Detection of Malicious Code-Injection Attack Using Two Phase Analysis Technique

D. Swathigavaishnave
M.Tech Student, Department of Computer Science and Engineering
Pondicherry Engineering College
Puducherry, India

R. Sarala
Assistant Professor, Department of Computer Science and Engineering
Pondicherry Engineering College
Puducherry, India

ABSTRACT
In today's world code injection attack is a very big problem. Code injection attacks are to exploit software vulnerabilities and inject malicious code into target program. These malicious codes are normally referred as malware. Systems are vulnerable to the traditional attacks, and attackers continue to find new ways around existing protection mechanisms in order to execute their injected code. Malicious code detection is an obfuscation-deobfuscation game between malicious code writers and researchers working on malicious code detection. Malware writers obfuscate their malicious code to subvert the malicious code detectors, such as anti-virus software. Signature-based detection is the most commonly used method in commercial antivirus software. However, it fails to detect new malware. In this paper, we propose a two phase analysis technique. In first phase a malicious code with obfuscated techniques is detected by means of static analysis of instruction sequence. Phase II involves extracting opcode sequence from the dataset to construct a classification model and compare it to the output of phase I to identify it as malicious or benign.

Keywords
Obfuscation techniques, static analysis, classification algorithm

1. INTRODUCTION
Malicious codes are pieces of code that can affect the secrecy, the integrity, the data and control flow, and the functionality of a system. Therefore, their detection is a major concern within the computer science community as well as within the user community. As malicious code can affect the data and control flow of a program, static flow analysis may naturally be helpful as part of the detection process. Still various approaches are evolved to detect malicious code. Various anti-virus scanners are used to detect the malicious code. But all these techniques are signature based approach. A signature is a unique sequence of bytes that is always present within malicious executables and in the files already infected.

The main issue of this approach is that malware analysts must wait until new malware has harmed several computers to generate a signature and provide a solution. Analyzed suspect files are compared with this list of signatures. When the signatures match, the file being tested is classified as malware. Although this approach has been proven as effective when threats are known in beforehand, these signature methods are surpassed with large amounts of new malware.

Another most important problem in anti-virus packages is that signature database must be updated regularly to find new malware. A malware can be easily modified by means of simple obfuscation techniques.

Malwares writers may use various obfuscation techniques to hide themselves from the various anti-virus tools. Simple obfuscation involves inserting NOP (no operation) instructions, swapping registers, and reordering independent instructions. Malware can be obfuscated using two techniques: Polymorphic techniques and metamorphic techniques.

In creating new malware, black hats generally employ one or both of the following techniques: obfuscation and behavior addition/modification in order to circumvent malware detectors. Obfuscation attempts to hide the true intentions of malicious code without extending the behaviors exhibited by the malware. Behavior addition/modification effectively creates new malware, although the essence of the malware may not have changed.

2. RELATED WORKS
In this section we discuss about various techniques used for malicious code detection.

In [1] author used data mining techniques for extracting variable length instruction sequences that can identify Trojans from clean programs. The analysis is facilitated by the program control flow information contained in the instruction sequences. Based on general statistics gathered from these instruction sequences, support vector machine classifier is trained.In [2] author presents CWSandbox, which executes malware samples in a simulated environment, monitors all system calls, and automatically generates a detailed report to simplify and automate the malware analyst's task. It monitors all the executed functionality. [3] Focuses on deobfuscation of actual obfuscated code in order to reveal true intent of that piece of code. [4] Proposed a statistic-based metamorphic virus detection technique and proves that detection based on statistics is a useful approach in detecting self-mutated malwares.

Six statistic features that include percentage of NOP instruction at the end of subroutines, percentage of NOP instruction in random, JMP instruction profile, short jump instruction profile, all subroutine profile, and subroutines without a CALL instruction were taken.

In [5], a behavior-based detection approach is proposed to address malware detection. The behaviors of interest are defined as static system call sequences and they are derived by statically analyzing binary code.

Machine-learning methods, including the K-nearest neighbor, Support Vector Machine, and decision tree methods are used to classify executables. SigFree uses a new data-flow analysis technique called code abstraction that is generic, fast, and hard for exploit code to evade. SigFree is signature free, thus it can block new and unknown buffer overflow attacks; SigFree is also immunized from most attack-side code obfuscation methods [6].

