ANALYSIS OF PROGRAM MACHINE CODE AND APPLICATIONS TO DATA SECURITY

By

Humera Arshad

2007-NUST-MS-PhD IT-06

Supervisor

Dr. Fauzan Mirza

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Science in Information Technology (MSIT)

In

NUST School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST), Islamabad, Pakistan

(April 2011)

DEDICATION

All Praise And Thanks To Almighty Allah. To my parents who helped me carry on my studies despite of all hardships they faced.

APPROVAL

It is certified that the contents and form of thesis entitled **"Analysis of Program Machine Code and Applications to Data Security"** submitted by **Humera Arshed** have been found satisfactory for the requirement of the degree.

Advisor: _____Dr. Fauzan Mirza________

Signature: ____________________________

Date: ____________________________

Committee Member 1: _Dr. Ali Khayam___

Signature______________________________

Date:_________________________________

Committee Member 2: _Mr. Ali Sajjad___

Signature ______________________________

Date: _________________________________

Committee Member 3: _Mr. Kamran Zaidi_

Signature ______________________________

Date: _________________________________

CERTIFICATE OF ORIGINALITY

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Humera Arshed

Signature: ________________

Acknowledgements

I bow my head in gratitude to Allah, the creator of the mind that seeks knowledge and sets this pen into motion.

I would like to thank my parents for their support and encouragement, then my supervisor Dr.Fauzan Mirza, who has consistently encouraged me and provided me with the necessary prospective to complete this task. And I am thankful to my brothers Sohaib Arshed and Shoieb Arshad who were there to support me whenever I needed. I am also thankful to my friends for their support specially Anila Sahar Butt who constantly helped me during my project. My thanks are also due to my husband who gave me the opportunity to complete this project after marriage.

I thank you all!

Humera Arshed

ABSTRACT

The basic idea of exploit code detection inside network flows with the goal of preventing remote exploits is not new. For fast, efficient and real time analysis of network flows statistical methods are the most feasible choice. Unlike other anomaly based detection systems instead of modeling benign behavior we are trying to figure out the probability of

presence of executable code in normal network traffic flows. Markov chains are used to model the transition probabilities. This technique is supposed to look for assembly language code in any kind of compressed and uncompressed data streams. We are trying to overcome the size constraints faced by already presented statistical methods in this domain. Our technique should be efficient enough to locate few bytes (i.e. 20 to 30 bytes) of assembly code in fairly large files which may larger than 20 MB.

Like any stochastic process in coding at any step if it is provided with present state, the chances that future states will be equally likely are very less; some future states will have more probability of occurring then the others. This property is referred as Markov property: *“Given the present future states are independent of the past states”.*The changes of the state are called transitions and the probabilities associated with various state changes are called transition probabilities. Fairly efficient estimates can be obtained if the proper transition probabilities are obtained.

We have used this approach to design a novel algorithm that can distinguish malcode/exploit shellcode from benign code or random data. We show that our algorithm can identify shellcode as short as 57 bytes with reliability.

Table of Contents

CHAPTER 1 10
1.1 Problem Background 10
1.2 Problem Definition 15
1.3 Problem Statement 16
1.4 Objectives and Goals of Research 17
1.5 Complexity Involved 17
CHAPTER 2 19
2.1 Stealth Techniques 19
2.2 Binary Content Analysis 21
2.3 OVERVIEW OF NETWORK-BASED DETECTION TECHNIQUES 22
2.3.1 Signature-based Techniques 22
2.3.2 Anomaly-Based 23
2.4 Problem Description 23
CHAPTER 3 25
3.1 RELATED WORK 25
3.2 Statistical analysis of binary content for malware detection 25
3.3 Using Intel's x86 architecture for malware detection 28
3.4 Discussion 29
CHAPTER 4 32
4.1 Significance of Statistical Methods 32
4.2 File Content Analysis 32
4.3 Feasibility of Statistical Techniques for Code Identification 33
4.4 Choice of platform 34
4.5 Statistical model for Code 34
4.6 EXPLANATION OF PROPOSED METHODOLOGY USING A TEST 35
PROBLEM 35
4.6.1 Original Problem Statement 35
4.6.2 Test Problem 35

4.6.3 Experiments 35
4.6.3.1 Setup 35
4.6.3.2 Training Data 36
4.6.3.3 Generating Random Text File 36
.6.4 N-gram analysis 37
4.6.4.1 Insufficiency for actual Data Set 38
4.6.5 Second Problem Statement 38
4.6.5.1 Identifying English Text Embedded in Other Language Text 38
4.6.5.2 Approach 39
4.6.5.3 Markov Property 39
4.6.5.4 Algorithm 40
4.6.6 Results 41
4.6.7 Conclusion 43
CHAPTER 5 44
5.1 DATA 44
5.1.1 Code Model 44
5.1.2 Benign Data Set 45
5.1.3 Exploit Code Data Set 45
5.2 Approach 45
5.2.1 Algorithm 46
5.3 Experiments 47
5.3.1 SHELLCODE Used for Following Test files 47
5.4 Results 52
5.5 Revising Code Model 53
5.6 Further investigation of Shellcodes 53
5.6.1 Percentage of each instruction in each shellcode 53
5.6.1 Percentage of Registers 55
5.6.2 Ratio of Operands 56
Chapter 6 57
Bibliography 59

CHAPTER 1

INTRODUCTION

This chapter introduces the research work that has been taken in this thesis. It includes the motivation for research, problem definition and also discusses the objectives and goals of the research work.

1.1 Problem Background

The steady evolution of communication systems that connect computers, particularly the outburst of the Internet and intranet networks, has resulted in the development of a new information era. A user can access to wide range of resources, including electronic business applications that provide a wide range of information and services with a single personal computer and a connection to the Internet .Though, more and more computers have become interconnected through various networks such as the Internet, exploitation by malicious computer users has also greater than before, particularly from invasions or exploits delivered over a network or over an information stream. As those experts in malware will recognize, these attacks come in many different forms. These attacks include computer viruses, computer worms, system component replacements, spyware, adware, denial of service attacks; even misuse/abuse of legitimate computer system features, all of which exploit one or more computer system vulnerabilities for illegitimate purposes. The professionals will realize that the various computer attacks are technically distinct from one another, Although for the sake of simplicity in description, all malicious computer programs will be generally referred to hereinafter as computer malware, or more simply, malware[1].

A computer connected to the Internet may receive an attack through the network as an information stream so that vulnerability on the computer can be exploited. A flaw in a system's security that can give way to an attacker consuming the system in a manner other than the designer anticipated. Such security risks are classified as vulnerabilities. The term "vulnerability" associated to some other fundamental security terms as shown in the following diagram:[2]

```
            + - - - - - - - - - - - - - +  + - - - - +  + - - - - - - - - - - - -
+
      | An Attack:              |  |Counter- |  | A System Resource:   |
      | i.e., A Threat Action   |  | measure |  | Target of the Attack |
      | +----------+            |  |         |  | +-----------------+  |
      | | Attacker |<==================||<=========                 |  |
      | |   i.e.,  |   Passive  |  |         |  | |  Vulnerability  |  |
      | | A Threat |<=================>||<========>                 |  |
      | |  Agent   |  or Active |  |         |  | +-------|||-------+  |
      | +----------+   Attack   |  |         |  |         VVV          |
      |                         |  |         |  | Threat Consequences  |
      + - - - - - - - - - - - - +  + - - - - +  + - - - - - - - - - - -+
```

There could be one or more vulnerabilities that can be found in a resource (both physical and logical) which may be exploited by a threat initiator in a threat achievement process. On accomplishment of threat   ,the Confidentiality, Integrity or Availability properties of resources (potentially different that the vulnerable one) of the organization customers and suppliers, can potentially compromises. The so called CIA (Confidentiality, Integrity or Availability) triad is the basis of Information Security.

