Survey on Malware Detection Methods

Vinod P.
Department of Computer Engineering,
Malaviya National Institute of Technology,
Jaipur, Rajasthan
e-mail: vinod_p22@yahoo.com

V.Laxmi,M.S.Gaur
Department of Computer Engineering,
Malaviya National Institute of Technology,
Jaipur, Rajasthan
e-mail: {vlaxmi|gaurms}@mnit.ac.in

***Abstract*— Malwares are malignant software's .It is designed to damage computer systems without the knowledge of the owner using the system. Software's from reputable vendors also contain malicious code that affects the system or leaks information's to remote servers.Malware's includes computer viruses, spyware, dihonest ad-ware,rootkits,Trojans,dialers etc. The paper focuses on various Malware detection methods like signature based detection, reverse engineering of obfuscated code, to detect malicious nature.**

***Keywords*—** Malware, obfuscation, Malware normalizer, reverse engineering

I. INTRODUCTION

Malware is a collective term for any malicious software which enters system without authorization of user of the system. The term is created from merging the words 'malicious' and 'software'. Malware is a very big threat in today's computing world. It continues to grow in volume and evolve in complexity. As more and more organizations try to address the problem, the number of websites distributing the malware is increasing at an alarming rate and is getting out of control. Most of the malware enters the system while downloading files over Internet. Once the malicious software finds its way into the system, it scans for vulnerabilities of operating system and perform unintended actions on the system finally slowing down the performance of the system.

Malware has ability to infect other executable code, data/system files, boot partitions of drives, and create excessive traffic on network leading to denial of service. When user executes the infected file; it becomes resident in memory and infect any other file executed afterwards. If operating system has a vulnerability, malware can also take control of system and infect other systems on network. Such malicious programs (virus is more popular term) are also known as parasites and adversely affect the performance of machine generally resulting in slow-down.

Some malware are very easy to detect and remove through antivirus software. These antivirus software maintains a repository of virus signatures i.e., binary pattern characteristic of malicious code. Files suspected to be infected are checked for presence of any virus signatures. This method of detection worked well until the malware writer started writing polymorphic and metamorphic malware. These variant of malware avoid detection through use of encryption techniques to thwart signature based detection.

Security products such as virus scanners look for characteristics byte sequence (signature) to identify malicious code. The quality of the detector is determined by the techniques employed for detection. A good malware detection technique must be able to identify malicious code that is hidden or embedded in the original program and should have some capability for detection of yet unknown malware. Commercial virus scanners have very low resilience to new attacks because malware writers continuously make use of new obfuscation methods so that the malware could evade detections.

This paper is organised as follows. Section II briefly describes various types of malware. Section III is a review of malware detector .Section IV reviews malware detection methods explains structural and functional aspects of polymorphic and metamorphic malware. Section V explains obfuscation and transformation techniques used by malware to avoid detection. Section VI explains malware similarity analysis method .Section VII reviews malware normalization with concluding remarks in Section VIII.

II. Malware Types

Malware can be broadly classified into following categories.

A. Viruses

Computer virus refers to a small program with harmful intent and has ability to replicate self. Mode of operation is through appending virus code to an executable file. When file is run, virus code gets executed. The original virus may evolve into new variants by modifying itself as in case of metamorphic viruses. A virus may spread from an infected computer to other through network or corrupted media such as floppy disks, USB drives. Viruses have targeted binary executable file (such as .COM and .EXE files in MSDOS , PE files in Windows etc.), boot records and/or partition table of floppy disks and hard disk, general purpose script files, documents that contains macros, registry entries in Windows, buffer overflow, format string etc.

B. Worms

Worms are self replicating programs. It uses network to send copies of itself to other systems invisibly without user authorization. Worms may cause harm to network by consuming the bandwidth. Unlike virus the worms do not need the support of any file. It might delete files, encrypt files in as crypto viral extortion attack or send junk email. Example Sasser, My Doom, Blaster, Melissa etc.

C. Spyware

Spyware is a collective term for software which monitors and gathers personal information about the user like the pages frequently visited, email address, credit card number, key pressed by user etc. It generally enters a system when free or trial software is downloaded.

D. Adware

Adware or advertising-supported software automatically plays, displays, or downloads advertisements to a computer after malicious software is installed or application is used. This piece of code is generally embedded into free software. The problem is, many developers abuse ad – supported software by monitoring Internet users' activities .The most common adware programs are free games, peer-to-peer clients like KaZaa, BearShare etc.