Basic common Techniques used for detecting malware can be categorized as shown in the fig.1

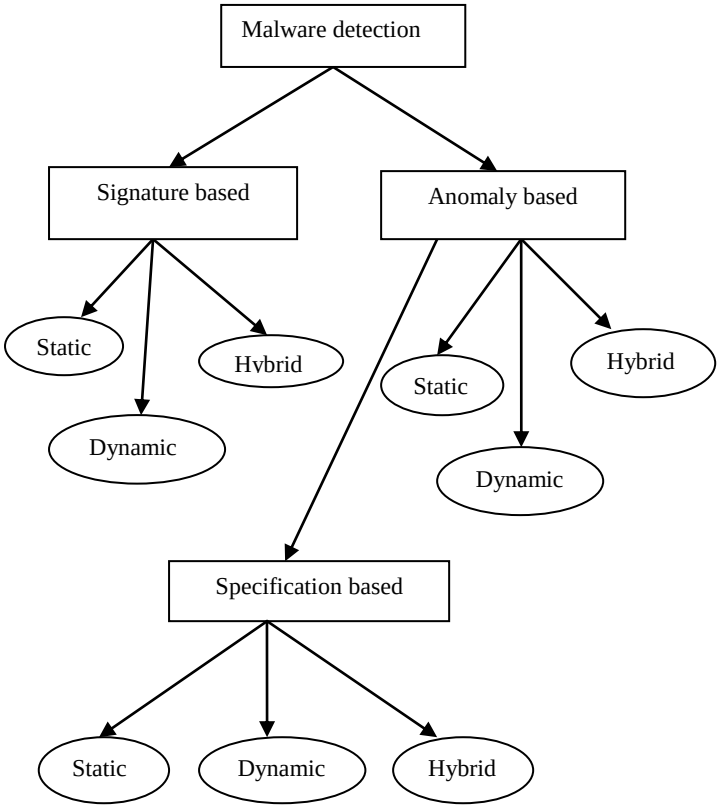

Fig.1 Techniques used for detecting malware

3. METHODS FOR OBFUSCATION

General code obfuscation techniques aim to confuse the understanding of the way in which a program functions. These can range from simple Layout transformations to complicated changes in control and data flow.

Programmers obfuscate their code to defeat manual analysis and signature based detection. Obfuscations are used to hide malicious behavior.

Given a code C and a transformation function T generates code C' such that the following properties holds true:

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

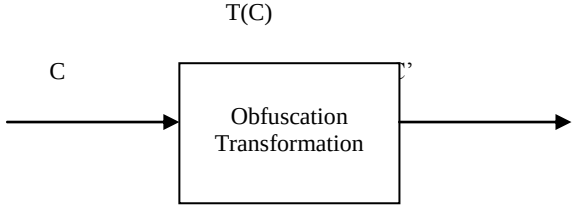

Fig.2 obfuscation

Obfuscation technique modifying the signature of the code is given below.

Original Code

Hex Opcodes	Assembly code
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 the original code is obfuscated by inserting a bunch of junk instruction like nop's. Then the obfuscated code and the new signature are as follows:

Modified 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

3.1 Garbage Insertion/ Dead-code insertion

Garbage insertion adds sequences of instruction which does not modify the functionality and behavior of the program. This insertion can be done anywhere in the program. Main purpose of junk code insertion is to change its byte value used as signature. There are various methodology used for garbage code insertion such as a sequence of NOPs (no operation instructions).another type of dead code insertion is

push ax
pop ax

Without any change, value is returned to the register from the stack.

inc bx **inc cx**
dec bx (or) **sub cx, 1**

In this the value of a register remains unchanged.

3.2 Code Reordering

The code reordering obfuscation changes the order of program instructions. The physical order is changed while maintaining the original execution order through the use of control-flow instructions (branches and jumps). Branches are inserted with conditionals defined and computed such that the branch is always taken. The conditional expression can be based on a complex computation. The execution order of instructions can be changed only if the program behavior is not affected. Independent consecutive instructions (without any dependencies between them) can thus be interchanged. Sample code transportation obfuscation of assembly source program is given below,

jz label
jnz label
label: malicious code

Although the instruction jz and jnz are conditional jumps, but the condition that decide the execution of these two instructions are complementary with each other. And the target addresses of the two conditional jump instructions are identical. Hence the function of these two conditional jumps is actually equal to one unconditional jump instruction. In this way, the conditional jumps obfuscation achieves the goal of hiding malicious codes.