Vulnerability having one or more fully-implemented and operational instance of any exploit, a vulnerability for which any exploit exists, is classified as an exploitable vulnerability. Exploit codes are designed to take advantage of existing vulnerability in system and make the target system reacts in a way other than which planned by the designer [2].

Attackers usually gain control of hosts is through remote exploits. These exploit codes take benefit of any vulnerability present at target so that the target system responds in a way as projected by attacker. An active attack may alter system resources or affect their operation: so it compromises Integrity or Availability. While a passive attack attempts to draw on or manipulate information from the system but does not affect system resources: so it compromises Confidentiality [2].

An exploit , in French language, it meant for "achievement", or "accomplishment", is a piece of software, a chunk of data, or sequence of instructions that utilize any software 'glitch' or 'bug' to cause inadvertent or unexpected behavior to occur on computer software, hardware. This commonly comprises denial of service attacks, gaining control of a computer system or permitting privilege escalation. Software development methodologies rely on testing to ensure the quality and functionality of any code before release; this process usually fails to discover rare potential exploits. The term "exploit" generally refers to small codes designed to manipulate any software flaw, either remote or local, that has been exposed [2].

In computer security, a **shellcode** is a small piece of code used as the payload used to exploit vulnerabilities. Shellcode has given name because it normally starts a command shell from which the attacker can control the targeted compromised machine. Shellcode is frequently written in machine code, but any piece of code that exploits the vulnerabilities of host machine can be called shellcode. Shellcodes are classified as *local exploits* if they give control to an attacker over the machine it runs, and are called *remote exploits* if they gain control over another machine through a network, is used when an attacker aims to target a vulnerable process running on another machine on the local network or Internet. On successful execution, the shellcode may offer the attacker access to the compromised machine across the network. Remote shellcodes provide access to the attackers to target machine by using standard TCP/IP socket connections [3].

Shellcode is defined as a sequence of instructions infused and then executed by an exploited program. Shellcodes directly control the registers and the functionality of a program, so shellcodes are normally written in assembly language and then translated into hexadecimal opcodes. A shellcode provide the initial mean of intrusion for the attacker and helps to gain access to a shell (command interpreter) on the targeted host, in so doing presenting a convenient way to control the host to the attacker.

An exploit codes has to inject the shellcode into the target process before or at the time it begins to exploits a vulnerability and gain control over the program counter. The program counter is adjusted at the beginning of shellcode, after which it gets executed and performs its intended task. Shellcode injection is often accomplished done by embedding the shellcode in a benign file, or infusing the shellcode in data sent over the network to the vulnerable process, which is read and executed by the vulnerable process or environment in the case of local exploits or through the command line.

In order to achieve this goal the attacker is required to inject his shellcode into the memory of the victim host by hook and crook. Then the exploited vulnerability in a server process on the compromised host will transfer the control to his shellcode. Once control is transferred to the shellcode it acquires all the privileges of the process being exploited can use the services provided by the operating system with these privileges [3].

Stealth techniques use normal-appearing document to deliver this malcode to the target. It is observed that commercial antivirus software usually fail to detect embedded malware even when the malware signature is present in the antivirus database, firstly they do not provide deep scan of documents and secondly the size of shellcode is supposed to very small.

In some cases when a malware reside on a computer, it is immediately noticed by the user if it shows various unpleasant effects on host, such as applications or files data being erased or corrupted; performance degradation or system devices being disabled; or the computer system crashing or being incapable to carry out regular operations. However sometimes malware actions are covert and not immediately obvious to the user. For example, spyware normally monitors a user's computer habits secretly, such as Internet browsing behaviors, and pass on potentially sensitive data to another host on the network. The potentially sensitive data may be used for marketing, such as identifying a commercial product that matches the observed tendencies of the user. Then the spyware or an associated adware program helps to put on show relative advertisement to the user that sponsors the identified commercial product. The spyware may not be desirable to the user, as these advertisements, and other actions performed by the spyware interrupts the regular operation of the application and threaten the privacy of users [1].

1.2 Problem Definition

A variety of network-based techniques exists which are used to detect the presence of malicious executable code in network traffic flows. These techniques either work on assumption that every executable code exceeds a certain length could be malicious, or by matching with various behavioral patterns normally observed in shellcode [4].

Currently documents contain embedded code fragments and the corresponding applications also utilize these embedded code fragments. During document rendering or editing this embedded code is able to invoke libraries and other applications on the host indirectly. For example, for an embedded chart in word document displaying the contents of a spreadsheet will invoke Excel components when the Word document is accessed. Attacker gets an easy way to break through the system and reach third-party host-based applications that may harbor vulnerabilities which are not directly exploitable through remote exploits. Disturbingly, attackers designs exploits by keeping in mind the way in which modern document-handling applications operate, instead of making use of temporary vulnerabilities or flaws [5].

So only detecting and identifying the presence of code fragment in the data file or network traffic is not enough to diminish the risk. The point here is to detect the presence of malcode in benign documents, and to recognize that either the identified code is an executable assembly language code because shellcodes are supposed to be written in low level assembly language. Manual reverse engineering is the contemporary method of analyzing shell code, which requires significant expertise, and is time-consuming, and almost impossible for a wide-scale polymorphic attack [6]. However by locating a small piece of executable assembly language code in a benign file it may be possible to further investigate and classify that code to find its intended purpose.

1.3 Problem Statement

The aim is to build up a competent technique for identifying the existence of any executable code in a data stream or in a document which is capable of providing considerably better results in terms of low false positive and high true positive rates. The point here is to ensure that the identified code is an executable assembly language code which is capable of carry out certain functionality. Secondly it must be computationally efficient enough to apply it in real time applications like network intrusion detection.

The exploit codes may possibly be of different length, they frequently tend to be small in size. The exploit codes are normally designed to adjust themselves into small memory buffers which are set by protocol or other elements. Because of this, shellcodes are always trying to get smaller. Their size could be as small as few bytes i.e. 30 bytes. Here Statistical methods offer better efficiency but experience the size constraints, and do not show their worth to detect such smaller codes in larger streams of data.

Methods that employ knowledge of the instruction set of the target host and behavioral patterns of codes to recognize the executable code offers better reliability in detecting the presence of code but are computationally expensive and give the impression of being infeasible to apply in real time environment.

1.4 Objectives and Goals of Research

So the desired technique should be able to identify the presence of smaller executable code in normal stream of data with higher accuracy ,it should be computationally efficient enough to be practical in any real time environment.

