San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

1-1-2010

DETECTING UNDETECTABLE COMPUTER VIRUSES

Sujandharan Venkatachalam
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Venkatachalam, Sujandharan, "DETECTING UNDETECTABLE COMPUTER VIRUSES" (2010). *Master's Projects*. Paper 156.

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

DETECTING UNDETECTABLE COMPUTER VIRUSES

A Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science in Computer Science

by

Sujandharan Venkatachalam

May 2010

© 2010

Sujandharan Venkatachalam

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Thesis Committee Approves the Project Titled

DETECTING UNDETECTABLE COMPUTER VIRUSES

by

Sujandharan Venkatachalam

_____

Dr. Mark Stamp, Department of Computer Science Date

_____

Dr. Robert Chun, Department of Computer Science Date

_____

Mr. Manikandan Alagappan, Cisco Systems Inc. Date

APPROVED FOR THE UNIVERSITY

_____

Associate Dean Office of Graduate Studies and Research Date

ABSTRACT

Signature-based detection relies on patterns present in viruses and provides a relatively simple and efficient method for detecting known viruses. At present, most anti-virus systems rely primarily on signature detection.

Metamorphic viruses are one of the most difficult types of viruses to detect. Such viruses change their internal structure, which provides an effective means of evading signature detection. Previous work has provided a rigorous proof that a fairly simple metamorphic engine can generate viruses that will evade any signature-based detection.

In this project, we first implement a metamorphic engine that is provably undetectable—in the sense of signature-based detection. We then show that, as expected, the resulting viruses are not detected by popular commercial anti-virus scanners. Finally, we analyze the same set of viruses using a previously developed approach based on hidden Markov models (HMM). This HMM-based technique easily detects the viruses.

ACKNOWLEDGEMENTS

I would like to thank my project advisor Dr. Mark Stamp for his guidance and insights throughout the project. I would also like to thank my committee members – Dr. Robert Chun and Mr. Manikandan Alagappan for providing me with their valuable feedback.

I would also like to thank my parents and friends for their support and encouragement throughout the Masters program.

Table of Contents

1 Introduction 1

2 Evolution of Viruses 2

2.1 Early stages 2

2.2 Stealth Viruses 3

2.3 Polymorphic Viruses 3

2.4 Metamorphic Viruses 5

3 Intrusion Detection Systems 7

3.1 Signature based Intrusion detection 7

3.2 Anomaly based detection 8

3.3 Emulation based detection 8

4 Code obfuscation techniques 9

4.1 Garbage Instructions 10

4.2 Instruction Reordering 10

4.3 Subroutine Reordering 11

4.4 Interchangeable Instructions 11

4.5 Swapping of registers 12

5 Virus detection using machine learning techniques 14

5.1 Neural networks 14

5.2 Data mining techniques 14

5.3 Hidden Markov models 15

5.4 Training the model 15

6 Generation of Metamorphic Viruses 16

6.1 Implementation method 16

6.2 Training the HMM Model 19

6.3 Testing the HMM Model 20

7 Experiment Setup and results 21

7.1 Experimental Setup 21

7.2 Creation of Seed Virus 22

7.3 Creation of Metamorphic Viruses 23

7.4 Testing Signature Detection 23

7.5 Testing using HMM Model 26

8 Conclusion and Future Work 27

References 29

Appendix A: HMM Scores for different Metamorphic Viruses 31

Appendix B : Scatter plot of HMM Scores of Metamorphic viruses 37

Appendix C : Garbage Instructions used 40

List of Figures

FIGURE 1: GENERATIONS OF A POLYMORPHIC VIRUS [17] 4
FIGURE 2: GENERATIONS OF A METAMORPHIC VIRUS [17] 6
FIGURE 3: SEARCH PATTERN FOR STONED VIRUS [18] 8
FIGURE 4: CODE REORDERING [18] 11
FIGURE 5: CODE OBFUSCATION IN OUR ENGINE 18
FIGURE 6: TEST DATA PREPARATION 19
FIGURE 7: TRAINING A HMM MODEL 20
FIGURE 8: TESTING SEED VIRUS WITH MCAFEE ANTIVIRUS 22
FIGURE 9: SAMPLE VERSIONS GENERATED BY CODE OBFUSCATION ENGINE 23
FIGURE 10: SCANNING THE GENERATED METAMORPHIC VIRUSES USING MCAFEE 24
FIGURE 11: SCANNING THE GENERATED METAMORPHIC VIRUSES USING AVAST 25
FIGURE 12: LOG FILE GENERATED BY AVAST 25
FIGURE 13: HMM SCORES GRAPH 26
FIGURE 14: NGVCK FAMILY VIRUSES WITH 2 HIDDEN STATES 37
FIGURE 15: G2 FAMILY VIRUSES WITH 2 HIDDEN STATES 37
FIGURE 16: VCL32 FAMILY VIRUSES WITH 2 HIDDEN STATES 38
FIGURE 17: NGVCK FAMILY VIRUSES WITH 3 HIDDEN STATES 38
FIGURE 18: G2 FAMILY VIRUSES WITH 3 HIDDEN STATES 39
FIGURE 19: VCL32 FAMILY VIRUSES WITH 3 HIDDEN STATES 39

List of Tables

TABLE 1: VIRUS DETECTION TECHNIQUES [2] **9**
TABLE 2: CODE OBFUSCATION TECHNIQUES IN DIFFERENT VIRUSES [4] **10**
TABLE 3: CODE OBFUSCATION OF NGVCK VIRUS [6] **13**
TABLE 4: EXPERIMENTAL SETUP **21**
TABLE 5: HMM SCORES FOR NGVCK VIRUSES WITH 2 HIDDEN STATES **31**
TABLE 6: HMM SCORES FOR G2 VIRUSES WITH 2 HIDDEN STATES **32**
TABLE 7: HMM SCORES FOR VCL32 VIRUSES WITH 2 HIDDEN STATES **33**
TABLE 8: HMM SCORES FOR NGVCK VIRUSES WITH 3 HIDDEN STATES **34**
TABLE 9: HMM SCORES FOR G2 VIRUSES WITH 3 HIDDEN STATES **35**
TABLE 10: HMM SCORES FOR VCL32 FAMILY WITH 3 HIDDEN STATES **36**

1 Introduction

Malware are programs that infect a machine and perform malicious actions on that machine. Viruses, worms and trojan horses are some typical categories of malware. The first PC virus, called Brain, was created in 1986; it infected the boot sector of the storage media [21]. Since the creation of Brain, virus creation and detection methodologies have evolved considerably [21]. Virus creators try to evade popular detection mechanisms. Once a virus bypassed an anti-virus system and infects a large number of computers, virus detection mechanisms are generally updated to prevent further infections.

Virus programmers have created several types of viruses that attempt to bypass certain detection systems [21]. Polymorphic viruses spread to host machines after encrypting their code with unique keys to hide their signature. In contrast, metamorphic viruses change their signature by altering their code [15]. The ultimate goal of an antivirus program is to develop a security mechanism that is strong enough to detect any virus and prevent further infection. However, the detection mechanisms currently available do not detect all types of viruses. In particular, metamorphic viruses are difficult to detect. In this project, we implement a metamorphic technique that evades any signature-based detection system. We then show that a machine learning technique can efficiently detect these types of viruses.

2 Evolution of Viruses

2.1 Early stages

During the early stages of virus creation, virus programmers tried to infect a large number of victims throughout the world. Viruses created were similar in their type of infection, but the malicious actions performed were different. Viruses were created to corrupt the disk system, email accounts, private networks, etc. The methods used to infect a host machine and spread to other machines were similar for all these viruses. Virus detection systems attempted to detect the infections based on the signature files and actions performed by viruses. Most of the early stage viruses were detected based on their signatures. As virus detection systems detected and stopped the infections with increasing strength, virus programmers started implementing new methods for creating viruses and spreading the infections.

During recent years, the number of malicious programs has grown rapidly. According to security experts, the number of viruses will be more than a million before the end of this decade [16]. Even though there the number of viruses has drastically increased, patches and removal tools for most of the viruses have been created immediately after their appearance. In addition, the infections spread by these malware are decreasing. For example, the amount of infections spread by email attachment viruses has reduced from one in 40 to one in 1000 in the last 10 years. This decrease in infections is due to efficient intrusion detection systems [16]. On a similar note, virus writers are developing new techniques to create and spread viruses without being detected. Thus, an intelligent intrusion detection system is necessary to handle different types of viruses.

2.2 Stealth Viruses

Stealth viruses are the first step taken by virus programmers to elude virus detection systems. These types of viruses take control of the file management system and conceal the changes it has made to the infected files. Due to the stealth nature, these types of viruses reside in memory and hide from virus detection systems. These viruses corrupt the files and sometimes encrypt the data present in the files [21]. When virus scanners attempt to scan the files, viruses redirect them to the proper data location and avoid detection. In order to detect such memory resident viruses, virus detection systems were built so that the active memory could be scanned for infections before the files were scanned. These types of detection systems worked more effectively when executed from a compact disc or floppy disk, which is write-protected. Brain, Frodo and Whale are some of the popular stealth viruses [21].

2.3 Polymorphic Viruses

Polymorphic viruses try to bypass virus detection systems by mutating themselves through self-encryption [5]. The code encryption implemented in polymorphic viruses hides the signature of virus files. The code is encrypted using different keys for the victim host machines. The decryption engine is attached in the code itself, which will then decrypt the code and execute the virus. This type of viruses is harder to detect since signature is hidden using encryption. Figure 1 illustrates different variants of a typical polymorphic virus.