3.3 Register Renaming

Register reassignment involves changing the usage of one register with another such as eax with ebx to evade detection.

Although all of these approaches change the code pattern in order to evade detection, the behavior of the malware still remains the same.

3.4 Instruction substitution

This obfuscation technique, involves substituting new instruction for existing one with their equivalent instruction.

By substituting an equivalent instruction its signature gets changed but its functionality remains the same. Example for this obfuscation is, to assign a value 0 to a register may take following forms,

mov eax, 0
xor eax, eax
and eax, 0
sub eax, eax

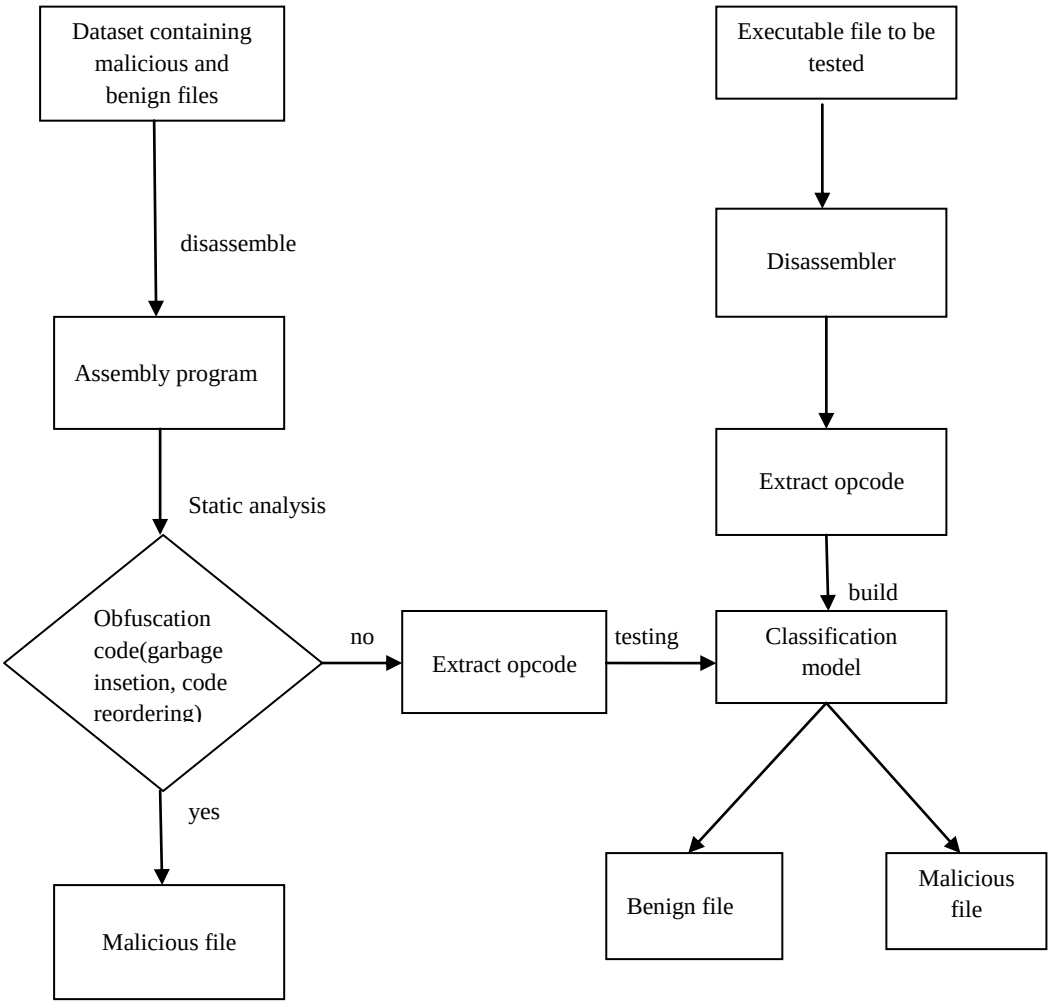

Fig.3 Overview of Proposed System

4. PROPOSED METHODOLOGY

Binary executable codes are stored as a sequence of bytes. Analyzing the byte sequence may detect malicious file but cannot detect obfuscated file. Both malicious and obfuscated file can be detected by analyzing an assembly code. Assembly code is obtained by disassembling an executable file. Various Disassembler tools are used for the conversion of byte sequence to assembly language.