The objectives are;

1) To detect the presence of code in data stream even if the code is very small in size

2) To identify the likely location of the embedded infection.

3) To be efficient enough to apply it on real time applications.

4) To be reasonably accurate.

1.5 Complexity Involved

The point at which the executable part of the attack begins is referred as the entry point of code. To find the entry point is a difficult problem, particularly without any prior knowledge about the program being exploited or vulnerability.

Exploits that use a NOP area may be detected by looking for a sequence of bytes that is executable from every word-aligned offset. Bearing in mind that NOP area is used to bring the flow of execution into the shellcode, in the NOP area execution may start at any point. However, NOP areas are not generally used on the windows platform. The detection mechanism become more complicated due to two properties of this architecture, the two properties are

- The length of instructions is not fixed,
- The majority of random byte combinations decode to valid instructions (it is a dense instruction set).

This means that depending on the exact position from where the decoding process being started, many different instruction sequences can be obtained from the same data. In addition to these two properties the fact that code may contain branches or loops make these techniques, based on instruction set, computationally expensive. Some authors suggest preliminary filtering of code before applying these techniques to lessen computational complexity [5].

CHAPTER 2

BACKGROUND STUDIES

This chapter provides background knowledge that is helpful in understanding the context of this research. It includes basic definition of important terms like data alignment, keywords, key phrases, semantic relations, taxonomy and indexing

2.1 Stealth Techniques

Malware is progressively employing more, stealth techniques to hide on a computer or otherwise avert detection by programs intended to protect a computer (e.g., antivirus software, anti-spyware software, and the like). A computer connected to the Internet may be attacked so that vulnerability on the computer is exploited and the malware is transported over the network as an information stream. By way of another example, malware may become occupant on a computer using social engineering techniques. For example, a user may access a resource such as a Web site and download a program from the Web site to a local computer. While the program may be depicted on the Web site as providing a service desirable to the user; in actuality, the program may perform actions that are malicious. [1]

Stealth techniques used to transfer malcode to a targeted host in an otherwise normal-appearing network stream or in a benign document. Object-oriented dynamic compos ability of current document applications and formats, can be exploited by the attackers the malcode hidden in benign documents can reach third-party applications which may not be as straightforward by network-level service attacks. Such attacks can be very choosy and difficult to detect compared to the typical network worm threat, due to the complexity of these applications and data formats, as well as the huge number of document-exchange vectors. [1]

It is observed that malcode has been embedded in PDF, Word, Excel, and PowerPoint documents [8,9,10] converting them into a medium of transportation for host intrusions. These Trojan-infected documents can be served up by any arbitrary web site or search engine in a passive “drive by” fashion, spread over email or instant messaging (IM), or even introduced to a system by other media such as CD-ROMs and USB drives, bypassing all the network firewalls and intrusion-detection systems. Furthermore, the attacker can use such documents as a stepping stone to reach other systems, inaccessible via the regular network. Any machine within a network with the ability to open a document with embedded malcode can become the distribution point for the malcode to reach any other target host inside that network [5].

The authors of , A Study of Malcode-Bearing Documents, had investigated the possibility of detecting embedded malcode in Word documents using two techniques: static content analysis using statistical models which use document content for constructing statistical model, and dynamic tests which carried out at run-time on diverse platforms. The experiments demonstrate these approaches are good at detecting zero-day attacks as well as known malware. Although both approaches are not perfect and are suffering from different problem, but even then these approaches presents an insight to challenges in tackling the problem and prospects for future research [5].

Un-patched vulnerabilities found in network services offer opportunities for external attackers to triumph over computer systems by taking advantage of these vulnerabilities. Several techniques have been suggested to defy this well-known problem. Bugs and flaws left by software developers in software give way to vulnerabilities, although programming language security approaches supposed to detect these detect these bugs automatically. All bugs cannot be located and removed because of technical complexities involved in static analysis of programs. Intrusion detection is an alternative approach is to detect attacks at runtime .But significant overheads as an undesirable side-effect are observed due to runtime checks [6].

Existing and most popular defense mechanisms are signature-based techniques which use known byte patterns. It has been shown in that due to some performance issues commercial antivirus software do not carry out exhaustive scan and are unable to identify embedded malware even when the malware signature is present in the antivirus database [26].

2.2 Binary Content Analysis

Primarily n-gram approaches are being used for Probabilistic modeling in the area of content analysis [14,15,16].A distribution for the frequency of 1-gramor fixed size n-gram, is computed from the binary contents of file. An initial research attempt in this area is the Malicious Email Filter [17] applied a naïve Bayes classifier algorithm to the binary content of email attachments suspicious to be viral. To determine whether emails possibly include malicious attachments that should be filtered, the classifier was trained on both known viruses and “normal” executables. Similar techniques had been applied by others, for example, Abou-Assaleh et al [18, 19] to detect viruses and worms. Moreover, Karim et al. advocate that malicious programs are repeatedly related to preceding ones [20]“n-perms” were defined as a variation on n-grams. An n-perm represents every possible permutation of an n-gram sequence, to detect possibly permuted malicious code; n-perms can be used to match. McDaniel and Heydari [21] used byte-value distributions of file content to set up algorithms for generating “fingerprints” of file types Though, they computed a single representative fingerprint for the entire class instead of computing a set of centroid models. Maybe this strategy was unwise. Information may be lost in mixing the statistics of different subtypes and then by taking average of the statistics of an aggregation. For data protection and automatic file identification, AFRL proposed the Detector and Extractor of File prints (DEF) process [22] By applying the DEF process, visual hashes were generated, known as file prints, to compare the similarity between data sequences, to calculate the integrity of a data sequence, , and to recognize the data type of an unfamiliar file. Goel introduced Complexity metrics, for file type identification [23][5].

2.3 OVERVIEW OF NETWORK-BASED DETECTION TECHNIQUES

2.3.1 Signature-based Techniques

In Signature-based techniques detection methods work by scanning traffic for evidence of known attacks so these are also as misuse-based techniques, these systems basically works by starting simple string searchers, but the recent literature much more complex. To begin with, an up to date “signature” can acquire many different shapes. In recent past information flows in networks are getting huge in volume, signature generation by hand is no more feasible, recently experts have developed ways to automatically generate signatures from suspicious content to deal with fast-spreading worms. [4]

A signature for a certain attack can be created only when there is some distinctive characteristic f that attack. It is essential to set the victim program in a the appropriate state for exploitation so some techniques assume that even most recent polymorphic attacks most critical part of exploit stay invariant and so does the signature. [11].Exploit code itself can serve as signatures. However it is considered as dead end because of modern polymorphic encoders. Even though some researchers have reported achievement with these methods, but this option is usually dismissed against the current crop of polymorphic encoding engines [4].

2.3.2 Anomaly-Based Intrusion Detection Systems

Anomaly-Based Intrusion Detection Systems is considered best for detecting zero-day exploits. They used monitor system activity and detect intrusions and misuse by and classifying system behavior as either *normal* or *anomalous*. Anomaly-based techniques generate an alert when it encountered unusual input and mark it as an attack. Instead of looking for an attack it used to model normal behavior of the system, anything which deviates from normal behavior is considered as anomaly. Unfortunately, these systems require expensive human monitoring because they tend to produce huge number of false positives, that's why automated responses are not sufficient. However, accuracy of network-based anomaly detection systems can be enhanced by coupling them with another second detection system which can lessen false positives and system may still is effective [3].

