Volume 3, Issue 10, October 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

An Opcode Statistical Analysis for Metamorphic Malware

Shiv Kumar Agarwal[1], Vishal Shrivastava[2]

Department of Computer Science and Engineering

Arya College of Engineering & I.T., Jaipur, India

Abstract—A common technique that virus writers use to avoid detection is to enable the virus to change itself by having some kind of self-modifying code. Metamorphic viruses, which are arguably the most dangerous of all, change their structure or signature each time they propagate without changing the functionality of the virus. Metamorphism provides one of the strong known methods for evading malware detection. If the virus is metamorphic it could potentially be difficult to find a single signature that will consistently be found in every version of a metamorphic virus. Signature based detection systems are unable to detect metamorphic viruses, since such viruses change their internal structure from generation to generation. In this paper we have provided a sophisticated way to find some unique features such as most frequently occurred (MFO) opcodes that may be considered as a predictor for metamorphic malware detection.

Keywords— Most frequently occurred opcodes, Metamorphic malware, Obfuscation, Polymorphic, Mutation engine.

I. INTRODUCTION

Malware is short for 'malicious software' and is another term for 'computer viruses' [15]. A malware is virus software spread through malicious programs or software known as malware. Malware is designed to delete, block, modify or copy data, or disrupt the performance of computers or computer networks [3]. "Malware" is the general term covering all the different types of threats to your computer safety. The term malware includes viruses, worms, trojan horses, rootkits, spyware, keyloggers and more[11],[12]. Once malwares enter to the system, they start to find the vulnerabilities within the operating system then perform unintended operation in the system. Most of the malwares basically attack on performance of the system, data integrity and privacy [1]. They also play the major role in denial of service attack [1], [2]. These malwares are also capable to infect other executable files and data. Malwares depending on their behaviour collect the information about host and harm the host computer without consent of the owner. The exceed utilization of Internet has increased the appearance of malware in digital world. Since our generation relies too much over the computers, they faced many problems in their daily lives with the appearances of malware in their computer systems. All softwares which are developed with some malicious intention, considered as a malware. They are categorized into two major classes based on the type of propagation. Spyware, BotNets, Worms etc. are considered as mobile malware and Viruses are referred to static malware [14]. Viruses are having the ability to replicate and distribute themselves over the network. They are also having the serious impact on individual computer systems and corporate.

Today, there are some malware families known as modern malware families, which have the capability to change the signature of new variant in each generation by using a variety of code obfuscation techniques [13]. Our proposed method "An opcode statistical analysis for metamorphic malware" is trying to find some of the most critical opcodes called as MFO opcodes, which appeared frequently in any portable executable and having comparable frequencies to distinguish metamorphic malware variants from benign samples. Proposed method provides us a better analysis of metamorphic malware to improve their detection rate in future. This paper is organized as follows. In section 2, we described the related work. Section 3 explores the proposed method, and then section 4 shows the experimental results. Finally, we conclude in section 5.

II. RELATED WORK

In their proposed work, authors [7] and [8] created a rewriting engine for detecting morphed malware variants. The analysis of variants of malware is based on syntactic as well as semantic structure of a program. Krugel *et al* [9] proposed a method based on the CFG generated for worms, which describes a fingerprint for worm. Their system is found to be resilient against common code transformation techniques. Authors in [10] proposed a novel method for analyzing malware based on code graph. Each malware executable was inspected and instructions corresponding to system call sequence were represented in the form of a topological graph. In their proposed work [4], authors proposed a semantic based approach for detecting variants of malware. This method is based on the functionality of system call executed by malware samples. The main focus is to identify all instructions and its parameters which are used for calling a system call.

In [5], Hidden Markov Models (HMMs) were used to represent statistical properties of a set of metamorphic virus variants. Malware data set was generated from metamorphic engines: Second Generation virus generator (G2), Next Generation Virus Construction Kit (NGVCK), Virus Creation Lab for Win32 (VCL32) and Mass Code Generator (MPCGEN). Also, the authors [16] found that HMM based detector can be defeated if the malware variants are generated

by inserting garbage instructions which are extracted from subroutines of benign files. The authors in [1] proposed a *"phylogeny"* model for identifying malware. The n–gram feature extraction technique was proposed and fixed permutation was applied on the code to generate new sequence, called n-perms.

III. PROPOSED METHOD

Metamorphic malware changes its shape in each generation. During the execution of metamorphic malware different kinds of obfuscation techniques are applied by the mutation engine to change the structure of new variant while keeping the same behaviour. To maintain the same behaviour, it cannot mutate its variant at large extent. Therefore, the detection of such malware is still possible. In order to identify the variants of metamorphic malware, a strong algorithm is required which can handle all kind of possible mutation in the original code. As we know that even a simple change in existing code can make the significant change in its new generation. It makes the job of antivirus products more difficult. Today, metamorphic malware have been created the serious problem in front of antivirus researchers [6]. It is the most challenging task in present scenario to deal with such type of advanced malware. Now it has become very popular among antivirus writers. This is one research area in which several researchers are working individual or in group. They are trying and putting their efforts to invent effective detection system for metamorphic malware. When we did start to work on malware then day by day we get excited to know more and more about such malware. In this thesis work, our prime motto is to get familiar with metamorphic malware and identified the features (MFO Opcodes) which can act as a predictor in detection of such malware.