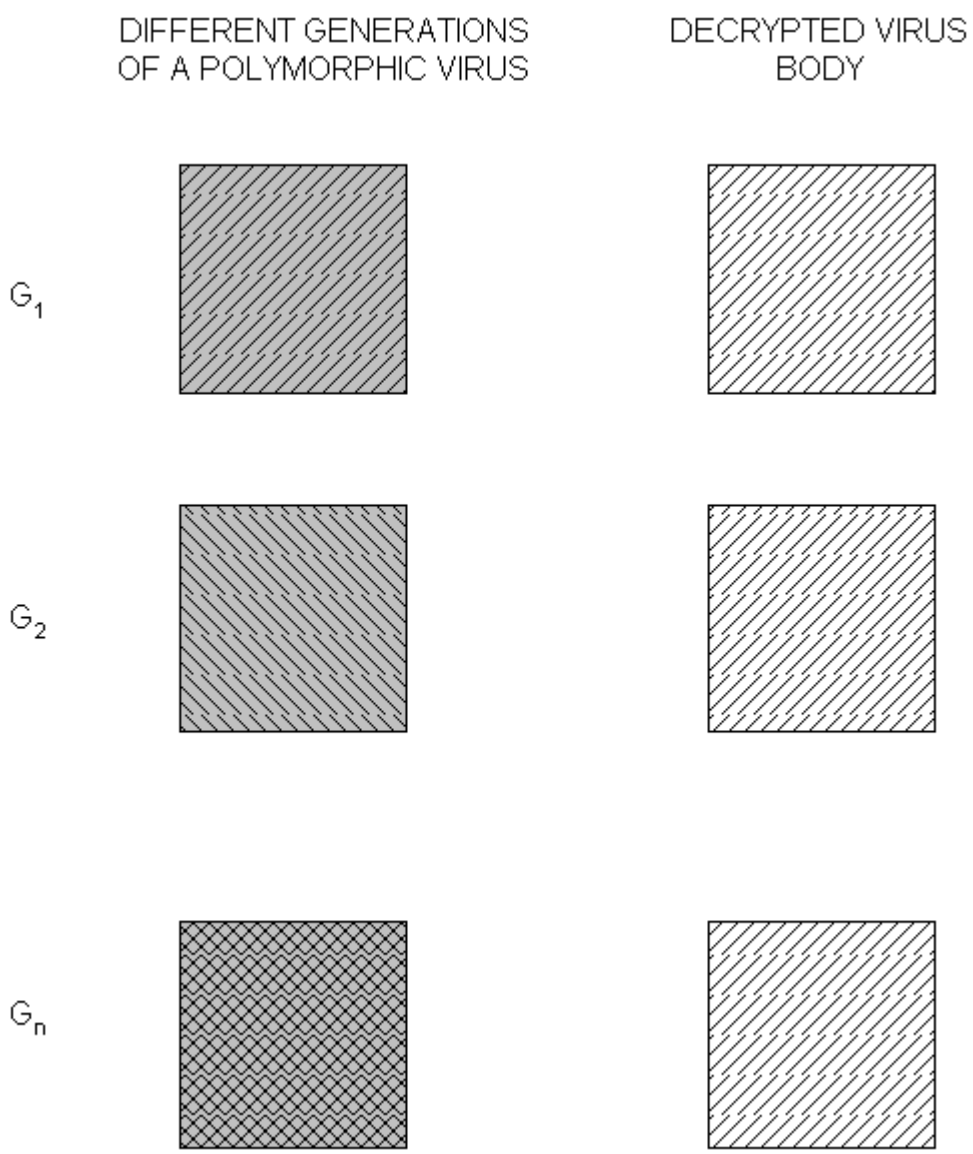


Figure 1: Generations of a Polymorphic Virus [17]

Although the code is encrypted, the decryption engine used is the same. In order to bypass the detection systems, these viruses use multiple encryption schemes and carry different decryption engines [17]. While decrypting the code, one of the decryption engines is used. So this type of viruses can be possibly detected using emulation based detection systems [2]. The virus named 1260 was the first polymorphic virus; it paved the way to the creation of sophisticated polymorphic viruses with different techniques for encryption and decryption.

2.4 Metamorphic Viruses

Metamorphic viruses modify their code to produce an equivalent one during propagation [3]. These viruses attempt to evade detection through static analysis by implementing code obfuscation techniques. Such techniques implemented are swapping interchangeable instructions, inserting garbage instructions and introducing conditional jumps to produce the child virus [9]. The child virus will basically do the same function but will have a different signature. In this method, the signature of a virus is broken by changing the order of instructions without altering the control flow. A sophisticated type of this virus will generate code based on the host's operating system by translating the instructions to the corresponding machine code [8]. Figure 2 illustrates different variants of a typical metamorphic virus.

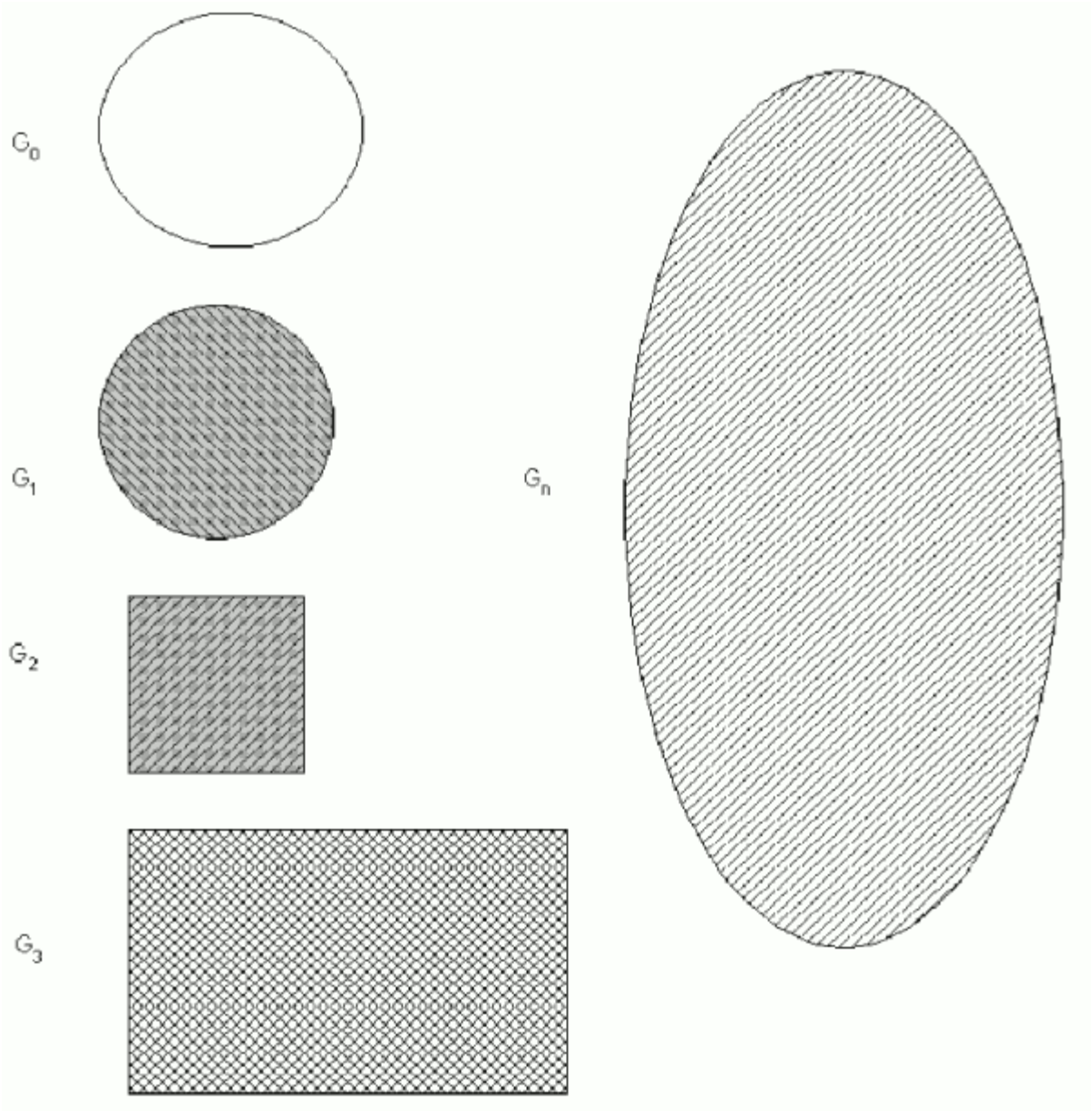


Figure 2: Generations of a Metamorphic Virus [17]

The detection of these viruses using their signature is challenging since the signature is broken in each version of the virus. In order to detect such metamorphic viruses, the detection system should be designed to extract the essential instructions of the virus from virus instance. This extracted instruction set should be used to detect the viruses of that type [9].

3 Virus Detection Systems

3.1 Signature based virus detection

Signature based detection systems scan the files for specific signatures that are present in them. The pattern of instructions present in a virus code is identified as the signature of the virus file. This will raise an alarm for virus if the signature of a virus is detected in any of the files scanned. This method of intrusion detection is fast and accurate since the chances of false alarms are very low in this system. The main requirement of the system is to have an updated database of all the signature files of malware. The accuracy is totally dependent on the signature database of the system. Signature based detection systems cannot detect a new virus since the database will not have any information about the new virus.

An antivirus scanner extracts the opcode pattern from an executable file and searches the signature database for the input opcode pattern. The input opcode pattern is considered as the signature of the input file. If a match is found in the signature database, the input file is classified as the corresponding virus family matched in the signature database. For example, if the signature of the input file is 83EB 0274 EB0E 740A 81EB 0301 0000, then this will be searched in the signature database and the file will be classified as W32/Beast virus since 83EB 0274 EB0E 740A 81EB 0301 0000 is the signature of the W32/Beast virus [18]. A similar search pattern used to detect Stoned virus is shown in Figure 3 [18].

```
seg000:7C40 BE 04 00                    mov     si, 4           ; Try it 4 times
seg000:7C40                                                     ;
seg000:7C43
seg000:7C43                 next:                               ; CODE XREF: sub_7C3A+27↓j
seg000:7C43 B8 01 02                    mov     ax, 201h        ; read one sector
seg000:7C46 0E                          push    cs
seg000:7C47 07                          pop     es
seg000:7C48                             assume es:seg000
seg000:7C48 BB 00 02                    mov     bx, 200h        ; to here
seg000:7C4B 33 C9                       xor     cx, cx
seg000:7C4D 8B D1                       mov     dx, cx
seg000:7C4F 41                          inc     cx
seg000:7C50 9C                          pushf
seg000:7C51 2E FF 1E 09 00              call    dword ptr cs:9  ; int 13
seg000:7C56 73 0E                       jnb     short fine
seg000:7C58 33 C0                       xor     ax, ax
seg000:7C5A 9C                          pushf
seg000:7C5B 2E FF 1E 09 00              call    dword ptr cs:9  ; int 13
seg000:7C60 4E                          dec     si
seg000:7C61 75 E0                       jnz     short next
seg000:7C63 EB 35                       jmp     short giveup
```

Figure 3: Search Pattern for Stoned virus [18]

3.2 Anomaly based virus detection

Anomaly based detection systems monitor the processes on a host machine for any abnormal activity. If any abnormal activity is identified, the system raises an alarm signaling the possible presence of malware [15]. In this detection technique, the system uses the collected heuristics to categorize an activity as normal or malicious. Even though chances of false alarm are relatively higher in this method, it is more reliable because it is also capable of detecting new viruses. The important thing to note is that raising a false alarm is not as potential harmful as allowing a new virus. However, these systems can be trained gradually by intruders to consider abnormal behavior as normal. Thus, system will fail to detect the abnormal activity in such cases [15].