2.4 Problem Description

Increase of fast-spreading network worms is the biggest challenge in today's network security world. The threat caused by these worms is apparent: the network worms can flood the vulnerable host's population within no time, this scenario left very limited time for human intervention and gives dominating ability to the attack of executing arbitrary code on the victim hosts. How to counter that issue is an open problem for recent research in that field, and still there is no eventual solution. In fact, the developments of network worms seem to have same pace of development as developments in computer viruses [12, 13].

Vulnerabilities in server software are exploited by network worms they facilitates an attacker to execute arbitrary code on host machine. If they are defined by considering only this perspective this perspective, then they are just a self-replicating type of attack that has existed for a long time. Due to un availability of proper term, these attacks are referred as “shellcode attacks” in literature, as the name describes it is a code an attacker uses to establish control over the victim host [4].

Network-based solutions for the problem used to scan the network flows to the host to look for malicious or anomalous traffic and are generally designed by performance in mind, so they can successfully monitor the large amount of network traffic flows which passes over a link to detect the exploit code inside traffic. These solutions are usually developed from existing developments in network intrusion-detection devices [4].

An orthogonal approach involves detecting exploit code inside network flows in averting remote attacks. A significant benefit of this proactive approach is that the countermeasures can be applied even before the exploit code starts disturbing the target program [2].

For preventing remote exploits the idea of exploit code detection inside network flows is already known. Snort and Bro suggested packet-level pattern matching for network-based intrusion detection systems, this exploit code detection mechanism require to specify corresponding signature. Signature based systems are trouble-free in implementation and perform sound. Their security guarantees are highly dependent on signature repository. Evasion can simply be achieved by operating outside the signature repository and can be easily done either by altering or instruction sequences (metamorphism) or instructions, encryption/ decryption (polymorphism), or zero-day exploits revealing an entirely new vulnerability and the corresponding exploit.

It is acknowledged by the authors that a determined attacker can dodge exiting techniques for network-level worm detection. if future threats are taken in consideration and 100% accuracy is desired , then it can be concluded that current network-level detection is not up to the mark. [24,25]

CHAPTER 3

LITERATURE REVIEW

3.1 RELATED WORK

I observed two major techniques during literature review. One is the statistical analysis of binary content of documents and data streams, and the other interesting category of detection techniques was identified that uses knowledge of the instruction set of the target host to identify the executable part of shellcode. These techniques appear to be reasonably effective against current polymorphic techniques.

3.2 Statistical analysis of binary content for malware detection

In “Towards Stealthy Malware Detection” the authors presented the idea of identifying the embedded malcode in the documents by applying statistical n-gram method. The method trains n-gram models from a collection of input data, and uses these models to test whether other data is similar to the training data, or sufficiently different to be deemed an anomaly. The method allows for each file type to be represented by a compact representation of statistical n-gram models. Using this technique, they classified files into different types, or validated the declared type of a file, according to their content, instead of using the file extension only or searching for embedded “magic” numbers [16].

In a research paper on embedded malware detection named “Towards stealthy malware detection” the authors proposed to use n-gram analysis for embedded malware detection. They used 1-Centroid, Multi-Cancroids and Exemplar files as centroids for modeling benign and malware files. 1-gram and 2-gram distributions were used for this purpose. Mahanalobis distances of a given (unknown) file from the benign and the malware model were used for classification. The authors in this paper evaluated prior work they set two objectives first is to detect the presence of malcode and second is the location of embedded malcode. In prior study it was shown that the assumption of using truncated file is unrealistic, infection can be present anywhere in the file. In the next step they computed n-gram on whole file but no significant change in distribution is observed. It was argued that the statistical contents of the embedded malware are averaged out by large amounts of benign data hence failed to show any discernable change in the distribution [16].

In “Towards stealthy malware detection” the authors used block wise n-gram analysis. The in this paper used same method o n their dataset and get some representative results of the Mahanalobis distance between the block-wise 1-gram distribution and the benign file model. Block size of 1000 and 500 bytes were used. But they observed that distance value stays more or less constant in the embedded malware blocks. Then they repeated the experiment by using 2-gram analysis but in spite of increased complexity in calculation no significant change in performance was observed. They support that observation by the argument that 1000 bytes does not provide enough data for statistical distribution particularly in the case of 2-gram analysis. Straightforward solution is to increase the size of the block but the block size roughly defines the lower bound on the size of malware that can be detected .So there is an obvious tradeoff between the block size and minimum malware size that can be detected.

Increase in block size will increase high false negative rates so the study was stopped at 2-gram. At this point the authors come up with the idea of testing the effectiveness of the models being used. They conjecture that either the either n-gram analysis is not a good method for embedded malware detection or Mahanalobis distance is not a good enough quantification measure for differentiating between benign and malicious n-grams. It is assumed that statistical contents of the malware are different from the benign file, and then entropy of the block wise distribution on the infected file should change at the embedding location. But it was observed that entropy calculation on 2-grams provide qualitatively similar results to the Mahanalobis distance. By the failure of both Mahanalobis distance and entropy measures the authors concluded that a simple n-gram distribution does not provide sufficient information to detect embedded malware[16].

A similar approach is followed in “Embedded Malware Detection using Markov n-grams”. The authors suggest some improvements in the statistical model and extended this work to obtain better results. However, both cases face the limitation that when the size of benign file is significantly larger than the malware size then statistical contents of the embedded malware are averaged out by large amounts of benign data hence failed to achieve significant results [26]. Then they tried to overcome those shortcomings in the model. They observed that 2-gram distribution is in fact the joint distribution of two 1-gram symbols. This joint distribution may contain some redundant information which is not pertinent to the present embedded malware detection problem. For accurate detection, it is important that this redundancy is removed. They studied the different statistical properties and find an interesting property that gives insights into statistical properties of file data was the analysis of byte level autocorrelation of benign files. By checking the correlation authors observed that benign files exhibit a clear 1st-order dependence which is missing in malcode due to their size constraints. Authors used that property to define statistical model for this problem and find that Markov chains can be used to model the conditional byte distribution. Then a mathematical measure is required to quantify the changes. Entropy rate is used, as it is assumed that statistical properties of the embedded malware will be different from the statistical properties of the benign file in which it is embedded, expected entropy of the consequent Markov chain (derived from the infected file) should be perturbed at the embedding locations. This technique gives significantly better results as compared to other techniques but it gives high false positive rate. Underlying principle of this proposed detector is based on statistical analysis; a crafty attacker may launch a mimicry attack by modifying the malcode to have a benign looking statistical distribution [26].

3.3 Using Intel's x86 architecture for malware detection

The most prevalent instruction set architecture for servers and personal computers at the moment is Intel's x86 architecture. There is a category of identification techniques that make use of knowledge about instruction set and executables.