In our proposed modal "An opcode statistical analysis for metamorphic malware", we have considered two families of metamorphic malware as given below:

1) Next Generation Virus Construction Kit (NGVCK)
2) Virus Creation Lab for Win32 Virus (VCL32)

We have performed our experiments with the 100 samples of NGVCK, 100 samples of VCL32 and 100 benign samples. NGVCK and VCL32 are virus construction kit available on VXheavens website. These virus construction kit are assumed to be good in construction of strong metamorphic malware. In each execution, they mutate the original virus sample by employing the variety of code obfuscation techniques in order to generate the new variant with different signature and same behaviour. During the experiments, first we have disassembled all 300 executable files with the help of objdump utility program. After that we have fetched the opcode from each disassembled code and count their frequency in each portable executable file. As we know, all executable files in windows are having the portable executable (PE) format while in case of UNIX operating system, extensible linking format (ELF) is considered as one of the standard format for all UNIX executables. In this work, we are working on window's malware; therefore at each stage we are using the word "Portable Executable".

A. Most frequently occurred opcodes

In depth analysis of opcode's frequencies, we observed that only few opcodes termed as most frequently occurred (MFO) opcodes are redundantly appeared in each portable executable while other opcodes are rarely appeared and their average frequency in each family (either malware or benign) occurred in negligible amount. Based on our analysis, we observed that these few opcodes around 17 in quantity have major role in construction of any portable executable. Therefore, they may be considered as a predictor for metamorphic malware detection. These 17 opcodes are having the comparable frequencies in both benign and malware samples and can be used in any detection approach those we have studied in our literature survey where opcodes were considered as prime element in their detection algorithm. With the use of these MFO opcodes with any detection approach, we can get almost same results with very high performance. The detailed description about these 17 opcodes or you may say MFO opcodes is given in Section IV. In literature survey, we found that many researchers proposed their detection modal with the use of entire opcodes or opcode sequence as appeared in the PE file. In their work they obtained very effective results but as the running time of their proposed method is concerned they got hopeless. The number of opcodes during the disassembling process of portable executable was very high and non effective opcodes were increasing the running time of their proposed modal. The main concept behind our work is to explore only effective opcodes from PE file which may employ a significant impact on performance and accuracy of any detection modal utilized opcodes as one of the most powerful feature inside.

B. Algorithm Design

In this section, we are providing the brief introduction about our proposed method while detailed explanation of each step is given in section 4. Throughout the implementation, we learn lot of things about malware and their working. Each step involved in the proposed method is properly addressed and evaluated. In our experiment, we have performed some set of steps that have been formulated in form of algorithm as given below:

Algorithm_MFO_for_prediction *(100 NGVCK, 100 VCL, 100 Benign)*

{

Call_Disassembler (100 NGVCK, 100 VCL, 100 Benign)
Call_Opcode_Fetcher (Disassembly code)
Call_Frequency_Counter (Fetched opcode)


```
        For each family
                {
                    Call_Average_Opcode_ Frequency (family | Ngvck, Vcl32, Benign)
                }
        Call_Average_Average_opcode_Frequency (Ngvck.csv, Vcl32.csv, Benign.csv)
        Call_sort_Opcodes_inrespectof_Average (($) having Ngvck, Vcl32, Benign, Average)
        Call_MFO_Fetcher (Opcodes, Average)
}
```

In the proposed method, total 7 modules are called one by one to accomplish the task for getting the desired outcome. These seven modules are mutually depended on one another as the outcome of one module becomes the input for another module. It appeared as a scenario like interprocess communication between two commands through a pipe. Therefore, to get the desired outcome, it is necessary for us to execute these functions in order as defined in the above algorithm.

The main purpose of proposed algorithm is to extract the MFO opcodes from PE files which is achieved through module 7 named as "***Call_MFO_Fetcher***". The detailed description of each module we presented with the help of experiments in section 4.

IV. EXPERIMENTAL RESULTS

In order to make the detection system more effective in terms of metamorphic malware, we have identified the MFO opcodes from PE files. These MFO opcodes can be further utilized as a predictor for metamorphic malware detection. In this thesis, we have implemented our work with the help malware samples generated from VCL32 and NGVCK kit. These virus construction kits are very popular among the malware researchers due to the low detection rate of malware samples generated by these kits. All commercial antivirus products are not effective to handle such type of modern malware. These virus construction kits are embedded with mutation engine, which is responsible to make the significant change in their generated variants.

A. Experimental Setup

We have implemented our work with malwares generated from VCL32 and NGVCK kit on Ubuntu 11.10. In this project work, we have used Borland Turbo Assembler (TASM) version 5.0 under DosBox version 0.74 to compile the assembly code of virus program. System configuration consists of Intel(R) Core(TM) i3-2328M CPU @ 2.20 GHz with 4 GB RAM.

B. Implementation process

We started our implementation process, with the use of virus generation kits as described in section III. We have created the multiple variants of different families. In this section, we have described the implementation process with malware families constructed by various virus construction kits such as VCL32 and NGVCK.

Utilizing the VCL32and NGVCK kit, we created assembly code of virus program, which is further assembled through Turbo assembler to generate portable executable file. For assembling the virus code, we had used different commands in Borland Turbo Assembler (TASM) version 5.0 under DosBox version 0.74. DosBox is an X86 emulator with sound and graphics. DosBox creates a shell, which looks like old plain DOS. After successful compilation of source file, turbo assembler creates corresponding object file. In our observation, we found that many malware writers recommend TASM for corresponding Microsoft assembler (MASM). Object file is further passed to turbo linker (version 1.6.71.0) to link it with kernel32 libraries. Turbo linker produces portable executable file after successful completion of linking phase.