3.3 Emulation based detection

The emulation based detection is an effective method where a virus is executed in a virtual environment by emulating the instructions in the virus code. This type of detection is used to detect polymorphic, as well as metamorphic, viruses. The virus instance can be executed in the

virtual environment in order to identify instruction sequence or behavior of the virus [21]. In addition to the virtual environment, code optimization techniques can be applied to the execution process to decrease the time for detection. Table 1 lists the strength and weakness of these detection methods.

Table 1: Virus Detection Techniques [2]

Detection technique	**Strength**	**Weakness**
Signature based	Efficient	New malware
Anomaly based	New malware	Costly to implement, False Positives, unproven
Emulation based	Encrypted viruses	Costly to implement

4 Code obfuscation techniques

Metamorphic viruses use one or more code obfuscation techniques to produce different metamorphic versions of the same virus. The obfuscation techniques are used in this method to break the signatures of the virus files. Therefore, most of the virus programmers will implement as many as possible obfuscation techniques to bypass the intrusion detection systems. In some cases, the obfuscation techniques implemented may help the detection systems detect the viruses. This is due to the excess amount of obfuscation implemented rather than the obfuscation required to bypass the detection system. The code obfuscation techniques implemented in various viruses are shown in Table 2.

Table 2: Code obfuscation techniques in different viruses [4]

	Evol (2000)	Zmist (2001)	Zperm (2000)	Regswap (2000)	MetaPHOR (2001)
Instruction Substitution				✓	
Instruction Permutation	✓	✓			✓
Garbage code Insertion	✓	✓			✓
Variable Substitution	✓	✓		✓	✓
Altering Control Flow		✓	✓		✓

4.1 Garbage Instructions

Inserting garbage instructions like NOP instructions or opaque predicates in between the actual code blocks is a simple obfuscation technique used in all of the virus generators. These garbage instructions will not alter the functionality of the code but will increase the size of the code. Viruses that contain garbage instructions are hard to detect using the signatures since these instructions break the signature of the virus. The garbage instructions should be inserted within a threshold value. If the number of garbage instructions is high, the intrusion detection systems can easily detect the abnormality in the code. In our code obfuscation engine, the garbage instructions are inserted at random with a threshold value. Also, the instructions inserted between the blocks are not similar.

4.2 Instruction Reordering

In this method, the instructions in the virus code are reordered in a random fashion and control flow is adjusted to make it execute in the same order. This is accomplished by providing labels for each reorder and then using conditional jump instructions to jump the control to the labels. Thus, the instructions are reordered inside the code without altering the control flow. This method of obfuscation is well known for bypassing signature detection since it changes the order

of opcode sequence. The instructions are reordered in such a way that it does not introduce too many jump instructions. If too many jump instructions are inserted, the intrusion detection systems may detect the abnormal behavior and report that as malware. Figure 4 shows an example of code reordering

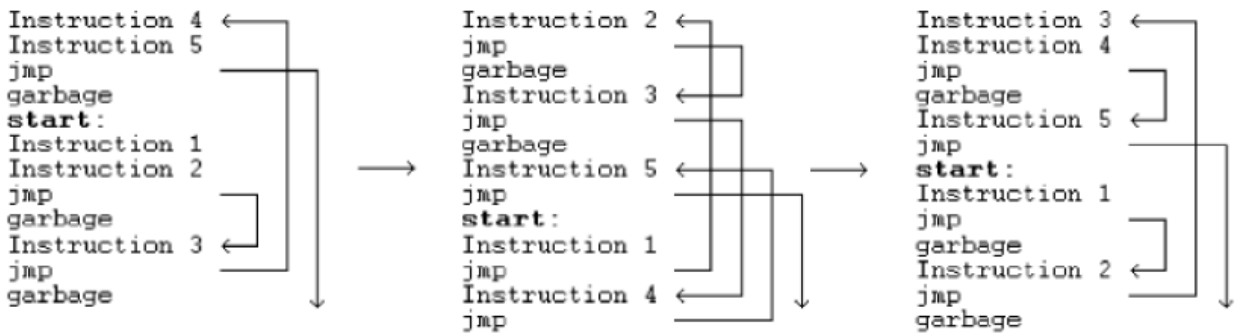


Figure 4: Code Reordering [18]

4.3 Subroutine Reordering

In this method, the subroutines in the virus code are reordered without changing the control flow of the virus code. This method is similar to instruction reordering where the subroutines are reordered using labels and conditional jump instructions.

4.4 Interchangeable Instructions

In this method, the instructions that have many equivalent instructions performing the same operation are replaced by one of its equivalent instructions. This introduces a smaller amount of metamorphism since the opcode pattern is changed due to this method. Thus, the metamorphic versions of a virus will have a different pattern of opcode but perform the same functionalities. This method is only successful for signature detection systems because it totally detects the viruses based on the opcode pattern. Also, the obfuscation introduced through this

method is not permanent. When the assembled executables of different virus forms, which uses this obfuscation method, are disassembled using any disassembler, these obfuscation are removed.

4.5 Swapping of registers

This is a similar method to Interchangeable instructions; instead of replacing instructions, the registers are replaced with equivalent registers. The underling idea is same in both the methods, which try to change the opcode pattern and bypass the signature detection. This technique was used in the generation of W95/Regswap virus [18]. Table 3 provides an example of code obfuscation used by the Next Generation Virus Construction Kit (NGVCK) [21].

Table 3: Code obfuscation of NGVCK [6]

Basic Version	**Morphed Version 1 (Code Reordering)**	**Morphed Version 2 (Garbage Insertion)**

Call Delta
 	
Delta: pop ebp
 	
Sub ebp, offset Delta
	Call Delta
	
	Delta: sub dword ptr[esp], offset Delta
	
	Pop eax
	
	Mov ebp, eax
		Add ecx, 0031751B ; junk
		
		Call Delta
		
		Delta: sub dword ptr[esp], offset Delta
		
		Sub ebx,00000909 ; junk
		
		Mov edx, [esp]
		
		Xchg ecx, eax ; junk
		
		Add esp, 00000004
		
		And ecx, 00005E44 ; junk
		
		Xchg edx, ebp

HEX equivalent: E8000000005D81ED05104000	HEX equivalent: E800000000812C2405104000588BE8	HEX equivalent: *812C240B104000*8B1424*83C404*87EA

5 Virus detection using machine learning techniques

Machine learning techniques can be applied to detect metamorphic viruses since they can be used to detect patterns between the generations of a specific family of virus. These detected patterns from a training model that can be used to test any input instance for similarities.

5.1 Neural networks

The Neural networks can be implemented in detecting viruses that possess a specific set of features. Initially, the features of a virus should be analyzed and the networks should be trained based on the features [7]. Then, the network model can be used to identify viruses that contain most of the features present in the model. These network models are well known for detecting viruses that are not in the same family as the training model, but possess some of the malicious features from the training model.

The efficiency of these network models also depends upon the threshold values for the minimum number of features to be present in a test file. A higher threshold value trains the network model to detect viruses only from the specific virus family whereas a lower threshold value results in higher false positive rates. This detection technique was implemented in IBM Antivirus program to detect boot sector viruses. The program was able to detect the boot sector viruses efficiently with a very low false positive rate [7]. This is because the scanner was able to cover most of the features of boot sector viruses in the network model.

5.2 Data mining techniques

Most data mining techniques are rule based methods that train the models with a set of rules about the functionality of the viruses. The training model classifies the test files based on

the rules covered by those files [22]. Once again, the chance of false positives is high in this case. However, this technique is used widely for pattern detection in a large set of data. Researchers have shown that the data mining techniques produce effective results when multiple data mining models are combined.

5.3 Hidden Markov models

Hidden Markov models (HMM) are statistical models used to analyze and understand a Markov process and provide a result based on a series of observations related to the process. This is a state machine based model, which completely relies on the current state and does not consider the past states. This model is used to take decisions based on a process using the observations that are obtained as input to the model. However, the underlying process is always hidden in this model and the observations and results are only visible to the outside world. It is demonstrated in [22] that hidden Markov models (HMMs) could be used for detection of metamorphic viruses.

5.4 Training the model

The HMM can be trained for a particular model using the observations and state transitions from a training set. Once the model is trained on a training data set, it is able to detect the similar patterns from any set of observations and make state transitions according to that. In this project, we train a HMM model with the observations for a particular metamorphic virus family. Once the model is trained, it is able to detect the metamorphic viruses using the similarities between the opcode patterns.

6 Generating Metamorphic Viruses

In this project, a metamorphic virus generator is implemented in Perl, which satisfies the conditions specified in [4]. This engine generates metamorphic versions of a seed virus, which is given as input. It also implements code obfuscation techniques, like instruction reordering and garbage insertion, to produce the metamorphic versions of a virus.

6.1 Implementation method

The input virus code is split into smaller blocks of code and then reordered using conditional jump instructions and labels. The number of instructions in each block is set to a variable and it is initially set to six. The virus code is split into blocks based on conditions provided in [4]. The code blocks should not end with a jump instruction or a NOP instruction. In addition to that, the entire virus code should be present in the code section of the assembly file. Viruses, which contain a part of the code in the data section, could not be given as input to the generator. After splitting the code into smaller blocks of code, the blocks are randomly shuffled. Then, labels are placed for each block of code and the control flow is then maintained by placing conditional jump instructions for each block. The code obfuscation techniques implemented in the generator are instruction reordering and garbage insertion. The overall process of the code obfuscation process is shown in Figure 5.

The low level description of the functions performed by the code obfuscation engine is:

- Any valid instruction present in the assembly file is identified
- Block generator generated the blocks based on the following specific conditions.