In "A Fast Static Analysis Approach to Detect Exploit Code inside Network Flows" the authors used static analysis by disassembling the binary streams in network flows. The approach was to perform binary disassembly starting from the first byte without any additional processing. The convergence property will ensure that at least a majority of instructions including branch instructions has been recovered. However, this approach is not resilient to data injection. Having performed binary disassembly basic blocks were identified by constructing the control flow graph (CFG). Then valid and invalid blocks were identified. To check the validity of identified CFGs it is followed by data flow analysis based on program slicing to complete the process of elimination. Program slicing is a decomposition technique which extracts only parts of a program relevant to a specific computation. Although this technique described an efficient static analysis based litmus test to determine if a network flow contains exploit code but on the downside, it was not fast enough to handle very large network traffic, and therefore, there are deployment constraints [7].

In “Network-Level Polymorphic Shellcode Detection Using Emulation”, the authors suggested the approach of executing every potential instruction on a NIDS-embedded CPU emulator to identify the execution behavior of polymorphic shellcodes. On the downside it works only with self-contained shellcode[27].As the self-contained shellcode do not have known address for variables but a motivated attacker could evade network-level emulation by constructing a shellcode that involves registers or memory locations with a priori known values that remain constant across all vulnerable systems. For example, if it is known in advance that the address 0x40038EF0 in the vulnerable process’ address space contains the instruction ret, then the shellcode can be obfuscated by inserting the instruction call 0x40038EF0 at an arbitrary position in the decoder code [27].

3.4 Discussion

The authors in this paper evaluated prior work they set two objectives first is to detect the presence of malcode and second is the location of embedded malcode. In prior study it was shown that the assumption of using truncated file is unrealistic, infection can be present anywhere in the file. In the next step they computed n-gram on whole file but no significant change in distribution is observed. It was argued that the statistical contents of the embedded malware are averaged out by large amounts of benign data hence failed to show any discernable change in the distribution [16].

In “Towards stealthy malware detection” the authors used block wise n-gram analysis. The in this paper used same method o n their dataset and get some representative results of the Mahanalobis distance between the block-wise 1-gram distribution and the benign file model. Block size of 1000 and 500 bytes were used. But they observed that distance value stays more or less constant in the embedded malware blocks. Then they repeated the experiment by using 2-gram analysis but in spite of increased complexity in calculation no significant change in performance was observed. They support that observation by the argument that 1000 bytes does not provide enough data for statistical distribution particularly in the case of 2-gram analysis. Straightforward solution is to increase the size of the block but the block size roughly defines the lower bound on the size of malware that can be detected .So there is an obvious tradeoff between the block size and minimum malware size that can be detected. Increase in block size will increase high false negative rates so the study was stopped at 2-gram [16][26].

At this point the authors come up with the idea of testing the effectiveness of the models being used. They conjecture that either the either n-gram analysis is not a good method for embedded malware detection or Mahanalobis distance is not a good enough quantification measure for differentiating between benign and malicious n-grams. It is assumed that statistical contents of the malware are different from the benign file, and then entropy of the block wise distribution on the infected file should change at the embedding location. But it was observed that entropy calculation on 2-grams provide qualitatively similar results to the Mahanalobis distance. By the failure of both Mahanalobis distance and entropy measures the authors concluded that a simple n-gram distribution does not provide sufficient information to detect embedded malware [26].

Then they tried to overcome those shortcomings in the model. They observed that 2-gram distribution is in fact the joint distribution of two 1-gram symbols. This joint distribution may contain some redundant information which is not pertinent to the present embedded malware detection problem. For accurate detection, it is important that this redundancy is removed. They studied the different statistical properties and find an interesting property that gives insights into statistical properties of file data was the analysis of byte level autocorrelation of benign files. By checking the correlation authors observed that benign files exhibit a clear 1st-order dependence which is missing in malcode due to their size constraints. Authors used that property to define statistical model for this problem and find that Markov chains can be used to model the conditional byte distribution. Then a mathematical measure is required to quantify the changes. Entropy rate is used, as it is assumed that statistical properties of the embedded malware will be different from the statistical properties of the benign file in which it is embedded, expected entropy of the consequent Markov chain (derived from the infected file) should be perturbed at the embedding locations [26].

This technique gives significantly better results as compared to other techniques but it gives high false positive rate. Underlying principle of this proposed detector is based on statistical analysis; a crafty attacker may launch a mimicry attack by modifying the malcode to have a benign looking statistical distribution.

Although the methods based on instruction set provide a novel means of detecting code, they are computationally expensive and their resilience against evasion techniques is doubtful [4]. Thus, these techniques do not necessarily “raise the bar” for the attacker, while their cost for the defender in terms of the resources that need to be devoted to detection can be significant. At this point, it remains unclear whether accurate network level detection is feasible.

CHAPTER 4

Proposed Methodology

4.1 Significance of Statistical Methods

N-gram analysis has been proved practical in many scenarios for different, and is well understood and efficient to implement. One can map and embed the data in a vector space to efficiently by converting a string of data to a feature vector of n-grams, two or more streams of data can be compared. Otherwise, one may compare the n-grams distributions contained in a data set to determine the consistency of some new data with the previous one **[**16**].**

The authors of “Towards Stealthy Malware detection” used the statistical binary content of files to compute N-grams, to differentiate between normal executables and viruses. Although they do not obtain desired results but still that methodology opened up new avenues of future work .Later the authors of “Embedded Malware Detection using Markov n-grams” applied this n-gram approach for detection of embedded malware and get promising results.

4.2 File Content Analysis

From the previous discussion we came at the conclusion that if statistical methods and static analysis of instruction set could be combined into one place to achieve the targets of code efficiency and correctness while detecting the presence of exploit code in information stream. The basic idea is to apply some appropriate statistical model on the disassembly of the document or network stream. It is supposed that the syntactic information can reveal more structure and information about file formats, so disassembly of data can reveal more structure, meaning and information about the presence of code as compared to the information obtained by the analysis of byte order only. It could be observed from the sequence of instructions that either they present some meaningful or executable sequence of instructions or they are just random instructions disassembled from data.

4.3 Feasibility of Statistical Techniques for Code Identification

Coding is not a purely random process. If at any step provided with present state, the chances that future states will be equally likely are very less. Some future states will have more probability of existing then the other. So it’s a probabilistic or stochastic process. This property can be formally represented as Markov property. “Given the present future states are independent of the past states”.

At each step system may change its state from the current to another state according to a probability distribution. The changes of the state are called transitions and the probabilities associated with various state changes are called transition probabilities. The controlling factor in Markov chains is the transition probability it is a conditional probability for the system to go to a particular new state given the current state of the system. Being a stochastic process means that all state transitions are probabilistic. Fairly efficient estimates can be obtained if the proper transition probabilities are obtained.

For the code part instructions are arranged in a logical way and have to follow some syntactic rules so the dependence structure should be different from data files. This fact is supported by some observations as well.

4.4 Choice of platform

The most prevalent instruction set architecture for servers and personal computers at the moment is Intel’s x86 architecture. Most of the shell codes and exploits are usually written in assembly language. Even if they are not in assembly language the disassembly of any code, executable and data file can be obtained from its binary representation. So for this problem we are trying to identify patterns of the code in the disassembled files.