Here we use IDA pro disassembler tool [7] to get assembly code.Static analysis is performed on the obtained assembly code. Static analysis can examine an executable to determine if it is malicious without running the code. In contrast, Dynamic analysis monitors the execution of an executable to detect malicious behavior. As compared with Dynamic analysis, Static analysis can exhaustively analyze an executable by evaluating every possible execution path.

In our proposed methodology to reduce false negative two phase analysis is used to detect malicious file.

Overview of our proposed architecture is shown in fig-2.

Algorithm for the detection of obfuscated file

Input: Executable file (F)
Output: malicious (containing obfuscated code) or benign
Step 1: disassemble a file into ASM file
Step 2: generate control flow graph (CFG), G= (V, E) where V is a set of vertices and E is a set of edges.
Step 3: for each node v €V of CFG
Checking for obfuscated code like
If v has NOP instruction then /*because legitimate file doesn't contain NOP*/
F is a malicious file because of obfuscation code.
Else
F is benign
End for

In the first phase, if any obfuscation is present in ASM file then there is malicious intent. But sometimes this may lead to false positives. Our proposed algorithm can detect only a file which uses obfuscation techniques. Normal malicious files without obfuscation are not detected. So for further analysis of a particular file, we extract the opcode sequence of a file and give it to the classification model to conclude whether it is benign or malicious [8].

In phase II, to train the classifier a dataset consisting of malicious files (worms, Trojans, virus) and benign files from the Vx-Heavens website [9] is created. A decision tree is constructed based on opcode sequence extracted from the dataset containing malicious and benign executable.

Algorithm for extraction of opcode sequence

Input: dataset containing malicious files (V) and benign files (B)
Output: set of opcode sequence (O)
Step 1: disassemble each file to get ASM file
Step 2: for each file vi of V do
Extract the opcode and ignore operands
Parse the extracted opcode based on jump instruction (conditional or unconditional)
Record all opcode sequence
If particular opcode sequence already exist
Increment the count of particular opcode sequence.
End for
Select most frequent top L opcode sequence to train the classifier

There are numerous Intel x86 instructions, so instead of considering all these instructions, it is logical to examine only the most frequently occurred instructions and eliminate the less frequently used instructions. Based on the research done by Bilar in [10], only 14 instructions in total set of Intel instructions are most frequently occurred. They are MOV, PUSH, CALL, POP, CMP, JZ, LEA, TEST, JMP, ADD, JNZ, RETN, XOR, and AND.
Let's consider a virus example. The process involved in our methodology is given below.

```
pop     edx
mov     edi,0004h
mov     esi,ebp
nop
mov     eax,000ch
add     edx,0088h
mov     ebx,[edx]
nop
mov     [esi+eax*4+0001118],ebx
```

Win95.regswap virus

Our methodology can easily detect this virus. It generates CFG for the disassembled code and analysis each block i.e. node. Our algorithm detects the presence of nop (no operation) instruction and identifies it as malicious. This virus code is actually obfuscated so in the first phase itself it is identified as malicious.

```
push    ebp
mov     ebp,esp
mov     esi,dword ptr[ebp+08]
test    esi,esi
je      401045
mov     esi, dword ptr [edp+0c]
or      edi,edi.
je      401045
xor     edx,edx
```

Portion of the output of disassembled w95.bistro virus This virus code is not obfuscated. Our first phase analysis cannot detect this code. So it is taken for further analysis in the phase II which proceeds as given below.

The first step involves extracting opcode and eliminating operands.

```
push
mov
mov
test
je
mov
or
je
xor
```

Opcode sequence

In second step, the obtained opcode is parsed until a jump instruction is encountered.

```
push mov mov test je
mov or je
xor
```

This step returns the parsed opcode sequence.

The Parsed opcode sequence is given to the decision tree and it identifies the code as malicious or benign.

Decision tree model is used to obtain a set of rules that can classify each sample into either malicious or benign class. The Decision Tree (J48) classifier has an excellent feature selection capability, and requires much less training and testing time than other classifiers.

5. EXPERIMENTAL RESULTS

Our proposed methodology can detect the malware along with obfuscation when compared with existing antivirus tools. Because of two phase analysis method, our technique can detect all type of malicious code and can reduce the FP and FN rate.

We collected 500 virus files from vx-heavens and corpus dataset and benign files from windows.