- The first and last block of the code should not be changed.
- The last instruction of the block could not be a label, JMP and NOP
- Insert garbage instructions within a minimum threshold value. Also, the inserted instructions should not have affected the original virus code.
- The block numbers are generated using permutation at random.
- Then, the output file is written with the code blocks written in the order computed through permutation

The garbage insertion is implemented in the generator as an optional element. The amount of garbage instructions inserted could be controlled using a threshold value. Based on the threshold value selected, the garbage instructions, such as dummy copy instructions and opaque predicates, are inserted in between each pair of code blocks. The garbage instructions are inserted into the virus code after the blocks shuffling is done. Since the amount and content of obfuscation is varied every time for each generation of a virus, the metamorphic form generated has a different signature every time. The generator has been tested with virus families like NGVCK, Phalcon Skism G2 and PS-MPC.

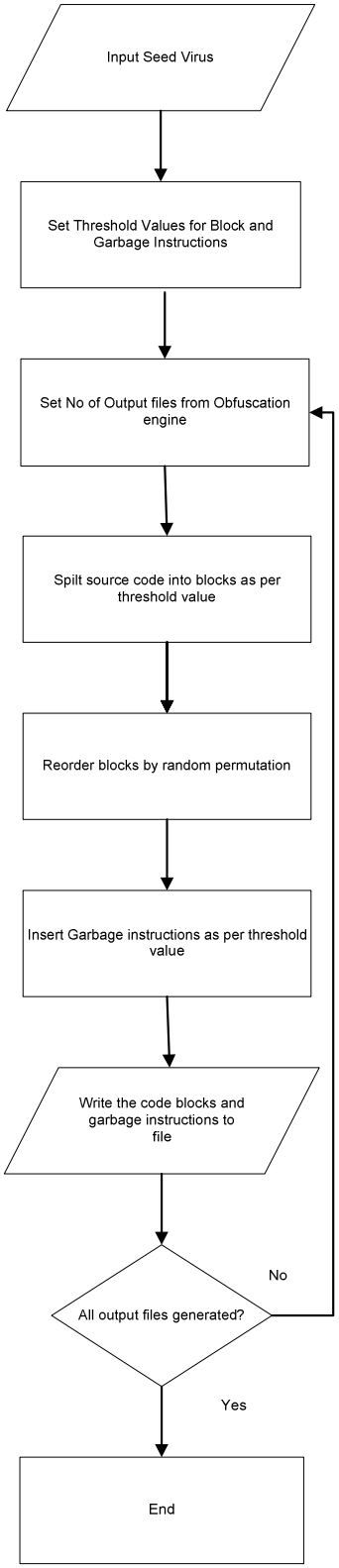


Figure 5: Code Obfuscation in our engine

6.2 Training the HMM Model

The HMM Engine developed for [22] is used for testing our implementation. In order to train the HMM engine, 200 different versions of a seed virus are created using the code obfuscation engine. The files created by the code obfuscation engine are ASM files with same functionality but different signatures. These 200 files are assembled using Borland Turbo TASM 5.0 assembler to produce corresponding OBJ and MAP files. Then, Borland Turbo TLINK 7.1 linker is used to produce EXE files from the OBJ files. The EXE files obtained in the previous step are disassembled using IDA Pro disassembler and the corresponding ASM files are produced. The steps performed in preparing the test data is shown in Figure 6.

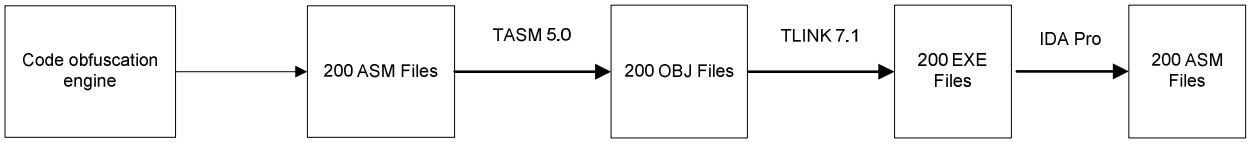


Figure 6: Test Data Preparation

Among the 200 ASM files, only 160 files are used for Training and the rest 40 ASM files are used for testing the HMM model. Instead of using the ASM files generated by the code obfuscation engine, the disassembled ASM files obtained from IDA Pro are used for final testing. This increased the efficiency of the comparison and removed the coding style discrepancies between the source ASM files [22]. The steps involved in training a HMM model is shown in Figure 7.

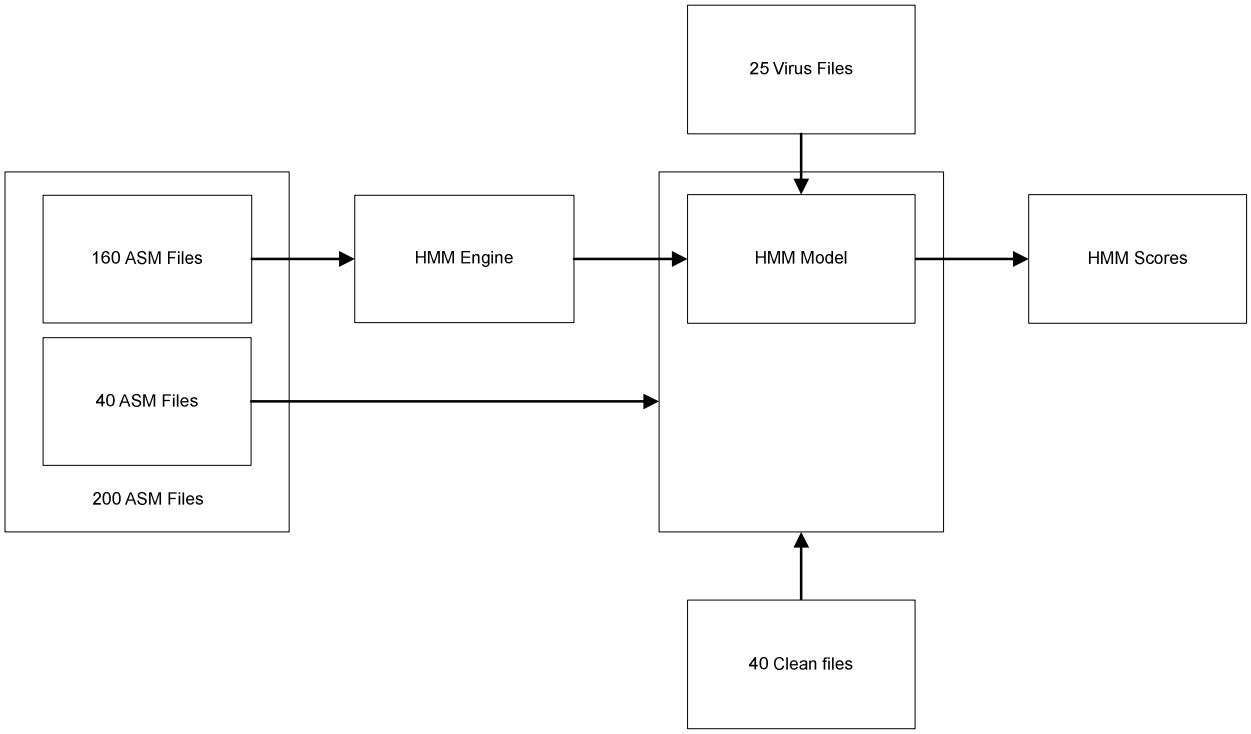


Figure 7: Training a HMM Model

6.3 Testing the HMM Model

The HMM engine is tested using the 40 ASM files that remained in the set of 200 files disassembled using IDA Pro. In addition to these 40 files, the test set also includes 25 other family viruses and 40 clean files. The other family viruses are included in the test case to ensure that scores between the family viruses and other viruses are different. We perform a k-fold cross validation with the data set provided as input to test the HMM model. It splits the input data set with 200 files into 5 equal sets. Among these five sets, four sets of files are used for training the model and one set is used to test the trained HMM model.

7 Experiment Setup and results

We analyzed viruses generated using different virus generators like MPCGEN (Phalcon/Skism Mass Code Generator), G2 (Generation 2 Virus Generator), VCL32 (Virus Creation Lab for Win32) and NGVCK (Next Generation Virus Creation Kit). In each test case, the popular anti-virus scanners could not detect the generated virus files. We were able to successfully bypass the signature detection, but, irrespective of the seed viruses that were used, HMM engine was able to detect the viruses effectively. In addition to that, the HMM engine was able to clearly distinguish between virus files and normal files.

7.1 Experimental Setup

Virus creation, analysis and testing were executed using the setup listed in Table 4.

Table 4: Experimental Setup

Experiment platform	Windows XP VMware virtual machine
Programming language	Perl5
Disassembler	OllyDbg v1.10 IDA Pro 4.9
Assembler	Borland Turbo Assembler 5.0
Linker	Borland Turbo Linker 7.1
Virus generator	MPCGEN (Phalcon/Skism Mass Code Generator) G2 (Generation 2 Virus Generator) VCL32 (Virus Creation Lab for Win32) NGVCK (Next Generation Virus Creation Kit)
Virus scanners	Avast Home Edition 4.8 McAfee Antivirus 2009

7.2 Creation of Seed Virus

The seed virus, which was given as the input to the code obfuscation engine, was created using a virus construction kit. Virus generators that we used for this implementation were MPCGEN (Phalcon/Skism Mass Code Generator), G2 (Generation 2 Virus Generator), VCL32 (Virus Creation Lab for Win32) and NGVCK (Next Generation Virus Creation Kit). These generators were downloaded from the vxheaven website. Each constructor had specific instructions and options to create a seed virus. Seed viruses were created following the instructions given by the virus construction kits. When the created ASM file of the seed virus was compiled, the anti-virus scanners detected the executables as the corresponding virus. This test was done to ensure that the anti-virus scanner used for testing the obfuscated files was able to detect the seed virus. The screenshot of the security alert displayed as soon as the seed virus is compiled is shown in Figure 8.