C. Proposed Algorithm

Proposed algorithm "An opcode statistical analysis for metamorphic malware" has already been discussed in Section III. In this section, we have shown the practical implementation of our proposed method which has been done with the use of our dataset as discussed in section III.

After successfully collecting the entire dataset of malware and benign samples, first we compiled the assembly code of virus programs through turbo assembler in order to get the required portable executable format for further processing. After that we have provided the following nomenclature to each of them to make the implementation easier to understand.

- 100 variants from VCL32 (labeled vcl_001 to vcl_100).
- 100 variants from NGVCK (labeled ngvck_001 to ngvck_100).
- For the benign samples, we had used 100 files.

After proper nomenclature, we stated our implementation with the module 1 as discussed in section III. Module 1 is basically used to disassemble the all data set containing 300 PE files (100 benign and 200 malware samples of VCL32 and NGVCK). We have designed this module in order to get the disassembly code for further analysis of opcodes. The detailed description about this concept has been provided in module 1.

Module 1: *Call_Disassembler (100 NGVCK, 100 VCL, 100 Benign)*

It is a reverse engineering process which is used to produce the equivalent assembly code of any portable executable file. Disassembled view of malware executable with objdump (objdump -D -M Intel ngvck_001.EXE) is shown in figure 1.

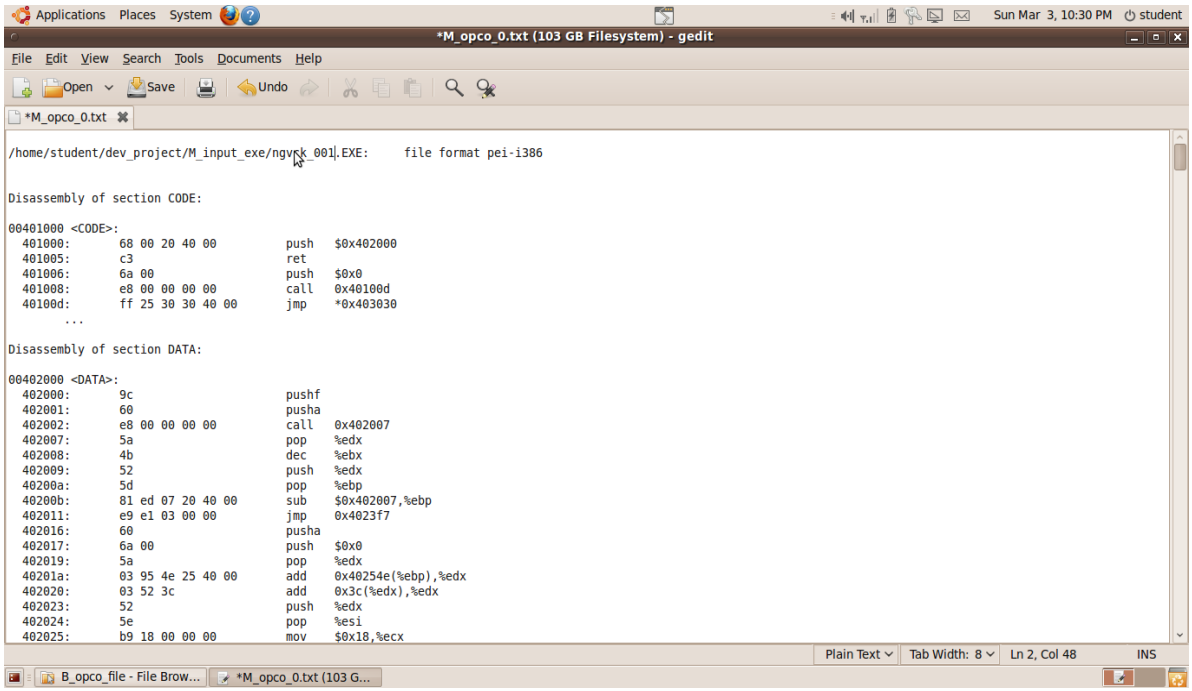


Fig.1 Disassembled view of NGVCK virus

Objdump is a linear disassembler. It is utility program which comes under the UNIX environment. It is one of the free open source popular disassemble program which has been used to disassemble our dataset of portable executable files in order to get the required assembly code for further analysis of opcodes and their frequency collectively in benign and malware samples.

Module 2: *Call_Opcode_Fetcher (Disassembly code)*

In the process of opcode fetching, we have extracted the opcodes only form the disassembled code as shown in figure 2. In this process we have discarded the operands associated with each opcode in disassembly program in order to effective analysis of variants, employ register renaming type of obfuscation technique. The disassembly program contains thousands lines of code in some malware and benign samples and our opcode fetcher fetches all the opcodes line by line as they appeared in disassembly program.