Input: calc.exe (windows system32 file)

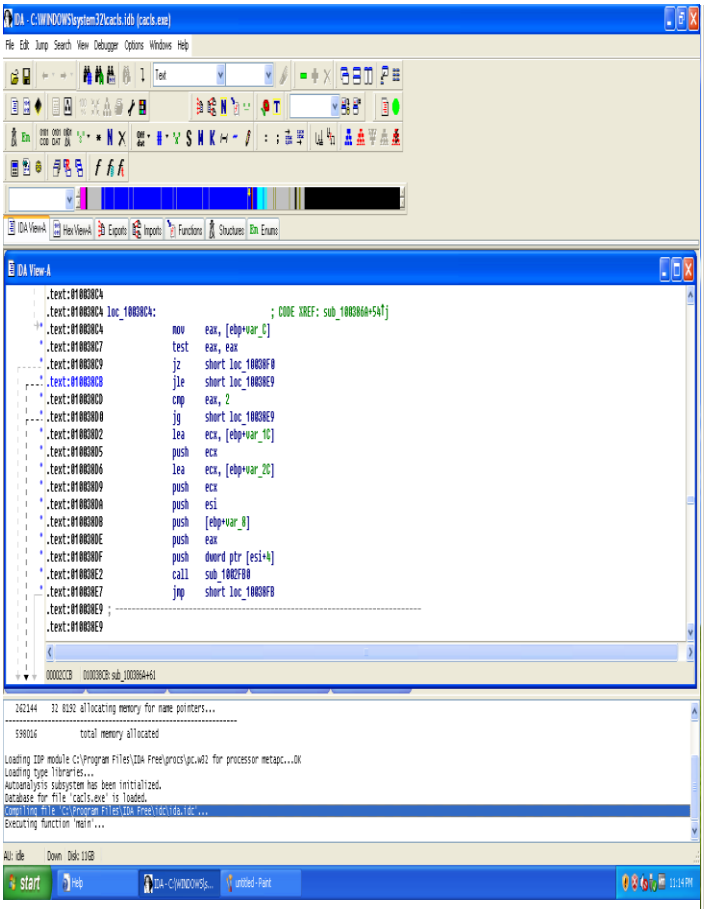

Fig.4 Disassembled ASM File

Generated asm file is analyzed for the presence of obfuscation. There is no obfuscation, the opcode is extracted from the file and compared to the classification model to classify it as benign or malicious.

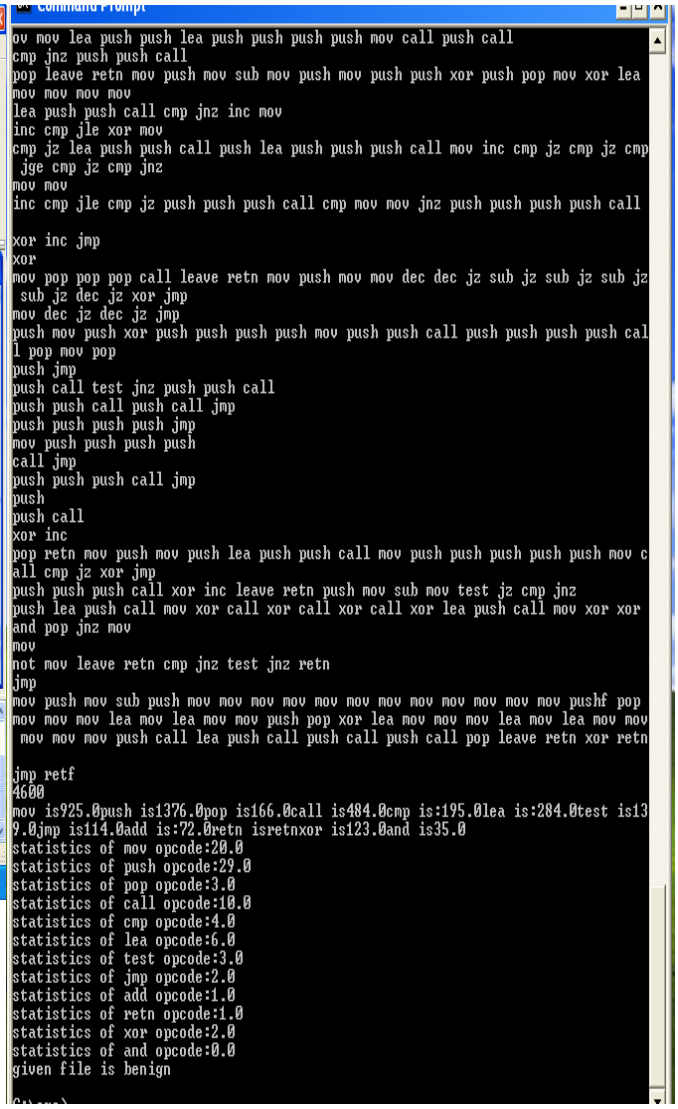

Fig.5 Extracted Opcode Sequence and the Output

Output: The given file does not contain obfuscated code. Hence it is classified as benign.