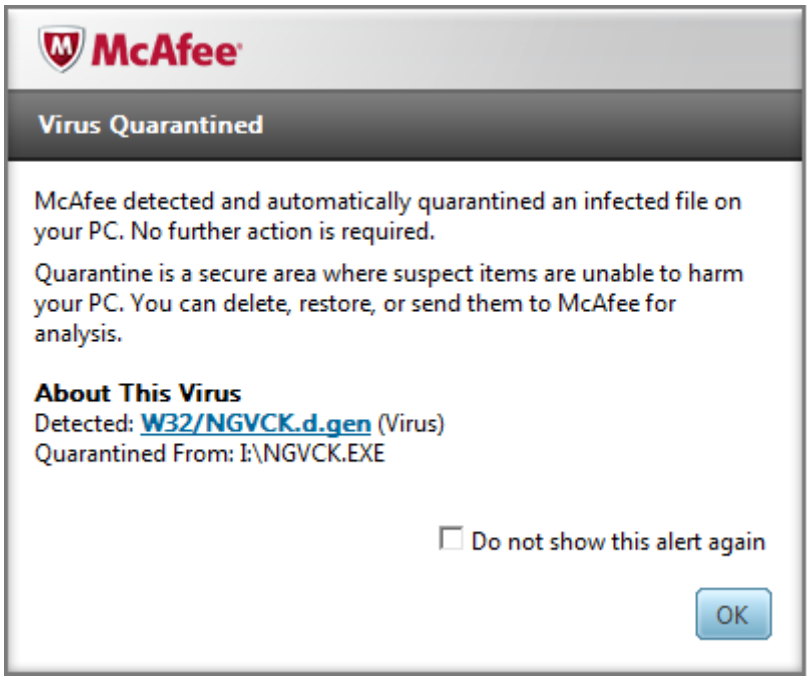


Figure 8: Testing Seed virus with McAfee Antivirus

7.3 Creation of Metamorphic Viruses

The metamorphic variants of the seed virus were created using our code obfuscation engine. The parameters were set to generate 200 different variants of seed virus with a threshold of two garbage instructions. The screenshots of two variants of NGVCK seed virus is shown in Figure 9.

ngvck98.asm

```
; Win32.NGVCK by SnakeByte
;
; This Virus is created with
; the Next Generation VCK by SnakeByte
; to get a copy of this Kit
; check www.kryptocrew.de/snakebyte/

.586p
.model flat
jumps
.radix 16
 extrn ExitProcess:PROC
.data
 VirusSize equ (offset EndVirus - offset Virus )
 NumberOfApis equ 10d

.code
start:
VirusCode:
Virus:
jmp labelblock1
add    bx, 0
shl    cx, 0


labelblock22:
 add eax, -1
 inc eax
 jz UnMapFile
 mov dword ptr [ebp+MapAddress], eax
 clc
ret
UnMapFile: ; Unmap the file and store it to disk
 Call UnMapFile2
CloseFile: ; Close the file
 push dword ptr [ebp+FileHandle]
jmp labelblock23
shr    bx, 0
test    cx, 0

labelblock56:
Notagoodfile:
```

ngvck141.asm

```
; Win32.NGVCK by SnakeByte
;
; This Virus is created with
; the Next Generation VCK by SnakeByte
; to get a copy of this Kit
; check www.kryptocrew.de/snakebyte/

.586p
.model flat
jumps
.radix 16
 extrn ExitProcess:PROC
.data
 VirusSize equ (offset EndVirus - offset Virus )
 NumberOfApis equ 10d

.code
start:
VirusCode:
Virus:
jmp labelblock1
xor    bx, 0
shr    cx, 0


labelblock50:
 push dword ptr [ebp+MapAddress]
 pop ebx
 mov ebx, [ebx+3Ch]
 add ebx, dword ptr [ebp+MapAddress]
 mov edx, dword ptr [ebp+CheckSum]
 mov dword ptr [ebx+58h], edx
NoCheckSum:
 mov ecx, dword ptr [ebp+InfCounter]
 add ecx, -1
jmp labelblock51
or    cx, 0
add    cx, 0


labelblock58:
```

Figure 9: Sample versions generated by Code Obfuscation Engine

7.4 Testing Signature Detection

Viruses created using the code obfuscation engine were assembled and compiled using TASM and TLINK to produce executables of the viruses. These viruses were scanned using

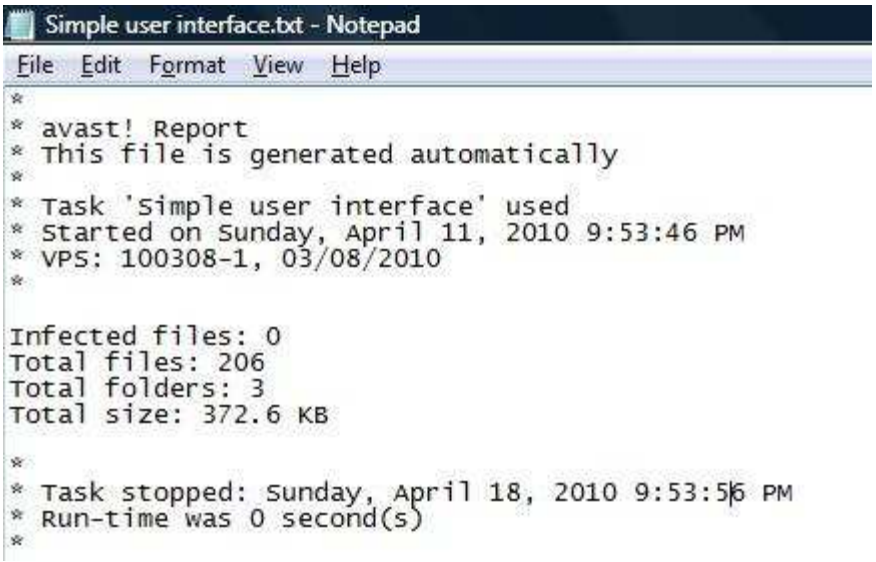
popular anti-virus scanners like Avast and McAfee. The post-scan summary is shown in Figure 10 and Figure 11.

Figure 10: Scanning the generated metamorphic viruses using McAfee

These scanners were not able detect these executables as viruses since the signature was totally broken with the help of code obfuscation engine.

Figure 11: Scanning the generated metamorphic viruses using Avast

The log file generated by Avast antivirus after the scan is also provided. The log file did not have any information about malicious executables. The screenshot of the log file is shown in Figure 12.



Simple user interface.txt - Notepad

File Edit Format View Help

```
*
* avast! Report
* This file is generated automatically
*
* Task 'Simple user interface' used
* Started on Sunday, April 11, 2010 9:53:46 PM
* VPS: 100308-1, 03/08/2010
*

Infected files: 0
Total files: 206
Total folders: 3
Total size: 372.6 KB

*
* Task stopped: Sunday, April 18, 2010 9:53:56 PM
* Run-time was 0 second(s)
*
```

Figure 12: Log file generated by Avast

7.5 Testing using HMM Model

A HMM model was trained using the viruses generated and then tested against variants. The executables generated to test the antivirus scanners were disassembled using IDA Pro, and the ASM files produced were given as the input for the HMM Model. Then, we perform a k-fold validation with 800 iterations and 5 sets of 40 viruses each. Among the five sets, four sets were used for training the model and one set was used for testing the model. The number of observation symbols was in the range from 40 to 42 and the total number of observations ranged from 41472 to 42151. The HMM score graph is shown in Figure 13.

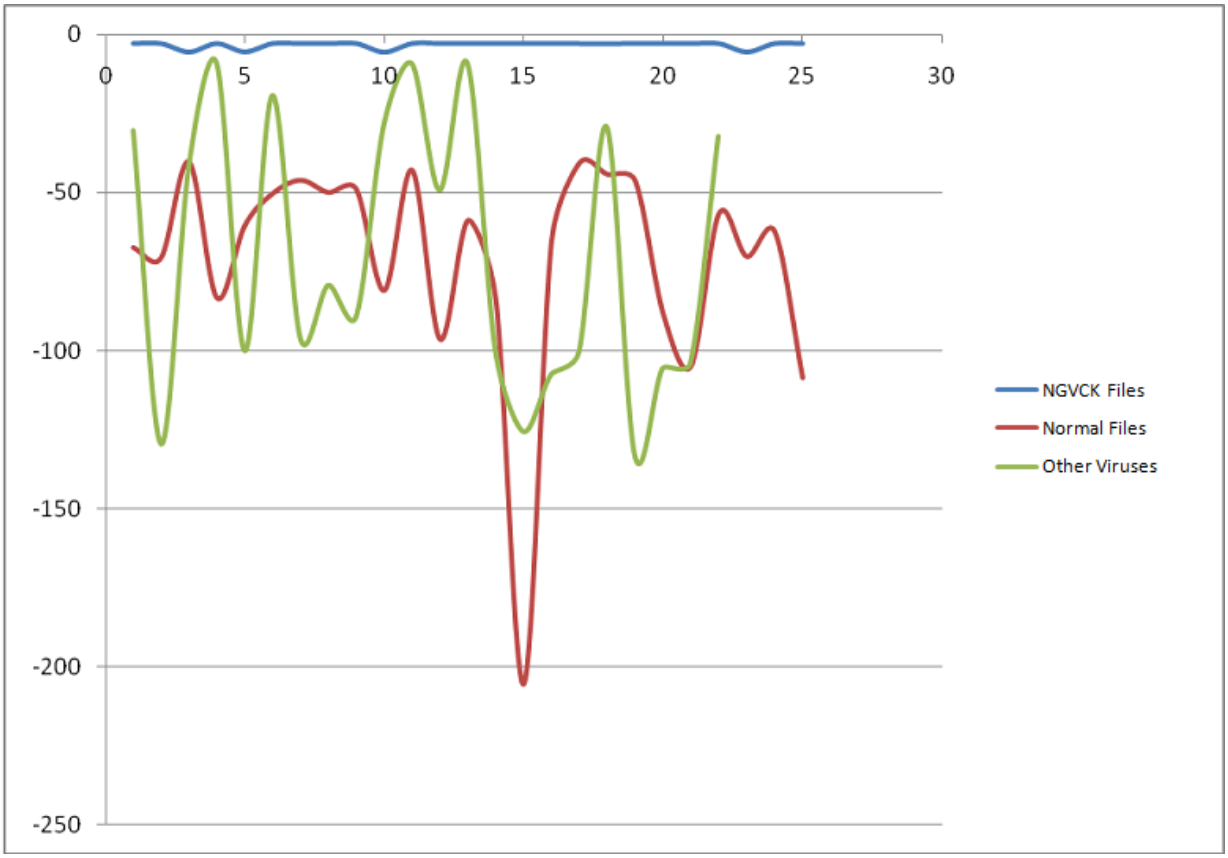


Figure 13: HMM Scores graph

8 Conclusions and Future Work

This project's main goal is to show that the metamorphic viruses generated satisfying the conditions in [4] will bypass the signature detection systems. This is due to the code obfuscation techniques implemented in generating the metamorphic viruses. The second goal of the project is to prove that the machine learning methods are effective in detecting these metamorphic viruses. In our second phase, a HMM model is trained using datasets from different metamorphic viruses and then used to detect the metamorphic viruses generated using the code obfuscation engine developed in the previous phase.