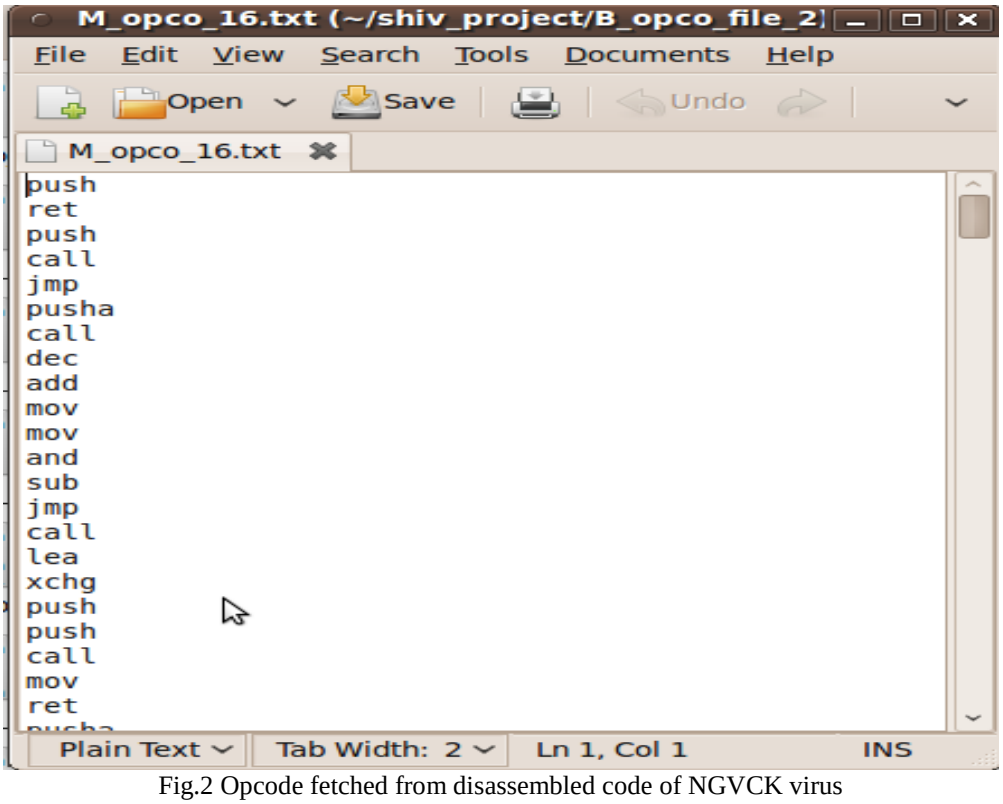


Fig.2 Opcode fetched from disassembled code of NGVCK virus

Module 3: *Call_Frequency_Counter (Fetched opcode)*

This is the most important part of our proposed model which has been utilized to count the frequency of each opcode in output of module 2. For making the implementation easier to understand we have been called this dictionary file as a file2 and output of module 2 has been called as a file 1. During the implementation of this module, we have fetched one opcode at a time from file 2 and calculate the frequency of that opcode in file 1. After that we have written that opcode and its corresponding frequency in file 1 into the file 3 as shown in figure 3. This process we have repeated for 160 times in order to get the frequency of each of the 160 opcodes in file 1.

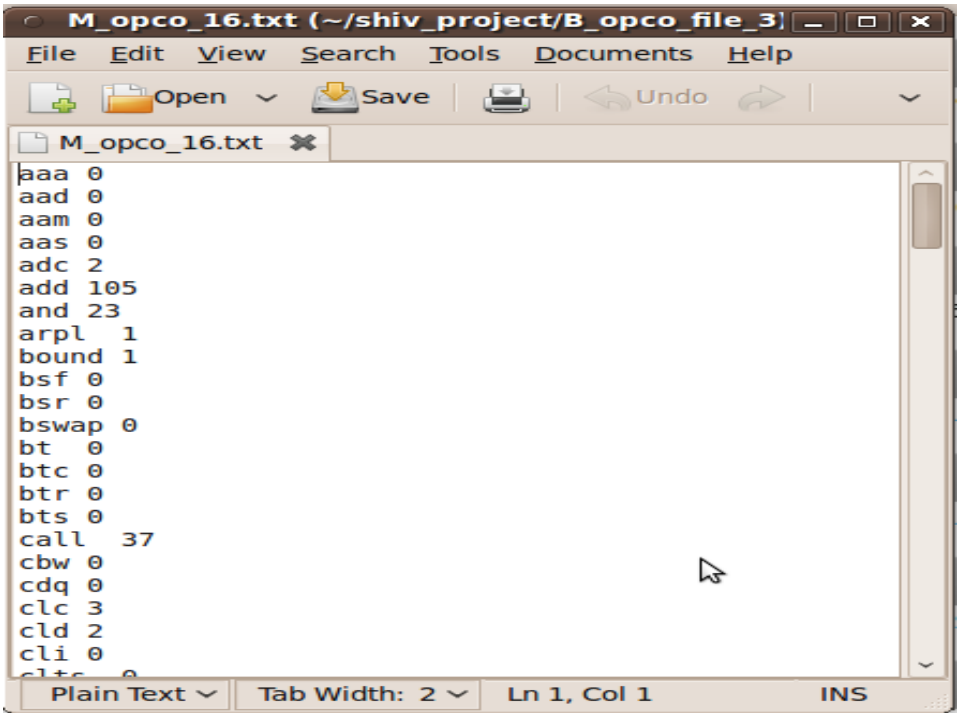
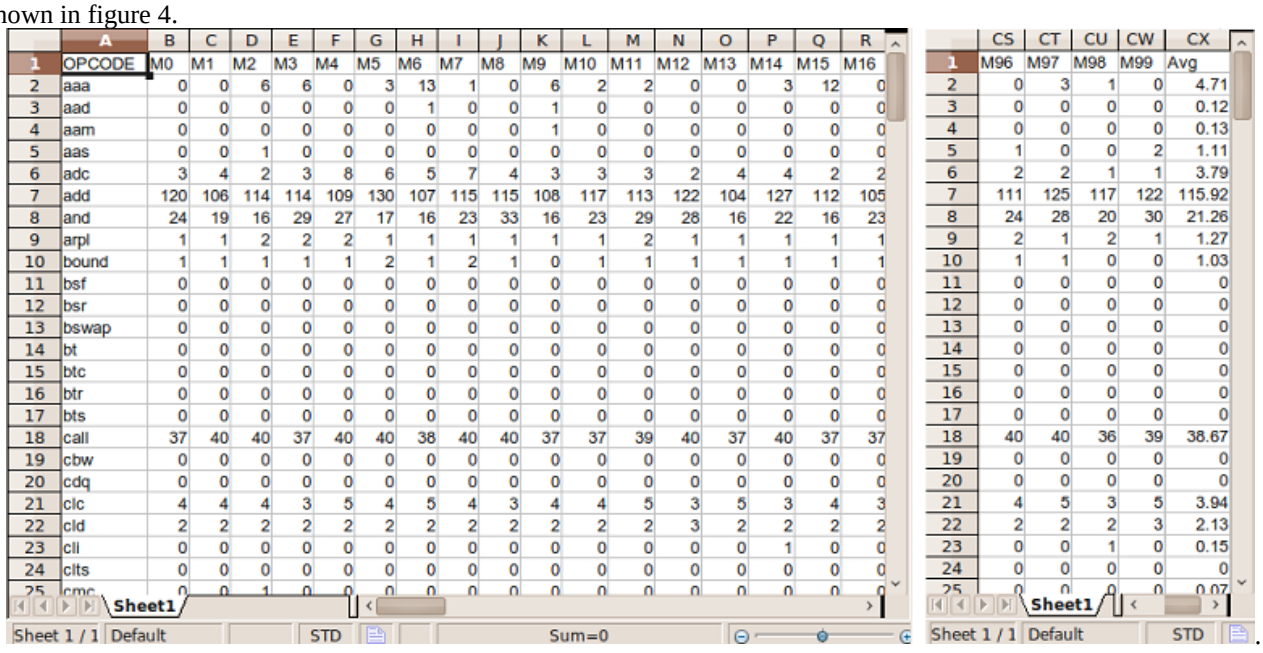