4.5 Statistical model for Code

Instead of modeling the structure of data files, patterns and behaviors of code can also be modeled for the identification of code from data stream for the application of statistical methods. As it was described the [5] that data files exhibit first order dependence structure and it is missing in the code files. This property cannot be used for training the code model. To identify the presence of code sequence it is necessary to know the properties of code or to have some reference model for code to compare with.

So for the start we decided to check the validity and relevance of these statistical parameters like n-grams, Markov chains and entropy for our data set and code model.

4.6 EXPLANATION OF PROPOSED METHODOLOGY USING A TEST PROBLEM

4.6.1 Original Problem Statement

The aim of this project is to find an efficient technique of locating any code embedded inside some data stream the code may be very small in comparison to data stream. i.e. code may consists of 40 to 50 bytes in a data whose size may be many kilo or mega bytes.

4.6.2 Test Problem

Initially we tested our proposed methodology to check its feasibility for the problem in hand; the data used for the test problem is different from the data set involved to deal with real problem. The purpose to solve this test problem is to initially apply the suggested techniques on smaller and more familiar data set. As a first step we were searching for a method to extract one type of data from another type of data which is exponentially larger than the data we desired to identify.

So for this purpose we decided to start from text, we chose to identify a valid English language sentence embedded in a file of random text, random text consists of alphabets arranged in words but are randomly arranged, as we were bearing in mind that the data stream as random set of instructions obtained after the disassembling the stream in comparison to embedded code.

4.6.3 Experiments

4.6.3.1 Setup

We constructed a stochastic model of English language by collecting various random texts from Internet including fiction, news, technical articles and political reviews. Then letter frequencies and transition probabilities of individual letters were calculated. We constructed a transition probability matrix for English text.

4.6.3.2 Training Data

English language is considered as random process. First we trained a stochastic model of English language to predict behavior of a random process, calculating the transition probabilities of n-grams. This model will be used as a reference of English text for future comparison while locating English text from random text. Anything that deviates from that model would be considered as random.

We constructed that model for the English language instead of a model for random text, because we were emphasizing on tracing English language. Otherwise if we hypothesized that anything that is not random in the random text file would be English text, would be an unjustifiable hypothesis. Secondly it provided us the benefit that once we have a reference model for English language then we can use it anywhere to compare with any other type of carrier file, like German or French text as it was needed at a later stage in the experiments.

4.6.3.3 Generating Random Text File

Random English text files were generated by random number generator by providing 48 bit seed. Randomness of generated file is checked by calculating the frequencies individual alphabets in generated file. Almost Uniform frequencies of alphabets showed the randomness of file.

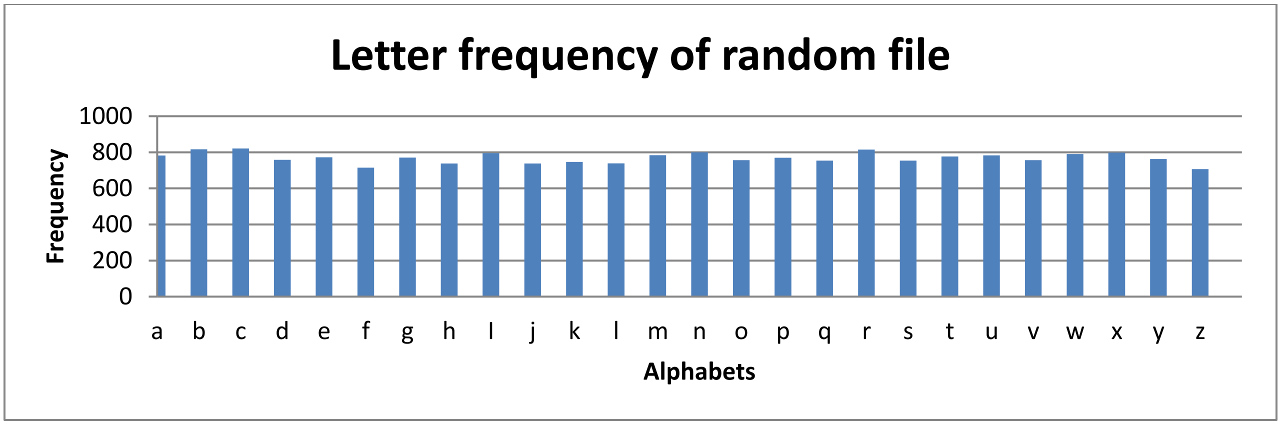


Fig:1. Frequencies of English alphabets observed in random text file.

.6.4 N-gram analysis

N-gram analysis using letter trigrams was proved sufficient to locate the presence of English sentence in random text. For sequences of characters, the 3-grams (sometimes referred to as "trigrams") that can be generated from "good morning" are "goo", "ood", "od ", "d m", " mo", "mor" and so forth.

$$P(Xi|Xi\text{-}1, Xi\text{-}2, Xi\text{-}3\ldots\ldots\ldots Xi\text{-}n)$$

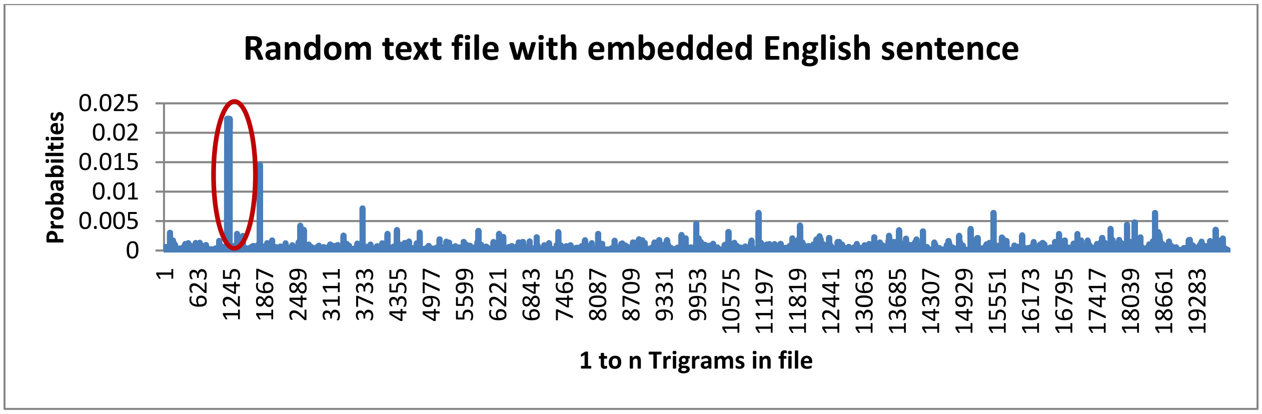


Fig:2. This file show the result of tri-gram analysis of a random file having an English sentence of length 51 embedded at index `1185`.

4.6.4.1 Insufficiency for actual Data Set

When we applied this approach on some representative files of actual data set , which consists of some benign documents in different formats like Text, PDF, PS,DOC,PPT or DOCX with embedded shellcodes. These files are then disassembled to assembly language. Each instruction corresponds to an alphabet, sequence of three instruction formed a trigram, N-gram analysis were applied on instruction trigrams. We carried out many experiments by changing the size and types of carrier files as well as the size and nature of embedded shellcodes.