We performed a five-fold cross validation by dividing the data set containing 200 viruses into five equal sets. Among these five sets, four sets were used for training the HMM model and the excluded set was used to test the model. Since it follows five-fold cross validation, five different models were generated and tested for efficient results. Finally, we were able to conclude that metamorphic viruses generated by following the conditions in [4] successfully evaded signature detection. The experiment results clearly showed that HMM models were able to detect these metamorphic viruses efficiently.

In this project, we obfuscated the code by inserting garbage instructions and shuffling the code blocks without altering the control flow. The HMM model was able to detect the opcode patterns in these viruses even after obfuscation. This implementation can be improved further by strengthening the code obfuscation process. The techniques currently used by metamorphic generators are not producing variants that challenge HMM models. The obfuscation process should be able to replace one or more instructions with a different set of equivalent instructions

performing same functions. As a result, viruses will contain different opcode sequences which might be challenging to detect by an HMM model.

The disassembly process implemented in our implementation takes considerable amount of time to prepare the input data files for HMM detection. Virus executables were disassembled using IDA Pro disassembler. Due to this, the time taken for the detection process is relatively more than signature detection. This time factor can be reduced by designing a disassembler that can speed up this process by extracting the opcodes from the raw binary file. If a faster disassembly process is implemented, the HMM models can be used to analyze a large set of virus variants and increase the efficiency of the detection.

References

[1] Avast Antivirus, http://www.avast.com/

[2] S. Attaluri, "Profile hidden Markov models for metamorphic virus analysis," Master's thesis, San Jose State University, 2007. http://www.cs.sjsu.edu/faculty/stamp/students/Srilatha_cs298Report.pdf

[3] "Benny/29A", Theme: metamorphism, http://www.vx.netlux.org/lib/static/vdat/epmetam2.htm

[4] J. Borello and L. Me, "Code Obfuscation Techniques for Metamorphic Viruses", Feb 2008, http://www.springerlink.com/content/233883w3r2652537

[5] P. Desai, "Towards an undetectable Computer Virus," Master's thesis, San Jose State University, 2008. http://www.cs.sjsu.edu/faculty/stamp/students/Desai_Priti.pdf

[6] J. Dickinson, "The New Anti-Virus Formula," Messaging News Press 2005. http://www.ironport.com/pdf/ironport_new_anti-virus_formula.pdf

[7] IBM Corporation. (1996). "Neural Networks for Computer Virus Recognition", Retrieved April 10, 2010, from http://www.research.ibm.com/antivirus/SciPapers/Tesauro/NeuralNets.html

[8] IDA Pro, http://www.hex-rays.com/idapro/

[9] E. Konstantinou, "Metamorphic Virus: Analysis and Detection," January 2008.

[10] A. Lakhotia, "Are metamorphic viruses really invincible?" Virus Bulletin, December 2005.

[11] P. Mishra, "A taxonomy of software uniqueness transformations", master's thesis, San Jose State University, Dec. 2003. http://home.earthlink.net/~mstamp1/mss_v.html#masters

[12] Orr, "The molecular virology of Lexotan32: Metamorphism illustrated," 2007. http://www.antilife.org/files/Lexo32.pdf

[13] Orr, "The viral Darwinism of W32.Evol: An in-depth analysis of a metamorphic engine," 2006. http://www.antilife.org/files/Evol.pdf

[14] M. Stamp, "A Revealing Introduction to Hidden Markov Models", January 2004. http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf

[15] M. Stamp, “Information Security: Principles and Practice,” August 2005.

[16] PCWorld. (2008). “Viruses Expected to Hit 1 Million This Year”, Retrieved April 10, 2010, from http://www.pcworld.com/article/144181/viruses_expected_to_hit_1_million_this_year.html

[17] P. Szor, P. Ferrie, “Hunting for Metamorphic”, *Symantec Security Response*. http://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf

[18] P. Szor, “The Art of Computer Virus Defense and Research,” Symantec Press 2005.

[19] A. Venkatesan, “Code Obfuscation and Metamoprhic Virus Detection,” Master’s thesis, San Jose State University, 2008.

[20] virus-scan-software.com, “A history of computer viruses”, Retrieved April 10, 2010, from http://www.virus-scan-software.com/virus-scan-help/answers/the-history-of-computer-viruses.shtml

[21] VX Heavens, http://vx.netlux.org/

[22] W. Wong, “Analysis and Detection of Metamorphic Computer Viruses,” Master’s thesis, San Jose State University, 2006. http://www.cs.sjsu.edu/faculty/stamp/students/Report.pdf

Appendix A: HMM Scores for different Metamorphic Viruses

Table 5: HMM Scores for NGVCK viruses with 2 hidden states

Scores for N=2, E=2		
NGVCK Files	**Other Viruses**	**Clean Files**
-2.795638257	-20.46668687	-51.84229625
-2.783737261	-22.8211953	-49.4776919
-2.809481701	-11.16723377	-51.804289
-2.928199381	-18.3846972	-58.59448795
-2.907143378	-14.58048271	-48.46444149
-2.792977033	-9.25559972	-58.13471737
-2.775864355	-8.526595463	-52.36134393
-2.8068171	-8.792757183	-61.3422692
-2.754450714	-9.306025624	-51.98709398
-2.798528503	-19.65062141	-48.72447636
-2.738060282	-7.265326069	-52.20651307
-2.783705431	-22.09065629	-49.04912269
-2.819092612	-41.67066789	-54.80200888
-4.113166106	-24.57961899	-37.80392716
-2.82903651	-136.2418747	-48.92127872
-4.094386148	-24.91196494	-35.49476913
-2.840284302	-6.019265607	-31.160086
-4.120369757	-10.25663359	-28.18345636
-2.979024953	-13.29981934	-24.43932797
-2.843236131	-27.07212734	-18.06110462
-2.785269053	-13.85353246	-26.77224016
-4.120656337	-15.95424393	-36.33806748
-2.78361351	-14.21023592	-29.28831
-2.880753103	-165.5174174	-28.31494963
-2.799422536	-22.17690111	-27.95208466

Table 6: HMM Scores for G2 Viruses with 2 hidden states

Scores for N=2, E=2		
G2 Files	**Other Viruses**	**Clean Files**
-2.6205159877823	-12.247479337036	-79.3246785755614
-8.14358690086442	-12.52469131331	-72.3660647401549
-5.29921223638555	-12.2371350698591	-45.2833432338538
-5.33984797517399	-12.3395445993703	-98.5463407281103
-2.77163421228652	-12.2053575787154	-68.3676481825592
-2.69751590170883	-12.2661003783605	-65.792285489484
-5.31314280233542	-12.3763461918607	-60.8845373911266
-2.6225655576533	-12.2767774008446	-64.272050389521
-2.63490499260176	-12.3080193181927	-64.6159635854055
-2.60489468633888	-12.2390868089399	-94.7139778335376
-2.78102147379221	-30.5389142645068	-52.7411147142203
-5.27369096934477	-41.7918420024093	-110.895614908444
-5.17290328774662	-32.0468219009278	-67.3343902315585
-2.6129652227626	-41.7250977343909	-97.576830457571
-2.6077090593913	-41.7575871948567	-217.670451016029
-2.60210306154853	-149.410439234569	-72.4819527258201
-2.71668352886356	-115.370956892922	-47.0889318731106
-2.60418566989924	-143.695969073837	-52.6878228513769
-2.60312602923405	-122.587785196554	-61.0664067943091
-2.63671453886313	-126.641841223169	-106.900592123753
-2.62495871119516	-144.243401596867	-110.476467465273
-2.60824523950797	-150.510480232401	-68.111454708464
-5.27739038276747	-125.567588260126	-78.9971111427349
-2.60075307530066	-122.644856638973	-70.3635893069172
-7.73824444386762	-122.498343946442	-121.342886830277

Table 7: HMM Scores for VCL32 Viruses with 2 hidden states

Scores for N=2, E=2		
VCL32 Files	**Other Viruses**	**Clean Files**
-4.1162788731	-19.502997645	-48.772653031
-4.0175265289	-22.903897592	-55.902548530
-2.6718335612	-13.679756870	-55.010666852
-4.0307656472	-15.231064754	-55.546251322
-3.9672703530	-14.060639809	-51.651700252
-4.1582169640	-8.7586332574	-55.164027849
-2.6720330259	-8.1451491072	-52.674444498
-2.9287837414	-8.4114654383	-55.286429670
-2.8336680018	-9.3660634918	-55.388481768
-2.8099083309	-31.608940127	-51.962286419
-2.7015367881	-12.787001180	-39.128085227
-2.6970031402	-22.138698721	-45.985012886
-2.6929261977	-5.6818496988	-51.353038658
-4.0113219950	-24.531140414	-28.752621669
-2.8414693261	-136.69748000	-45.774960653
-4.0203959733	-24.813590014	-37.463445148
-3.9193982077	-6.1474440361	-33.555714719
-2.7313382287	-10.342876898	-32.014717622
-4.0028881737	-13.370665761	-28.829339625
-2.8473225096	-28.785972370	-20.709775458
-3.9728352294	-13.334668869	-30.190934638
-2.8338993514	-16.668048301	-38.325225235
-2.7079095463	-13.592827479	-31.185974714
-2.8368426056	-6.9932035030	-30.258308408
-2.7919918355	-25.660516810	-29.812347974