Fig. 3 Opcode frequency counting process in NGVCK virus

Module 4: *Call_Average_Opcode_ Frequency (family | Ngvck, Vcl32, Benign)*

In this module, first we have taken 100 virus samples of NGVCK then frequency of each of the 160 opcodes has been calculated in each of the virus file from ngvck_001 to ngvck_100. As an output of module 3, we are basically having 100 of such files after successful completion of module 3, where each file contains the opcodes and their corresponding frequency as counted in its respective file. For example, suppose op_freq_ngvck_001.txt is one of the outputs of module 3 which contains the opcodes and their corresponding frequency as counted in its respective virus file ngvck_001.

During the execution of module 4, we mapped all of the 100 virus files into a single CSV file such as ngvck.csv as shown in figure 4.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	OPCODE	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
2	aaa	0	0	6	6	0	3	13	1	0	6	2	2	0	0	3	12	0
3	aad	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
4	aam	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	aas	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	adc	3	4	2	3	8	6	5	7	4	3	3	3	2	4	4	2	2
7	add	120	106	114	114	109	130	107	115	115	108	117	113	122	104	127	112	105
8	and	24	19	16	29	27	17	16	23	33	16	23	29	28	16	22	16	23
9	arpl	1	1	2	2	2	1	1	1	1	1	1	2	1	1	1	1	1
10	bound	1	1	1	1	1	2	1	2	1	0	1	1	1	1	1	1	1
11	bsf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	bsr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	bswap	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	bt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	btc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	btr	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	bts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	call	37	40	40	37	40	40	38	40	40	37	37	39	40	37	40	37	37
19	cbw	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	cdq	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	clc	4	4	4	3	5	4	5	4	3	4	4	5	3	5	3	4	3
22	cld	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2
23	cli	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
24	clts	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	cmc	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	CS	CT	CU	CW	CX
1	M96	M97	M98	M99	Avg
2	0	3	1	0	4.71
3	0	0	0	0	0.12
4	0	0	0	0	0.13
5	1	0	0	2	1.11
6	2	2	1	1	3.79
7	111	125	117	122	115.92
8	24	28	20	30	21.26
9	2	1	2	1	1.27
10	1	1	0	0	1.03
11	0	0	0	0	0
12	0	0	0	0	0
13	0	0	0	0	0
14	0	0	0	0	0
15	0	0	0	0	0
16	0	0	0	0	0
17	0	0	0	0	0
18	40	40	36	39	38.67
19	0	0	0	0	0
20	0	0	0	0	0
21	4	5	3	5	3.94
22	2	2	2	3	2.13
23	0	0	1	0	0.15
24	0	0	0	0	0
25	0	0	0	0	0.07

Fig.4. Ngvck.csv file containing the average frequency of opcodes in a NGVCK virus family

Module 5: *Call_Average_Average_opcode_Frequency (Ngvck.csv, Vcl32.csv, Benign.csv)*

In this module, we have considered the output of module 4 means all the three CSV files named as ngvck.csv, vcl32.csv and benign.csv. During the execution of module 5, we have created new output file ($) as shown in figure 5. In this output file ($), all the 160 unique opcodes has been placed at the first column and second column has been assigned with the average frequency as appeared in the last column of ngvck.csv. Thus, third and fourth column have been assigned with the average frequencies as appeared in the last column of vcl32.csv and benign.csv. The fifth and last column of output file ($) has been utilized to compute the average of second, third and fourth columns of output file.

