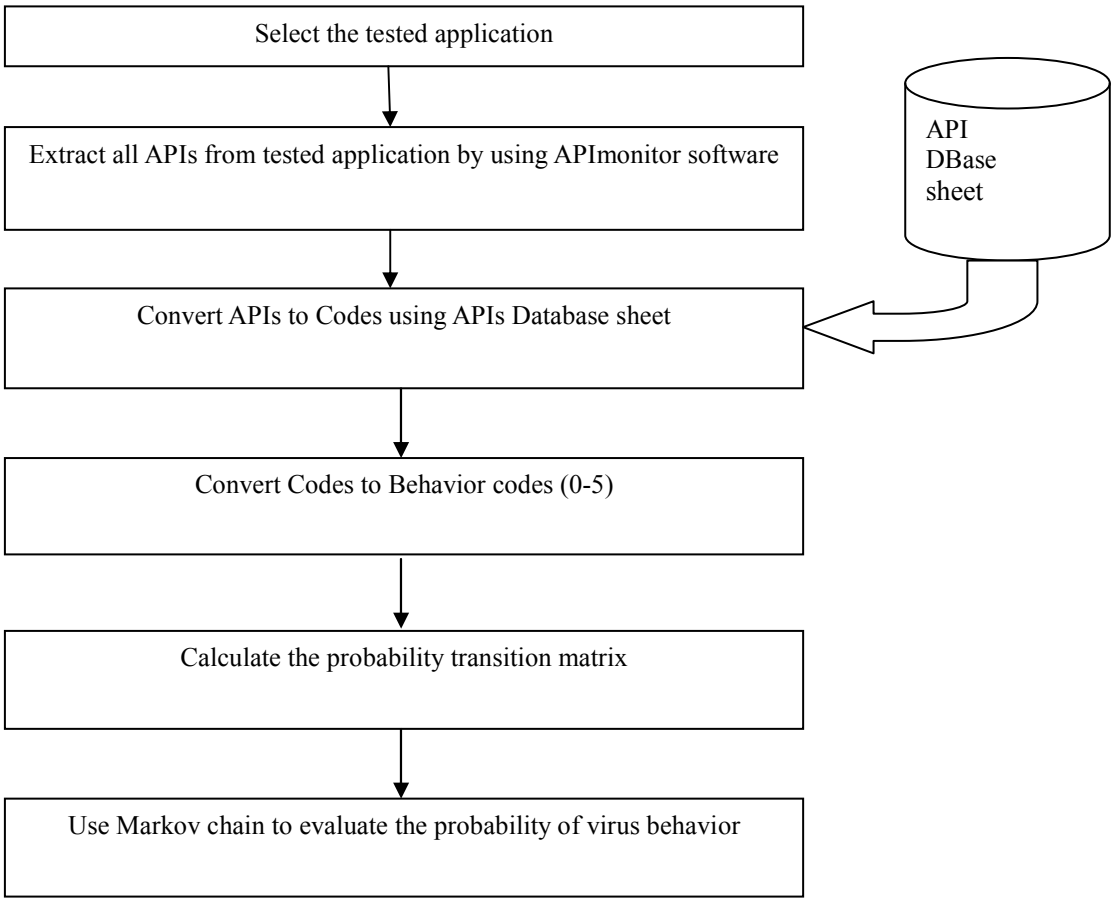

Figure 1. Research motivation flowchart

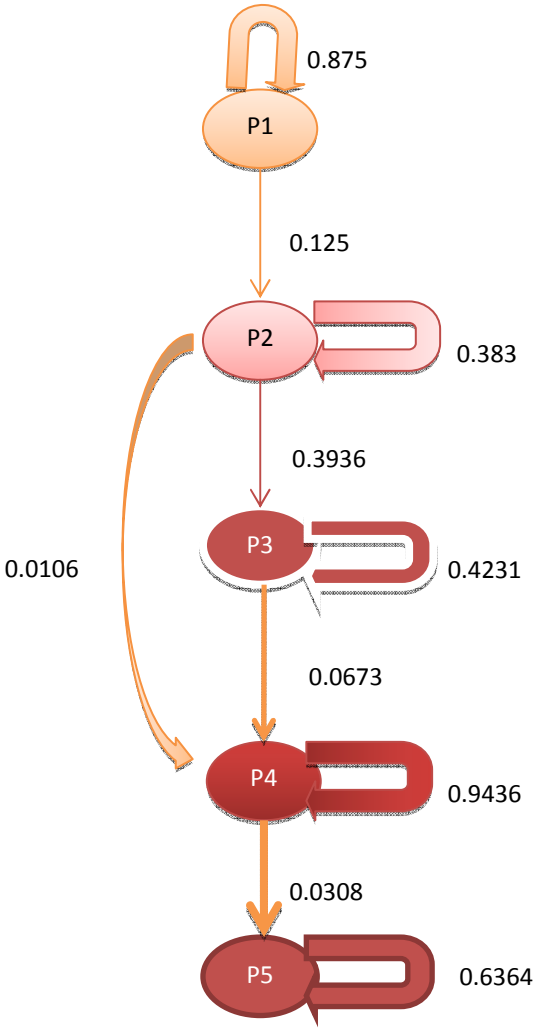

Figure. 2. Transition graph

References

A. H. Sung, J. Xu, *P. Chavez, *S. Mukkamala , 2004 ,"Static Analyzer of Vicious Executables (SAVE)",IEEE ,computer security applications conference,.

A. Skormin, D. Volynkin, I. Summerville, and J. Moronski, 2007.,"Run-Time Detection of Malicious Self-Replication in Binary Executables",in Journal of Computer Security,vol.15, no.3, pp. 273-01.

Ammar Ahmed E. Elhadi, Mohd Aizaini Maarof and Ahmed Hamza Osman , 2012, "Malware DetectionBased on Hybrid Signature Behavior Application Programming Interface Call Graph ", American Journal of Applied Sciences 9 (3): 283-288.

G.Hoglund and J.Butler.Rootkits,2005,"subverting the Windows Kernel". Addison Wesley Professional.

IDA Pro Dissasember .DataRescue, 2011 ,"An Advanced Interactive Multi-processor Disassembler," http://www.datarescue.com.

M. Alazab, S. Venkataraman, and P. Watters, 2010, "Towards understanding malware behavior by the extraction of API calls," IEEE 2nd Cybercrime and Trustworthy Computing Workshop , pp. 52-59.

M. Russinovich, , 2005, "Inside the Native API," http://www.sysinternals.com/Information/NativeApi.html.

R. Veeramani, and N. Rai, 2012 ,"Windows API based Malware Detection and Framework Analysis," in International Journal of Scientific & Engineering Research, vol. III, no. III.

R. Vieler. Professional Rootkits. Wrox Press, 2007.

Ronald E. Walpole , Raymond H. Myers,Sharon L. Myers ,Keying Ye,2007,"Probability & Statistics for Engineers &; Scientists" , Pearson Education International.