E. Trojans

Trojan horses emulate behavior of an authentic program such as login shell and hijacks user password to gain control of system remotely. Other malicious activities may include monitoring of system, damages system resources such as files or disk data, denies specific services.

F. Botnet

A botnet is remotely controlled software – collection of autonomous software robots. It is usually a zombie program (Worms, Trojans) under common control on public and private network infrastructure. Botnets are usually used to send spam /spyware remotely. Bot doesn't sit around on machine (infected machine) waiting for the instruction from a third party instead it looks for the communication with similar instances of bots awaiting instructions. Simplest bot configuration is where the bots are connected to single central hub. This configuration does not scale much because maintenance of various connections over single server is difficult. The next configuration is hierarchical structure where bot master connects to hundreds of bots which in turn is connected to many bots. Thus this configuration would scale much larger extent.

III.Malware Detector

A Malware detector 'D' is defined as a function whose domain and range are the set of executable program 'P' and the set {malicious, bening} [1].In other words malware detector can be defined as shown below.

$$D(p) = \begin{cases} \text{malicious} & \text{if p contains malicious code} \\ \text{benign} & \text{otherwise.} \end{cases}$$

The detector scans the program 'p' ε P to test whether a program is benign program or malicious program. The goal of testing is to find out false positive, false negative, hit ratio. The malware detector detects the malware based on signatures of malware.The binary pattern of the machine code of a particular virus is called as signature. Antivirus programs compare their database of virus signatures with the files on the hard disk and removable media (including the boot sectors of the disks) as well as within RAM. The antivirus vendor updates the signatures frequently and makes them available to customers via the Web.

a) False positive:

A false positive occurs when a virus scanner erroneously detects a 'virus' in a non-infected file. False positives result when the signature used to detect a particular virus is not unique to the virus - i.e. the same signature appears in legitimate, non-infected software.

b) False negative:

A false negative occurs when a virus scanner fails to detect a virus in an infected file. The antivirus scanner may fail to detect the virus because the virus is new and no signature is yet available, or it may fail to detect because of configuration settings or even faulty signatures.

c) Hit ratio:

A hit ratio occurs when a malware detector detects the malware.This happen because the signature of malware matches with the signatures stored in the signature databases.

IV. Malware Detection Techniques

Techniques used for malware detection can be broadly classified into two categories: anomaly-based detection and signature-based detection. An anomaly based detection techniques uses the knowledge of what is considered as normal to find out what actually is malicious .A special type of anomaly based detection is specification-based detection .Specification based detection makes use of certain rule set of what is considered as normal in order to decide the maliciousness of the program violating the predefined rule set. Thus programs violating the rule set are considered as malicious program. Signature based detection uses the knowledge of what is considered as malicious to fins out the maliciousness of the program under inspection.

IV (A) Signature-Based Malware Detection Techniques

Commercial antivirus scanners look for signatures which are typically a sequence of bytes within the malware code to declare that the program scanned is malicious in nature. Basically there are three kinds of malwares: basic, polymorphic, metamorphic malwares. In basic malware the program entry point is changed such that the control is transferred to malicious payload. Detection is relatively if the signature can be found for the viral code. Figure 1 show basic malware.

Fig.1 Basic kind of virus

Polymorphic viruses mutates while keeping the original code intact. A polymorphic malware consists of encrypted malicious code along with the decryption module. To enable the polymorphic virus the virus has got polymorphic engine somewhere in the virus body. The polymorphic engine generates new mutants each time it is executed. Signature based detection for such a virus is difficult because each variant new signature is generated which makes signatures based detection difficult. Strong static analysis based on API sequencing is used for polymorphic virus detection [9].Figure 2 shows polymorphic malware's.

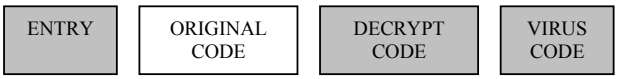

Fig.2 Polymorphic virus

Metamorphic malware can reprogram itself using certain obfuscation techniques so that the children never look like the parent [4]. Such malwares evade the detections from the malware detector since each new variant generated will have different signature, hence it is impossible to store the signatures of multiple variants of same malware sample. In order to thwart detection a metamorphic engine has to be implemented with some sort of disassembler in order to parse the input code. After disassembly, the engine will transform the program code and will produce new code that will retain its functionality and would still look different from the original code Figure 3 shows metamorphic malware and multiple signatures for multiple variants.