	A	B	C	D	E
1	OPCODE	ngvck	vcl32	benign	Average
2	aaa	4.71	0.01	5.24	3.32
3	aad	0.12	0.39	0.71	0.406667
4	aam	0.13	0	2.96	1.03
5	aas	1.11	0	9.49	3.533333
6	adc	3.79	1.58	20.21	8.526666
7	add	115.92	98.54	805.12	339.86
8	and	21.26	6.46	93.35	40.35667
9	arpl	1.27	2.37	31.98	11.87333
10	bound	1.03	0.95	9.31	3.763333
11	bsf	0	0	0	0
12	bsr	0	0	0.01	0.003333
13	bswap	0	0	0.04	0.013333
14	bt	0	0	0	0
15	btc	0	0	0	0
16	btr	0	0.26	0	0.086667
17	bts	0	0	0	0
18	call	38.67	44.68	221.24	101.53
19	cbw	0	0	0	0
20	cdq	0	0	0	0
21	clc	3.94	0.65	2.91	2.5
22	cld	2.13	0.26	1.92	1.436667
23	cli	0.15	0.2	2.39	0.913333
24	clts	0	0	0	0

M_feature_table_1 | Ready | 100%

Fig.5 Output file ($) containing the final average corresponding to average frequencies of different families

Average which is appeared in the last column as we can see in figure 5, provide the average of average frequencies associated with each family in respect of corresponding opcodes. After analysis of output file ($), we observed that only few opcodes among the all families contain the high amount of average values, while rest of the opcodes are having relatively very low average value. In order to sort the output file ($) in descending order of average values, we have passed it to module 6.

Module 6: *Call_sort_Opcodes_inrespectof_Average (($) having Ngvck, Vcl32, Benign, Average)*

In this process, we have taken the outcome of module 5 means output file ($) and sort the entire file in respect of last column named as average in the descending order of average vales as shown in figure 6.

	A	B	C	D	E
1	OPCODE	ngvck	vcl32	benign	Average
2	add	115.92	98.54	805.12	339.86
3	mov	123.8	56.64	564.07	248.17
4	push	62.6	88.41	547.41	232.8067
5	call	38.67	44.68	221.24	101.53
6	inc	15.81	16.23	186.38	72.80667
7	xor	67.61	24.93	121.66	71.4
8	pop	29.25	7.19	172.09	69.51
9	cmp	52.12	4.95	134.36	63.81
10	lea	10.99	8.92	116.44	45.45
11	dec	19.4	13.96	95.99	43.11667
12	and	21.26	6.46	93.35	40.35667
13	jmp	13.56	11.8	87.89	37.75
14	test	2.83	11.18	80.95	31.65333
15	sub	33.11	12.67	48.17	31.31667
16	ret	21.82	11.25	51.71	28.26
17	popa	14.33	13.86	49.18	25.79
18	imul	3.39	8.4	64.3	25.36333
19	or	4.03	1.73	31.83	12.53
20	arpl	1.27	2.37	31.98	11.87333
21	xchg	12.57	3.37	10.27	8.736667
22	adc	3.79	1.58	20.21	8.526666
23	pusha	9.05	8.53	3.78	7.12
24	sbb	1.92	1.06	16.8	6.593333

M_feature_table_1 | Average: 10.17564579 Count: 805 Sum: 6512.413307

Fig.6 Sorted output file ($) in respect of last column named as average

The output of module 6 has been utilized to identify the opcodes which are most frequently occurred in any portable executable file. In order to find the most frequently occurred opcodes, we have further transferred the outcome of module 6 to module 7.

Module 7: *Call_MFO_Fetcher (Opcodes, Average)*
This is last and final step of our proposed method, which has been used to extract the most frequently occurred (MFO) opcodes from our dataset of PE files. As per our analysis, we observed that top 17 opcodes as shown in figure 6 are the most frequently occurred opcodes. Figure 7 shows the average opcode distribution in any portable executable file.

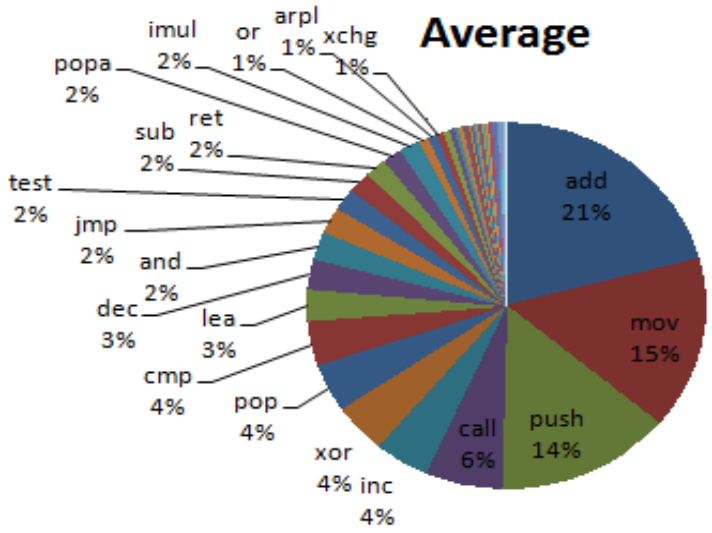


Fig.7 Average opcode distributions in portable executable files

After proper analysis of figure 7, we observed that only top 17 opcodes as shown in figure 6 played the major role in construction of any PE file. Among these 17 opcodes, each opcode occupied the average memory space in the range from 2% to 21% in the construction of any portable executable. These 17 opcodes occupied total 91% of memory space in any portable executable. Rest of the opcodes occupied total 9% of memory space where each opcode gained only 1% or less than 1% of memory space in any PE file. Therefore, these rare opcodes may be discarded during the development of malware detection algorithm in order to improve the performance of overall detection system.