Table 8: HMM Scores for NGVCK Viruses with 3 hidden states

Scores for N=3, E=2		
NGVCK Files	**Other Viruses**	**Clean Files**
-2.5938059773	-42.111747613	-19.224023406
-2.7487193121	-45.812091511	-22.551070159
-3.8978454345	-48.382720132	-13.378005975
-2.6828385775	-48.830503255	-22.135848931
-2.5970854097	-45.040852114	-18.653927844
-2.5920816226	-48.484005015	-9.5675136651
-2.6341866218	-48.997107204	-9.3021469562
-2.5935327091	-51.693663345	-10.107856622
-2.5828247462	-48.607347225	-10.762110353
-2.5940426088	-45.235810697	-29.932178343
-2.7070912310	-38.584737858	-13.432962248
-3.9092215344	-42.324778822	-29.680686705
-3.9005588072	-47.348863022	-6.1157958200
-2.6264436357	-28.139849698	-24.361091642
-2.5724594386	-42.149674862	-142.20417996
-2.5886148009	-35.230399918	-24.551830026
-2.6118651992	-30.923391407	-7.3312967500
-2.5817540054	-27.907348624	-10.022793812
-2.6290215015	-24.178519958	-13.114092089
-2.5920166114	-17.618252880	-43.150421721
-2.6221344733	-26.524258848	-12.111914355
-2.5587086996	-36.166836989	-16.513617074
-2.5868409797	-29.096822543	-14.578562992
-3.9069214000	-28.110741912	-11.636799346
-3.9459464585	-27.674360449	-26.768453122

Table 9: HMM Scores for G2 Viruses with 3 hidden states

Scores for N=3, E=2		
G2 Files	**Other Viruses**	**Clean Files**
-2.521844850	-9.0315411858	-70.53248125
-2.539329293	-12.336743061	-84.34477031
-2.539620458	-9.0339657877	-46.09604099
-2.560674651	-9.0908540169	-95.64308908
-2.533625571	-9.0281052133	-69.48808074
-5.235913553	-9.0542863222	-53.61526851
-2.535453650	-12.339828126	-50.03673783
-2.550721665	-12.260743188	-57.23544111
-2.547813507	-12.274105877	-51.75387924
-2.548059971	-9.0381307106	-82.45604016
-2.516284482	-30.206718334	-47.15222029
-2.548689623	-41.199766140	-101.3738618
-2.539588016	-28.097114204	-72.44001522
-2.512748109	-45.809802167	-86.28219741
-5.216968093	-41.044651643	-208.8949451
-2.522537069	-137.42779013	-71.31213083
-2.511034608	-110.18986124	-47.75647831
-5.225479206	-130.70579353	-50.21575890
-2.524738831	-113.47012747	-58.11210863
-2.507610660	-106.15910082	-94.01584709
-2.543425779	-132.59716236	-112.0842567
-2.528832979	-140.72759942	-67.23492483
-2.523484644	-119.46452946	-77.44992692
-2.549455456	-118.38014241	-64.15727682
-2.524322434	-114.31042862	-112.7658265

Table 10: HMM Scores for VCL32 Family with 3 hidden states

Scores for N=3, E=2		
VCL32 Files	**Other Viruses**	**Clean Files**
-2.4505796246	-18.5074314470	-71.774981951
-5.1346394281	-15.4651412631	-74.160822211
-5.1798024193	-12.0938250457	-42.018605632
-5.0901421107	-15.4105235928	-97.497172276
-7.7144216044	-27.9761474186	-69.103957339
-5.0373129583	-21.7148233595	-76.590319606
-2.4562123917	-12.2503494857	-75.540041967
-2.4188462886	-15.2978819038	-80.063503900
-2.4203878681	-21.8123140650	-74.932663496
-2.4335440110	-15.3117210899	-97.188939719
-2.4380260251	-44.2342997034	-46.222516254
-2.4536110216	-50.9008694756	-107.26019358
-2.4461638872	-39.1034482625	-63.125749204
-5.0176314001	-55.5124377950	-92.400695342
-5.0452703920	-53.8868826395	-212.48915338
-2.4566510976	-141.576226480	-68.450851322
-5.1607640361	-115.439544161	-45.498376604
-2.4159925105	-139.878408178	-51.654185612
-5.1374100666	-124.102824933	-60.413640532
-2.4307716445	-115.086911528	-113.25886743
-5.0314336171	-140.828825106	-108.26150071
-5.0362169363	-142.836616588	-65.860880381
-2.4311948001	-123.627192792	-78.379537030
-2.4373640352	-122.727992078	-82.862764026
-2.4392003480	-120.594278439	-118.89145742