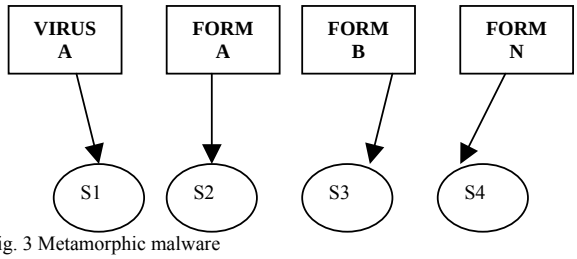

Fig. 3 Metamorphic malware

Let 'S' be the set of malware signatures. For the above figure - 3, $S_i \in S$ are signatures of metamorphic variant belonging to single metamorphic sample 'M'.

The main problems with the signature based detection method is as follows:

- Signature extraction and distribution is a complex task.
- The signature generation involves manual intervention and requires strict code analysis.
- The signatures can be easily bypassed as and when new signatures are created.
- The size of signature repository keeps on growing at an alarming rate.

IV (B) Specification-based Detection

Specification-based detection is the derivate of anomaly-based detection. Instead of approximating the implementation of a system or application, specification-based detection approximates the requirements of application or system. In specification-based system there exists a training phase which attempts to learn the all valid behaviour of a program or system which needs to inspected. The main limitation of specification based system is that it if very difficult to accurately specify the behaviour the system or program. One such tool is Panorama which captures the system wide information flow of the program under inspection over a system, and checks the behaviour against a valid set of rule to detect malicious activity [2, 3].

IV (C) Behaviour -based Detection

Behaviour based detection differs from the surface scanning method in that it identifies the action performed malware rather than the binary pattern. The programs with dissimilar syntax's but having same behaviour are collected, thus this single behaviour signature can identify various samples of malware. This types of detection mechanisms helps in detecting the malwares which keeps on generating new mutants since they will always use the system resources and services in the similar manner. The behaviour detector basically consists of following components which are as follows:

- *Data Collection:* This component collects the dynamic / static information's are captured.
- *Interpretation:* This component converts the raw information collected by data collection module into intermediate representations.
- *Matching Algorithm:* It is used to compare the representation with the behaviour signature.

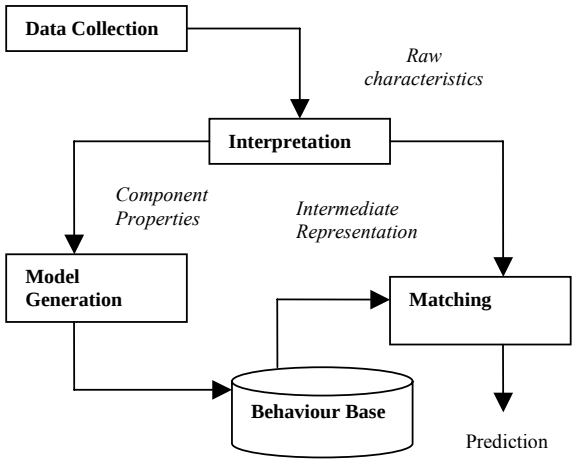

Fig. 4 Behaviour Detector [14]

V. Obfuscation

Obfuscation is to hide the information such that others cannot find the true meaning. Software vendors make use of obfuscation so that the software would be difficult to reverse-engineer. Malware writers take it as advantage and obfuscate the malicious program using various obfuscation transformations so that the Malware is difficult to reverse-engineer and hence its malicious intent cannot be learned.

V (A) Obfuscation theory

Given a program *P* and a transformation function *T* generates program *P'* such that the following properties holds true:

- *P'* is difficult to reverse engineer.
- *P'* holds the functionality of *P*.
- *P'* performs comparable to *P*

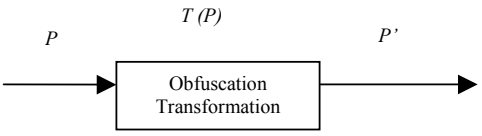

Fig. 5. Obfuscation

Many metamorphic and polymorphic make use obfuscation techniques so that they can defeat the signature based detection .Obfuscation techniques can easily change the signatures of Malware. Let us first look at some example of obfuscation technique modifying the signature of the code given below.

Original Code

Hex Opcodes	**Assembly**
51	push ecx
50	push eax
5B	pop ebx
8D 4B 38	lea ecx,[ebx+38h]
50	push eax
E8 00000000	call 0h
5B	pop ebx
83 C3 1C	add ebx,1Ch
.	
.	