But it does not provide any promising result for the identification of shellcode within data file. It could be concluded that the data files were not random with respect to shellcodes.

4.6.5 Second Problem Statement

So the next problem statement is to “Identify the location of embedded English language sentence in languages like French, German, and Italian”.

4.6.5.1 Identifying English Text Embedded in Other Language Text

So we decided to revise our test problem so that we could locate the desired information from less random data. This time we had emphasize to locate English text embedded in some other language like Italian, French or German text. It is a well known fact that the letter frequencies and alphabetical patterns in these languages are different from English language but surely not random.

4.6.5.2 Approach

It is obvious from the above experiment that English text is not random, the letters are arranged in some predictable fashion, so is the case for other languages like French, German and Italian. That’s why the method we used for the detection of embedded English sentence in random text did not proved sufficient for identifying embedded English sentence in German, French or Italian text. As these languages like French, German and Italian uses same alphabets like English but definitely there syntax and grammar and lexical properties are different from English Language. So we need to find a method which considers these differences.

4.6.5.3 Markov Property

As the Markov property states that the conditional probability distribution for the system at the next step (and in fact at all future steps) given its current state depends only on the current state of the system, and not additionally on the state of the system at previous steps, so it can be represented as:

$$P(Xi|Xi\text{-}1,Xi\text{-}2,Xi\text{-}3\ldots\ldots\ldots Xi\text{-}n)=P(\mathrm{Xn}|\mathrm{Xn}\text{-}1)$$

Here I consider the simplifying assumption of N-gram models that the current state depends on a constant number of the preceding states. Here we used two-grams or bigrams. Each letter depends on previous two letters. n is length of file:

$$P(S) = P(X_i)P(X_{i+1}|X_i)P(X_{i+2}|X_i,X_{i-1})\ldots\ldots P(X_{i+n}|X_{i+n-1}X_{i+n-2})$$

$P(X_{i+1}|X_i)$ presents the conditional probability of next letter X_{i+1} if the probability of previous letter X_i is known.

$$P(S)= P(x_i)*(P(x_i \Pi x_{i+1})/P(x_i))*(P(x_{i+1}\Pi x_i x_{I+1})/P(x_i x_{i+1}))\ldots\ldots\ldots P(x_{i+n}|x_{i+n-1}x_{i+n-2})$$

4.6.5.4 Algorithm

```
While(end of file){
        Sentence:=first 30 character from text file,
        Var S:=0;
        for i from 0 to 30 by 1{
                $P= p(x_i)*(p(x_i \Pi x_{i+1})/p(x_i))*(p(x_{i+1}\Pi x_ix_{I+1})/p(x_ix_{i+1}))$
                S:=s+P
        }
        S:=0;
}
```

For this calculation I used 2 different transition probability matrices:

I-Probabilities of individual letters for getting $P(x_i)$

II-Probabilities of $P(x_i \Pi x_{i+1})$ are calculated from text samples and stored in a(27*27) matrix to provide the probabilities of every possible bi-gram i.e. aa,bb,as,ad,th

III-Probabilities of $P(x_{i+1}\Pi x_ix_{I+1})$are calculated from text samples and stored in a (729*27) matrix to provide the probabilities of every possible tri-gram i.e. aaa,bb,ash,and,the

4.6.6 Results

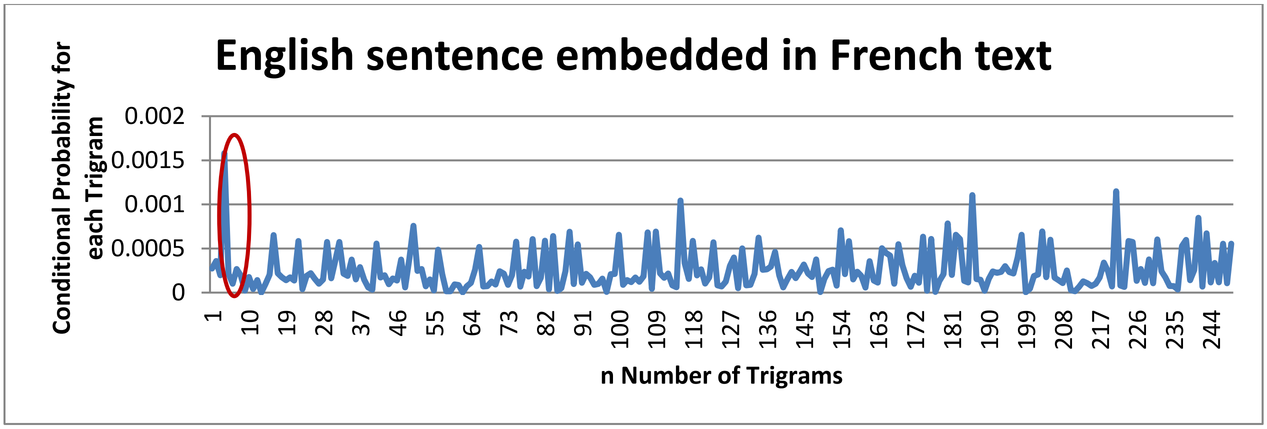


Fig:3. The file size is 5 kb having French Text and English sentence is 42 character (without space) long. English sentence is embedded in the start of the file.

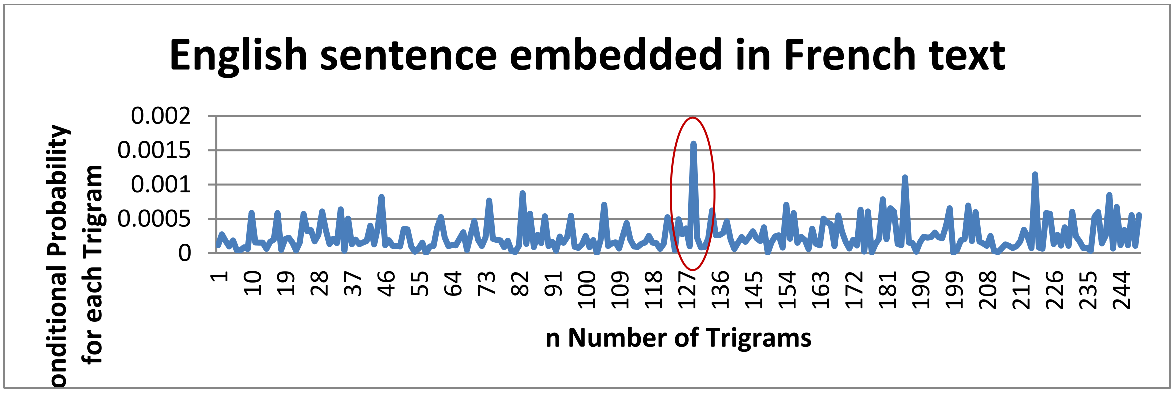


Fig:4. The file size is 5 kb and English sentence is 42 characters (without space) long. English Sentence is embedded in the middle of the file.