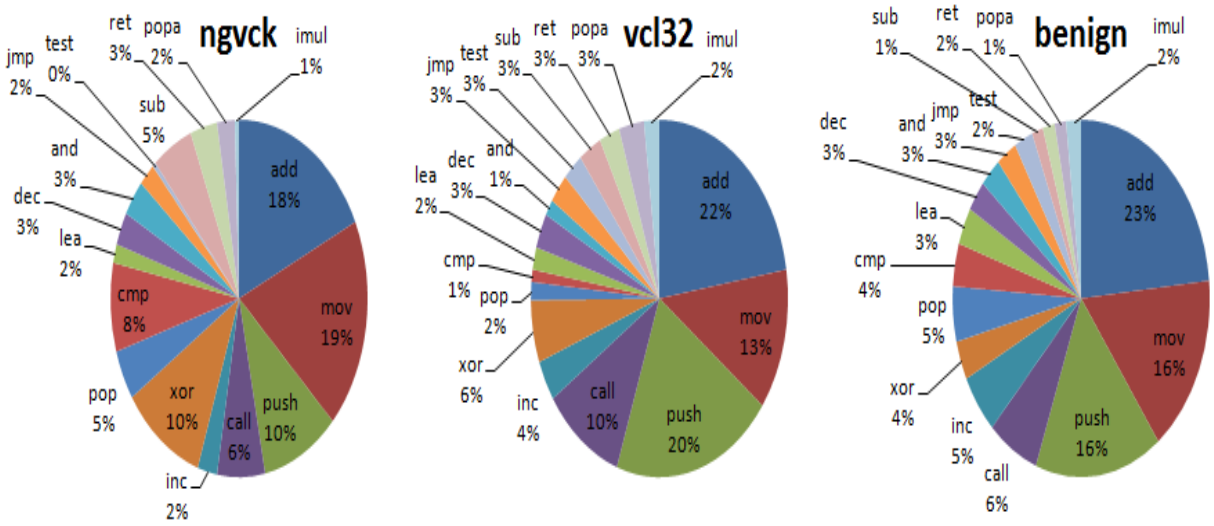
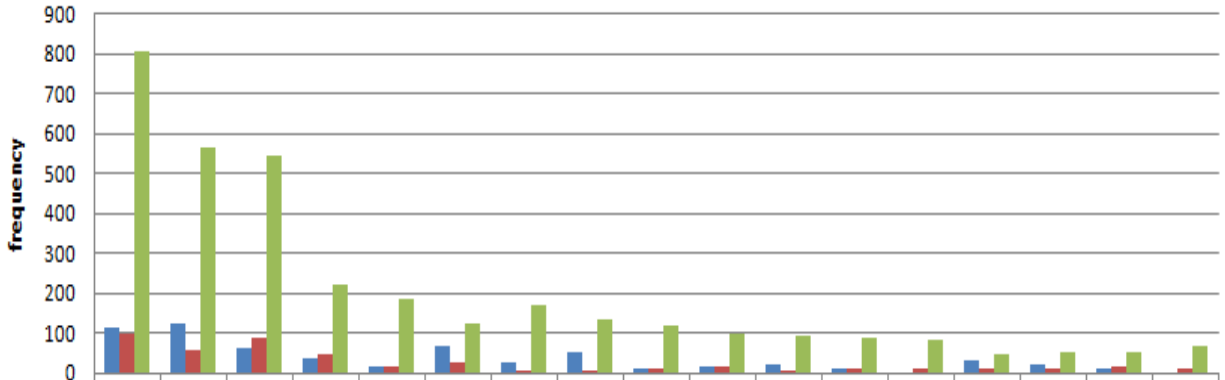


Fig.8 Frequency analysis of opcodes in different family of malware and benign samples

Figure 8 has been designed with the consideration of 17 MFO opcodes. It has been also observed in the figure 8 that only 4 opcodes such as add, mov push and call are consumed more than 50 % space in all families. These four opcodes can only be used in any detection method in order to improve the efficiency of detection system. But it may also produce the high amount of false alarms during the detection process. As being a malware researcher, it is our responsibility against the society to develop the effective and accurate system. It remains our prime focus to improve the detection rate of malware samples with very low false alarm. Therefore, in our proposed method we have considered all 17 opcodes as a MFO opcodes instead of only four opcodes as discussed above.