.

Signature

5150 5B8D 4B38 50E8 0000 0000 5B83 C31C

Now suppose the original code is obfuscated by inserting a bunch of junk instruction like *nops.* Then the obfuscated code and the new signature is as follows:

Original Code

Hex Opcodes	**Assembly**
51	push ecx
90	**nop**
50	push eax
5B	pop ebx
8D 4B 38	lea ecx,[ebx+38h]
50	push eax
90	**nop**
E8 00000000	call 0h
5B	pop ebx
83 C3 1C	add ebx,1Ch

Signature

5190 505B 8D4B 3850 90E8 0000 0000 5B83 C31C

Thus the change in signature is not detected by Malware scanner and the false negative rate will increase enormously. Common obfuscation techniques fall into following main categories:

a) Dead-code-insertion
b) Code transportation
c) Register Renaming
d) Instruction Substitution

V (B) Dead-code-insertion

The example shown above falls into the first category, i.e. dead-code-insertion. This is can be done by either inserting bunch of *nops* (that does not accomplish anything), or inserting some number of *push x* followed by *pop x*, where *x* refers to register.

V(C) Code Transportation

Code transportation is done by inserting jump instruction or unconditional branch instructions so that the original control flow of the program is maintained. Other techniques such that swapping instructions which are not interdependent also exist.

Using the same example in section 4, this technique would yield a code that would look as follows:

```
        push ecx
        push eax
        jmp A
A:      pop ebx
        lea ecx,[ebx+38h]
        jmp B
C:      pop ebx
        add ebx,1Ch
B:      push eax
        call 0h
        jmp C
```

V (D) Register Renaming

This technique replaces the use of a register in an instruction with another unused instruction. Register replacement requires that no register dependencies in control flow are affected.

Original Code

```
Mov eax,32
Mov ecx,1024
Push eax
Sub eax,eax
Push eax
Mov eax,ecx
Add eax,2
Mov [eax+2],ebx
Pop eax
```

Transformed Code

```
Mov eax,32
Mov ebx,1024
Push eax
Sub eax,eax
Push eax
Mov eax,ebx
Add eax,2
Mov [eax+2],ebx
Pop eax
```

V (E) Instruction Substitution

A sequence of instructions is associated to a set of alternative sequence of instructions which are semantically equivalent to the original one. Every sequence of original instructions can be replaced by some arbitrary instructions. For example the following code is equivalent.

Original	**Transformed**

MOV reg,imm	a)MOV reg,Random
	ADD reg,imm-Random
	b)MOV reg,Random
	SUB reg,-(imm-Random)

MOV eax,eax	XOR eax,eax

Obfuscated Malware can be detected by collecting multiple variants of same malware sample and performing similarity analysis between the variants or generating the normalized code.

VI. Similarity Analysis

The known sample of malware such as Win32.Evol [4] is taken and multiple files of the same sample is generated by obfuscating the known sample. Similarities between the files are checked to find whether the variant is the child of the sample under inspection. Similarity analysis using Euclidean normal form [12] can be used to find the distance between some vectors *x* and *y* as:

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

A program is represented as some number of functions *f,* and each function contains some number of statements which are termed as vectors *x* and *y*. The total number of vectors for the same program *P* and for all function *f* is kept same. Similarity analysis can be performed by using cosine similarity measure [6] which is primarily used in text mining. SAVE tool uses the average of various similarity measures to estimate maliciousness of the program [7].

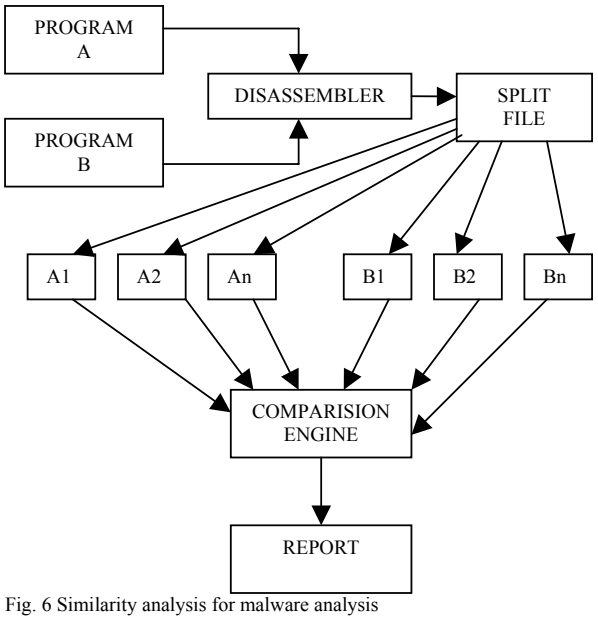

Fig. 6 Similarity analysis for malware analysis

Following steps are adopted to detect malicious nature of program.