Appendix B : Scatter plot of HMM Scores of Metamorphic viruses

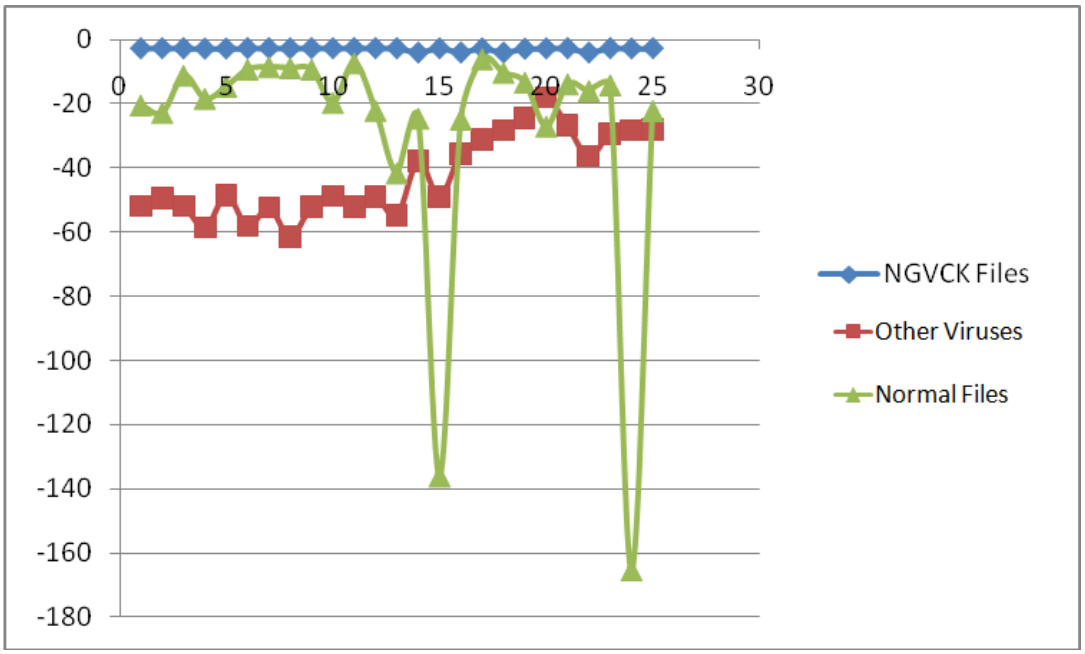


Figure 14: NGVCK Family Viruses with 2 hidden states

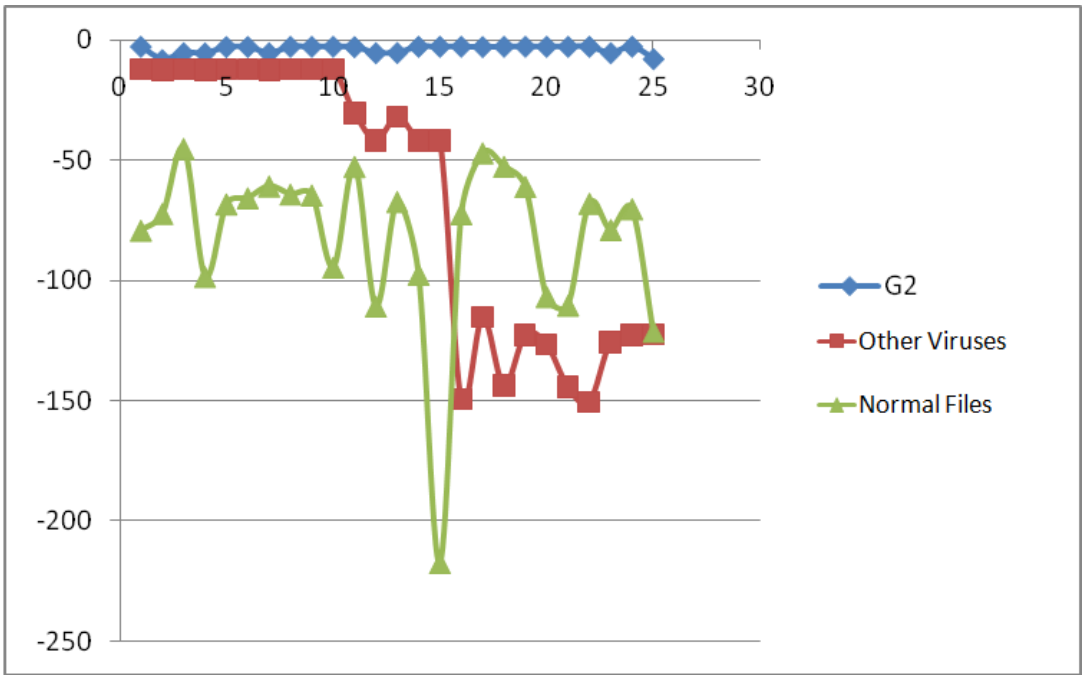


Figure 15: G2 Family Viruses with 2 hidden states

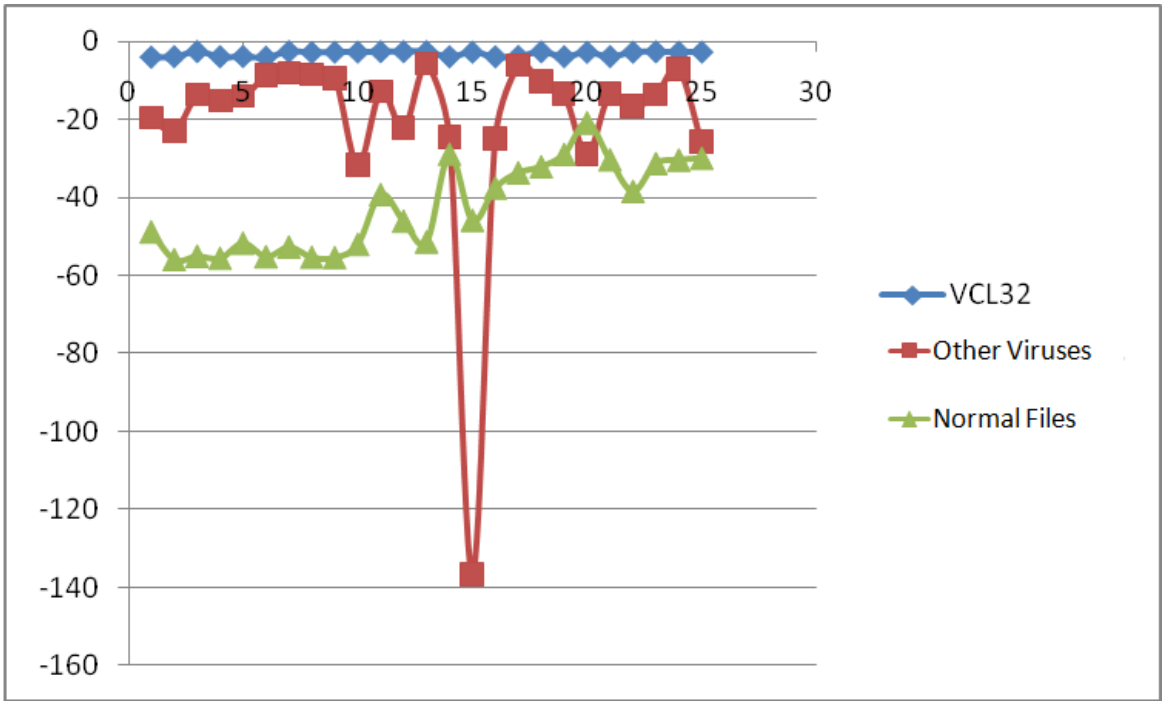


Figure 16: VCL32 Family Viruses with 2 hidden states

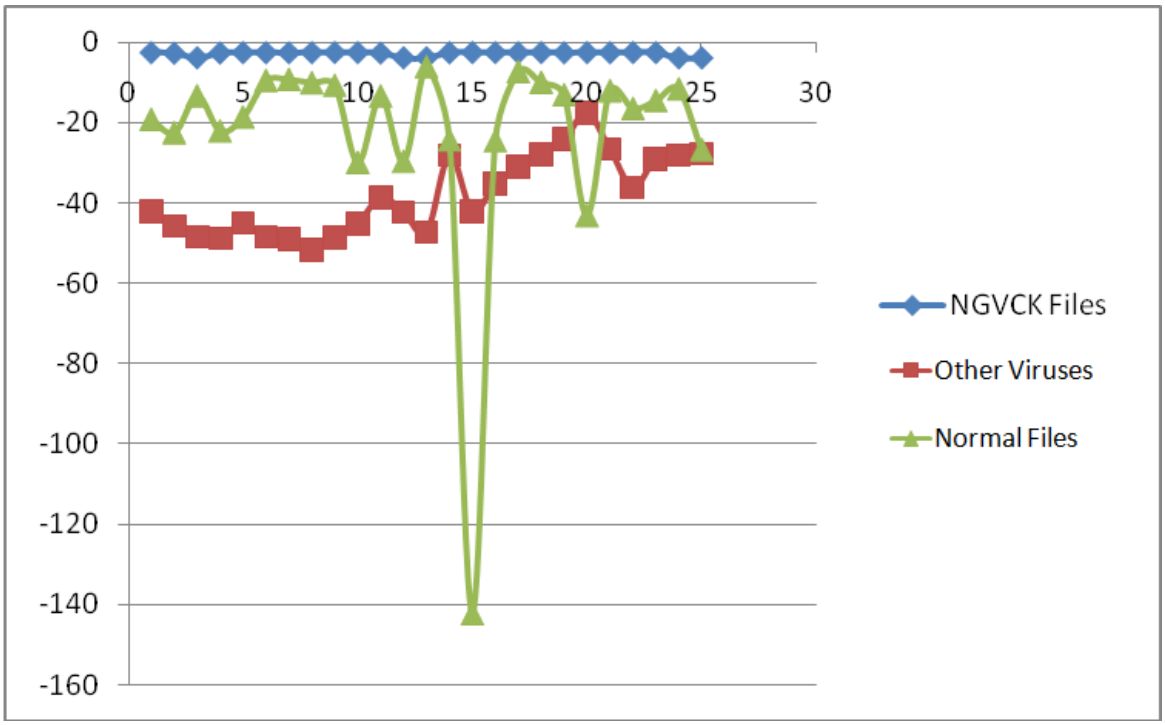


Figure 17: NGVCK Family Viruses with 3 hidden states

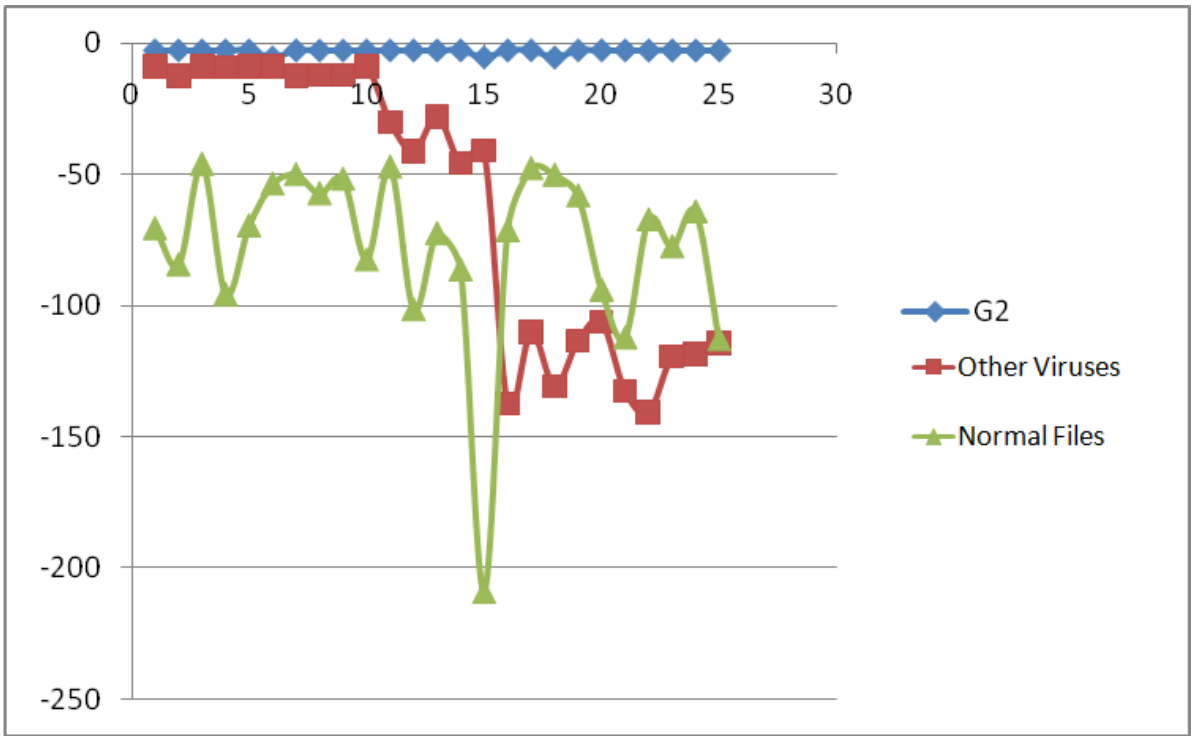


Figure 18: G2 Family Viruses with 3 hidden states

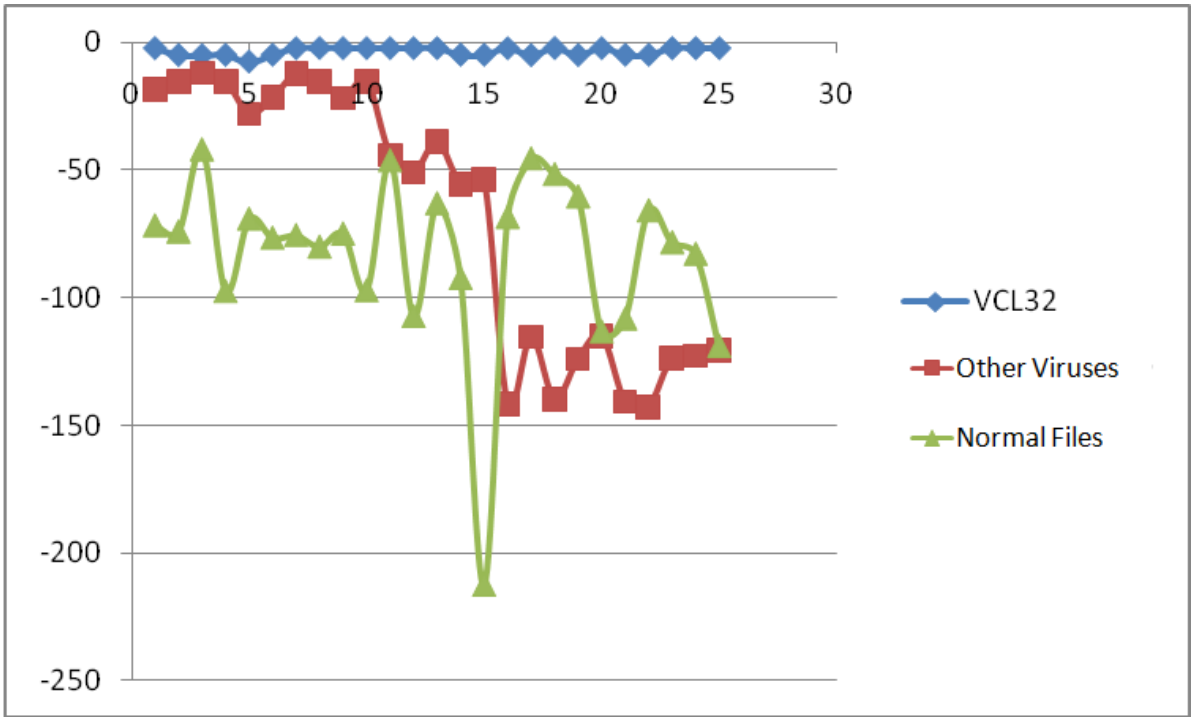


Figure 19: VCL32 Family Viruses with 3 hidden states

Appendix C : Garbage Instructions used

Shift Instructions

- Perform Shift Right by 0
- Perform Shift Left by 0
- AND with 1
- TEST with 1
- OR with 0
- XOR with 0

Floating Point Instructions

- Perform FADD,FSUB with 0
- Perform FMUL, FDIV with 1
- FLD and FST

Null Operation Instructions

- Swap register contents
- PUSH followed by POP
- Perform INC followed by DEC
- Perform ADD/SUB 0 on Registers

NOP Instructions

- NOP
- NEG CX
- NOT CX
- DEC CX