These 17 opcodes can be used as a predictor for the detection of metamorphic malware samples in any detection approach where opcodes have been considered as a prime element. With the help of comparative analysis of opcode frequencies as shown in figure 9, we have concluded that these 17 MFO opcodes are having relatively higher average frequency in benign samples. Therefore, it is easier for us to separate the malware samples from benign executables with the help of these 17 MFO opcodes. In figure 9, green color has been used to represent the frequency histogram of benign samples and the frequency variation of MFO opcodes is relatively high between malware and benign samples. It is clearly visible in the given figure that MFO opcodes provides the separation layer between malware and benign samples in terms of opcode frequency.



	add	mov	push	call	inc	xor	pop	cmp	lea	dec	and	jmp	test	sub	ret	popa	imul
ngvck	115.9	123.8	62.6	38.67	15.81	67.61	29.25	52.12	10.99	19.4	21.26	13.56	2.83	33.11	21.82	14.33	3.39
vcl32	98.54	56.64	88.41	44.68	16.23	24.93	7.19	4.95	8.92	13.96	6.46	11.8	11.18	12.67	11.25	13.86	8.4
benign	805.1	564.0	547.4	221.2	186.3	121.6	172.0	134.3	116.4	95.99	93.35	87.89	80.95	48.17	51.71	49.18	64.3

Fig. 9 Comparative analysis of opcode frequencies

As per the results shown in figure 9, the variation between the frequencies of MFO opcodes is very high among metamorphic malware and benign samples. Therefore, these 17 MFO opcodes may be further used in any malware detection approach in order to get the high detection rate with low false alarm. Although it is not possible to reduce the false alarm upto 0%. But, now we can say this confidently that with the proper utilization of these MFO opcodes we can prepare the better detection modal in the field of metamorphic malware research.

V. CONCLUSIONS

Malware is one of the most critical threats for computer systems. The numbers of malware are increasing day by day as the malware writers has been stated to employ the variety of obfuscation techniques in generation of new variant of known virus. In signature based method, signature database of known malware is maintained. Thus, it is not capable to deal with unseen malware which are created during the execution of known malware like metamorphic malware. In this research work, we have identified some unique features such as MFO opcodes which can be further utilized to improve the detection rate of metamorphic malware with relatively higher efficiency. As per the results shown in section 4, we concluded that these MFO opcodes can be used as a predictor for metamorphic malware detection. The variation in frequencies of these MFO opcodes was found relatively very high among metamorphic malware and benign samples as shown in figure 8. In literature survey, we studied various detection techniques for the detection of metamorphic malware. In all the existing techniques, researchers had been used entire sequence of opcodes as appeared in disassembly code of virus executables to detect the variants of metamorphic malware. Therefore, performance of their detection models was very low.

ACKNOWLEDGMENT

This research work has been carried out with the support of R&D cell, Department of computer science and engineering, Arya college of engineering and I. T., Jaipur. We thank to all researchers in field of metamorphic malware to get motivated and encouraged to do further work in this field.

REFERENCES

[1] Md.Enamul Karim, Andrew Walenstein, and Arun Lakhotia (2005) Malware Phylogeny Generation using Permutations of Code. Journal in Computer Virology, 1(1–2):13–23.

[2] V. Sai Sathyanarayan, Pankaj Kohli, and Bezawada Bruhadeshwar, "Signature Generation and Detection of Malware Families", in Proceedings of the 13th Australasian conference on Information Security and Privacy (ACISP '08), Springer-Verlag, Berlin, Heidelberg, 2008.

[3] P. Vinod, V. Laxmi, M.S. Gaur, GVSS. Phani Kumar, and Y.S. Chundawat, "Metamorphic virus detections through static code analysis" 2009. Retrieved on 11 September, 2012 from website: http://amrita.edu/cyber-workshop/proceedings/icscf09_submission_54.pdf.

[4] Qinghua Zhang and Douglas S. Reeves. MetaAware: Identifying Metamorphic Malware. *Computer Security Applications Conference, Annual*, 0:411–420, 2007.
[5] Mark Stamp Wing Wong. Hunting for Metamorphic Engines. 2006.
[6] Aditya Govindaraju," Exhaustive statistical analysis for detection of metamorphic malware", Master's thesis, 2010. Retrieved on 11 September, 2013 from website: http://scholarworks.sjsu.edu/etd_projects/66.
[7] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. Architecture of a Morphological Malware Detector. *Computer Virology*, pages 263–270, 2009.
[8] Matthieu Kaczmarek Guillaume Bonfante and Jean-Yves Marion. Control Flow Graphs as Malware Signatures. 2007.
[9] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic Worm Detection using Structural Information of Executables. In *In RAID*, pages 207–226. Springer-Verlag, 2005.
[10] Heejo Lee Kyoochang Jeong. Code Graph for Malware Detection. In *In International conference on Information Networking,ICOIN*, pages 1–5. IEEE, 2008.
[11] Griffin, Kent and Schneider, Scott and Hu, Xin and Chiueh, Tzi-Cker, "Automatic Generation of String Signatures for Malware Detection", Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection Springer-Verlag, Berlin, Heidelberg, 2009.
[12] A W. Wong and M. Stamp, "Hunting for Metamorphic Engines", Journal in Computer Virology, vol. 2, no. 3, pp. 211-229, Dec. 2006.
[13] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious patterns", Proceedings of the 12th conference on USENIX Security Symposium, Vol. 1, pp. 169-186, Aug. 2003.
[14] W. Wong, "Analysis and detection of metamorphic computer viruses", Retrieved on 28 August, 2013 from website: http://www.cs.sjsu.edu/faculty/stamp/students/Report.pdf.
[15] F. Cohen, " Computer viruses: theory and experiments", Comput. Secur., 6(1):22-35, February 1987.
[16] Lin, Da and Stamp, Mark, Hunting for undetectable metamorphic viruses, In Journal Computer Virology, volume (7), issue (3), pp. 201–214, August, 2011,