Step1: Program executable is decompressed (optional) if the program is compressed.

Step2: Decompressed program is disassembled using the disassembler like IDA PRO [10] or OllyDbg[11].

Step3: Each disassembled program is represented as a vector of functions. Each function is represented as array of equal length.

Step 4: The similarities between the functions of program *P'* and *P''* is computed using Euclidean normal form or cosine similarity measure or Pearson's similarity measure or Jaccard similarity measure.

Step 5: The value of the similarity is compared with the threshold value; if the value is very less than the threshold value then the program under inspection is benign otherwise malicious.

Note the choice of similarity is crucial. A high value of threshold increases the risk of false negative and low value increases the risk of false positiveness.

VII. Malware Normalization

A Malware normalizer accepts the obfuscated version of Malware and eliminates the obfuscation carried on the program and produces the normalized executables .Thus it can be said that the Malware normalizer increases the detection rate of the detector. Malware *M* is taken and passed through the normalizer. After normalization the signature of this Malware is extracted and compared with the signatures of canonical form [8]. Maximum length of matching signature of canonical form with the Malware is considered and the new signature of the canonical form is stored in the signature database for future comparisons. Let us consider a Malware *'M'* and $c_i \varepsilon C$ is canonical form of the malware M.

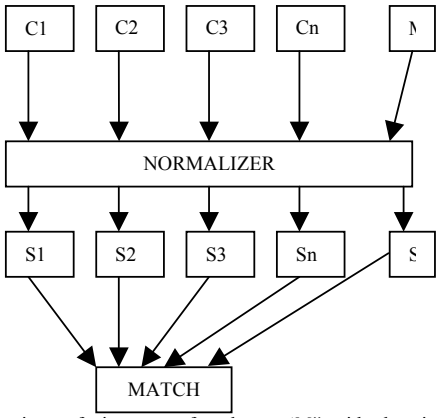

Fig. 7 Comparison of signature of malware *'M"* with the signatures of canonical forms c_i *'s*.

Following are steps involved in malware detection using malware normalizer:

Step 1: Malware PE binary code is decompressed (optional) using decompression software.

Step 2: The decompressed code obtained from above step is disassembled using the standard disassemblers.

Step 3: The disassembled code is then passed through the normalizer. The normalizer check for obfuscation performed eliminates obfuscation and produces normalized code.

Step 4: The normalized code is passed to the malware detector which extracts the signature of the normalized code, compares it with the signature stored in signature repository.

Step 5: The comparison is based on maximum match of the signature stored in the repository with the signature normalized code. For matching any sequence alignment algorithm can be adopted. Finally the signature of normalized code is stored in database for future signature comparisons with other variants.

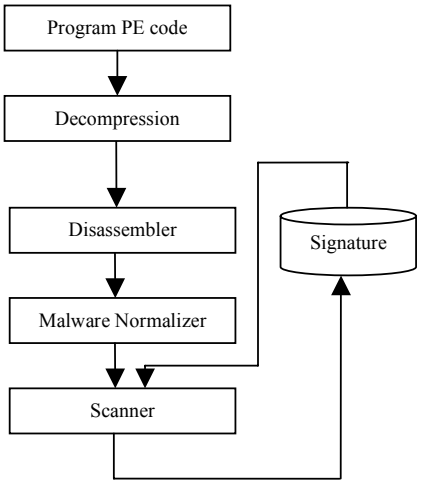

Fig.8. Malware Normalization and signature comparison [13]

Malware signatures are long and similarity analysis between signatures of different samples takes more number of comparisons. Thus there exists need of shorter signatures. Similarity analysis based on API sequence is used to detect polymorphic variants [9].The suspicious PE file is optionally

decompressed. Then it is passed through the PE parser. The output of PE binary parser is a sequence of API calls. This API sequence is 32 bit id. The first 16bit represents a API module and last 16 bit represents specific API within an API module. The API sequence call is compared with the sequence of calls of known malware. Then a similarity between these API's sequence is performed, which is compared against threshold value .If the value is lesser than threshold value then the sample under inspection is reported as benign otherwise malicious.