To reduce false positives and false negatives the opcode sequence is extracted and given to the classifier. It classifies the file as benign.

Input: ngvck025.exe

Output: The given file contains obfuscated code. Hence it is classified as malicious.

6. PERFORMANCE EVALUATION

The following metrics are used to evaluate our method with an existing system

True positive (TP): benign programs are correctly identified

True negative (TN): malicious programs are correctly identified.

False positive (FP): benign programs are wrongly identified as malicious.

False negative (FN): malicious programs are incorrectly classified as benign.

The performance of our methodology was evaluated using the true positive rate, false positive rate which are defined as follows,

True positive rate (TPR): percentage of benign programs correctly identified.

TPR= (TP/TP+FN)

False Positive Rate (FPR): percentage of malicious programs wrongly identified.

FPR= (FP/TN+FP)

500 virus file and 300 benign file are given as input. From which the accuracy of true positive rate (TPR) of our proposed methodology is higher than existing system and false positive rate (FPR) of our proposed methodology is lower than existing system.

Table.1: performance evaluation of proposed work

methods	TPR	FPR
Signature based detection	0.950	0.700
Proposed methodology	0.992	0.530

Table.2 Comparison of proposed and existing work

malware	Signature based detection	Proposed method
Obfuscated virus files	**Cannot detect**	**Can detect**
Malicious files without obfuscation	**Can detect**	**Can detect**

7. CONCLUSION

In this paper, we have proposed a two phase analysis technique to detect malicious code injection attack by using static analysis and classification model constructed by frequency of occurrence of opcode extracted from a dataset.

Experimental results indicate that the proposed methodology can detect both obfuscated malicious files and malicious files without obfuscation. Since we are using the two phase analysis technique, files with obfuscated code is detected in first phase by static analysis and there is no need of the second phase. Files without obfuscated code are detected in second phase by classification model which classifies them as malicious or benign.

Our methodology cannot detect register renaming obfuscation technique. We plan to detect this obfuscation technique in our future work.

7. REFERENCES

[1] D.M.A. Hussain et al. (Eds.): "Detecting Trojans Using Data Mining Techniques", CCIS 20, pp. 400–411, 2008.Springer-Verlag Berlin Heidelberg 2008.

[2] Carsten Willems, Thorsten Holz, Felix Freiling: "Toward Automated Dynamic Malware Analysis Using CWSandbox", IEEE Security and Privacy, vol. 5, no. 2, pp. 32-39, Mar/Apr, 2007.

[3] A. Lakhotia, E. U. Kumar, M. Vennable, "A Method for Detecting Obfuscated Calls in Malicious Binaries", IEEE transactions on Software Engineering, Vol 31, No 11, November (2006).

[4] Govindaraju. A, Faculty, Department of Computer Science, Master Thesis, "Exhaustive Statistical Analysis for Detection of Metamorphic Malware". San Jose State University, San Jose, CA (2010).

[5] Ding Yuxin*, Yuan Xuebing, Zhou Di, Dong Li, An Zhancha," Feature representation and selection in malicious code detection methods based on static system calls"Computers & Security (2011) ,article in press,science direct journal.

[6] Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu "SigFree: A Signature-Free Buffer Overflow Attack Blocker" ieee transactions on dependable and secure computing, vol. 7, no. 1, january-march 2010.

[7] IDA Pro Disassembler and Debugger, http://www.hex-rays.com.

[8] Raviraj Choudhary and Ravi Saharan malware Detection Using Data Mining Techniques" international Journal of InformationTechnology and Knowledge Management January-June 2012, Volume 5, No. 1, Pp. 85-88

[9] VXheavens http://vx.netlux.org

[10] Bilar. D," Statistical Structures: Fingerprinting malicious code through statistical opcode analysis", 3rd International Conference on Global E-Security, ICGeS 2007 (2007).