Saman Mirza Abdulla1, Assoc. Prof. Dr. Miss Laiha Mat Kiah, and Assoc. Prof. Dr. Omar Zakaria, 2012," Minimizing Errors in Identifying Malicious API to Detect PE Malwares Using Artificial Costimulation", International Conference on Emerging Trends in Computer and Electronics Engineering (ICETCEE'2012), March 24-2.

Sulaiman Al amro (1,2), Antonio Cau , *2012,* "Behavioural API based Virus Analysis and Detection" , *(IJCSIS) International Journal of Computer Science and Information Security,Vol. 10, No. 5*

Zhang Yichi, Pang Jianmin, Bai Lili, Fu Wen , ,2009,"Static Analysis of Malware to Detect Exception Return", International Forum on Information Technology and Applications.

The IISTE is a pioneer in the Open-Access hosting service and academic event management. The aim of the firm is Accelerating Global Knowledge Sharing.

More information about the firm can be found on the homepage:
http://www.iiste.org

CALL FOR JOURNAL PAPERS

There are more than 30 peer-reviewed academic journals hosted under the hosting platform.

Prospective authors of journals can find the submission instruction on the following page: http://www.iiste.org/journals/ All the journals articles are available online to the readers all over the world without financial, legal, or technical barriers other than those inseparable from gaining access to the internet itself. Paper version of the journals is also available upon request of readers and authors.

MORE RESOURCES

Book publication information: http://www.iiste.org/book/

Academic conference: http://www.iiste.org/conference/upcoming-conferences-call-for-paper/

IISTE Knowledge Sharing Partners

EBSCO, Index Copernicus, Ulrich's Periodicals Directory, JournalTOCS, PKP Open Archives Harvester, Bielefeld Academic Search Engine, Elektronische Zeitschriftenbibliothek EZB, Open J-Gate, OCLC WorldCat, Universe Digtial Library , NewJour, Google Scholar