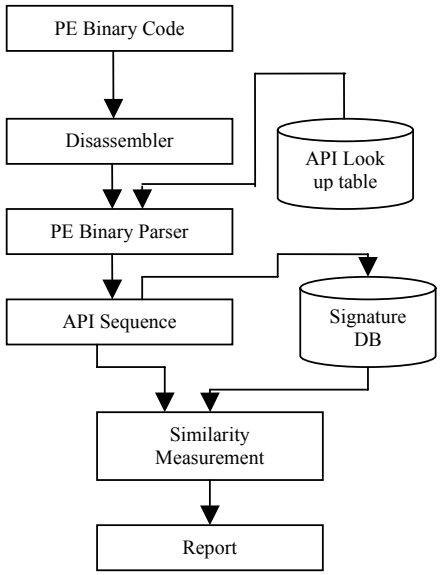

Fig.9 Similarity measure based on API sequence

VIII. Conclusion

In this survey a series of malware detection techniques have been presented. The problems related to traditional signature based detection method is also highlighted. Rate of new malware's causing destructions to systems world wide is increasing at alarming rate. Detection of malware's changing their signatures frequently has posed many open research issues. Challenge lies in the development of good disassembler , similarity analysis algorithm so that the variants of malware's can be detected in shorter time there by reducing the computation overhead.

References

[1] Mihai Christodorescu and Somesh Jha ," Testing Malware Detectors", in Proc. ISSTA'04, July 11 - 14, 2004.pages 33-44, Boston, MA USA, ACM Press.

[2] Heng Yin ,Dawn Song ,Manuel Egele,Christopher Krugel , and Engin Kirda , "Panorama: Capturing System – wide Information Flow for Malware Detection and Analysis", in Proc CCS'07, October 29 – November 2, 2007, Alexandria, Virginia, USA,ACM Press.

[3] Andreas Moser , Christopher Krugel , and Engin Kirda, "Exploring Multiple Execution Paths for Malware Analysis", Secure Systems Lab, Technical University Vienna.

[4] Arun Lakhotia , Aditya Kapoor , Eric Uday , "Are Metamorphic Viruses Really Invincible ? Part 2" , Virus Bulletin ,January 2005.

[5] Abhishek Karnik , Suchandra Goswami and Ratan Guha," Detecting Obfuscated Viruses Using Cosine Similarity Analysis", *in The Proceeding of IEEE Symposium First International Conference on Modelling & Simulation (ASM '07)*

[6] A.H.Sung, J.Xu, P.Chavez, S.Mukkamala,"Static Analyzer of Vicious Executables (SAVE)", Proceeding of the 20th Annual Computer Security Applications Conference (ACSAC 2004), 2004.

[7] Mihai Christodorescu , Johannes Kinder , Somesh Jha , Stefan Katzenbeisser , Helmut Veith , "Malware Normalization ", University of Wisconsin and Madison Technische Universit at M unchen , 2005.

[8] J.Xu, A.H.Sung, P.Chavez, S.Mukkamala,"Polymorphic Malicious Executable Scanner by API Sequence Analysis", Proceeding of 4th IEEE Symposium of International Conference on Hybrid Intelligent Systems (HIS '04).

[9] DataRescue, Inc. The ida pro disassembler www.datarescue.com/idabase,2006

[10] O.Yuschuk. 80x86 32 bit disassembler and assembler.www.ollydbg.de/srcdescr.htm,2006

[11] Mohamed R.Chouchane and Arun Lakhotia,"Using Engine Signature to Detect Metamorphic Malware", In Proc. Worm06 ,November3,2006, Alexandria, Virgina, USA, ACM Press.

[12] Mohamed R.Chouchane, Andrew Walenstein and Arun Lakhotia," Statistical Signature for Fast Filtering of Instruction-substituting Metamorphic Malware", In Proc. Worm07, November 2, 2007", Alexandria, Virgina, USA, ACM Press.

[13] Ran Jin,Qiang Wei,Pei Yang and Qingxian Wang ,"Normalization towards Instruction Substitution Metamorphism Based on Standard Instruction Set", In Proc. IEEE symposium on 2007 International Conference on Computational Intelligence and Security Workshops.

[14] Greoigre Jacob,Herve Debar,Eric Fillol,"Behavioral detection of malware:from a survey towards an established taxonomy",Springer-Verlag France 2008

[15] Gerard Wagener,Radu State,Alexandre Dulaunoy,"Malware Behavior Analysis",Springer-Verlag France 2007.