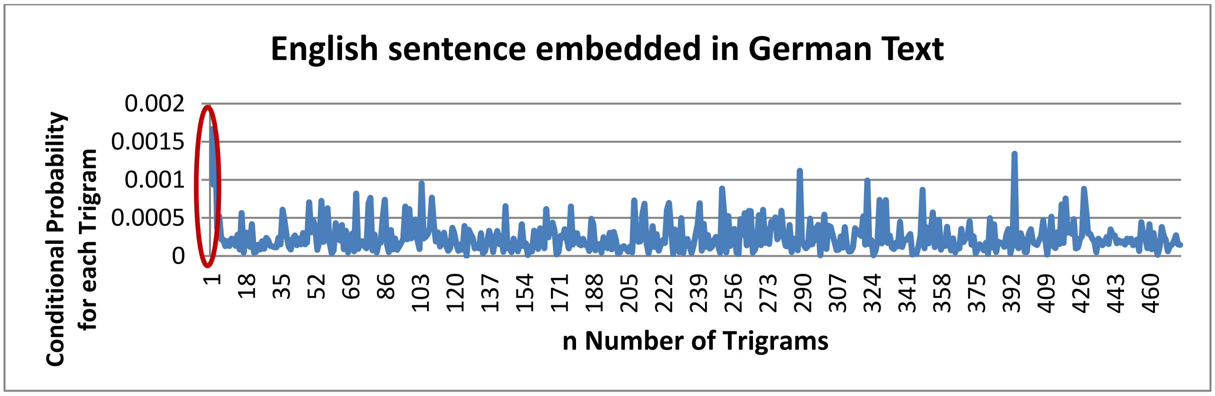


Fig :5. The file size is 10kb and English sentence is 73 characters (without space) long. English Sentence is embedded in the start of the file.

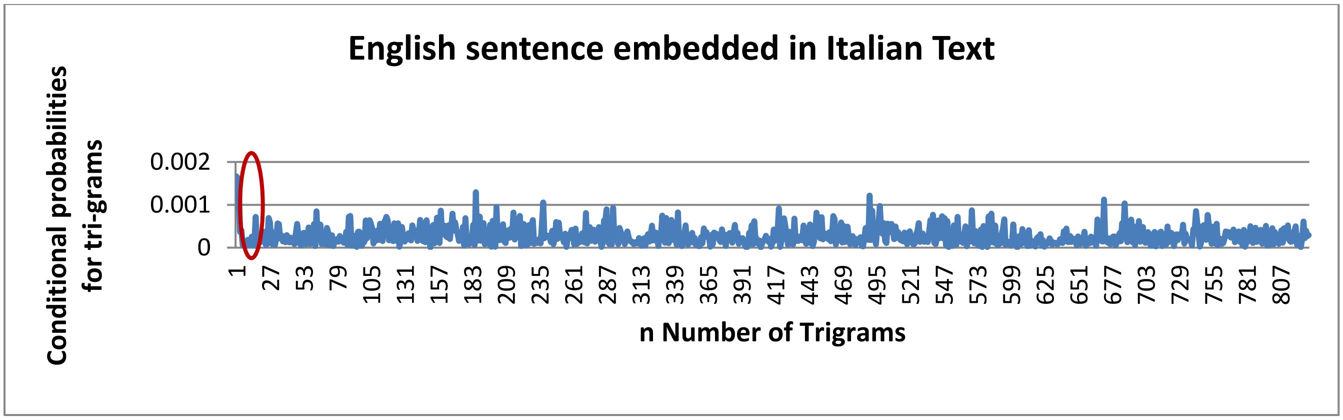


Fig :6. The file size is 17 kb and English sentence is 73 characters (without space) long. English Sentence is embedded in the start of the file.

4.6.7 Conclusion

The sum of conditional probabilities calculated over a sentence of length 30 characters long. The sum of probabilities is higher where the n-grams are constructed from English letters as shown in the above charts.

CHAPTER 5

RESULTS AND EVALUTAION

In this chapter we evaluate the results of the proposed algorithm, discussed in Chapter 4. We identify main evaluation criteria, the details of data set, system specification and results of the experiments carried through the system.

5.1 DATA

5.1.1 Code Model

A set of executable files for various applications was collected; the set included different type of executable like editors, web explorer, messengers, installers and software to handle multimedia. Binary representation of exe files was used. The size of extracted code varied from 8 kb to 15 Mb. For the current experiments we extracted code form the binary files of different type for a thorough study of code properties. These strings of hexadecimal characters are provided as input to the program. Each pair of hexadecimal character is representing each byte from the corresponding binary file.(i.e. C4 ,FF). These pairs of characters are then used as reference to lookup in opcodes table provided by Intel for x86 instruction set. Each pair of these hex characters would be disassembled to corresponding assembly language instruction. Whole file will be disassembled in similar way. The result will be stored in a file.

This disassembled file will be served as input to the next module. The code will count the occurrences for each pair of instruction in sliding manner. The number of times instruction *i* is followed by instruction *j* and then the number of times instruction *j* followed by *k* and so for every instruction. An nxn matrix is obtained by counting the transition probabilities for n instruction. n represents the set of instructions. A set of n=120 instructions was used in these experiment. The results are saved in the form of nxn table in an excel worksheet.

5.1.2 Benign Data Set

A set of normal documents of various types like JPEG,DOC,DOCX,MP3,PPT,PPTX,PDF were collected from internet. Also the binaries of these files were extracted to use in experiments. These would be use as a carrier files having shellcode embedded at some arbitrary position in them. List of these files and their description is attached as appendix.

5.1.3 Exploit Code Data Set

A set of 20 shellcodes is collected from **www.milw0rm.com** **.**Again the binary representations of these shellcodes were used to embed them in benign documents for detection and identification purpose. List of these shellcodes and their description is attached as appendix.

5.2 Approach

As the Markov property states that the conditional probability distribution for the system at the next step (and in fact at all future steps) given its current state depends only on the current state of the system, and not additionally on the state of the system at previous steps, so it can be represented as :

$$P(Xi|Xi\text{-}1,Xi\text{-}2,Xi\text{-}3\dots\dots\dots Xi\text{-}n)=P(Xn|Xn\text{-}1)$$

Here I consider the simplifying assumption of N-gram models that the current state depends on a constant number of the preceding states. Here we used the assumption that each instruction depends on previous two instructions. The probability model for code thus looks as follows:

$$P(S) = P(X_i)P(X_{i+1}|X_i)P(X_{i+2}|X_i,X_{i-1})\dots\dots P(X_{i+n}|X_{i+n-1}X_{i+n-2})$$

$P(X_{i+1}|X_i)$ presents the conditional probability of next instruction X_{i+1} if the probability of previous instruction X_i is known.

$$P(S)= P(x_i)*(P(x_i \Pi x_{i+1})/P(x_i))*(P(x_{i+1}\Pi x_i x_{I+1})/P(x_i x_{i+1}))\dots\dots\dots P(x_{i+n}|x_{i+n-1}x_{i+n-2})$$

5.2.1 Algorithm

```
While(end of file){
          Sentence:=first 20 instructions from text file,
          Var S:=0;
          for i from 0 to 20 by 1{
                    $P= p(x_i)*(p(x_i \Pi x_{i+1})/p(x_i))*(p(x_{i+1}\Pi x_i x_{I+1})/p(x_i x_{i+1}))$
                    S:=s*P
                    }
          S:=1;
  }
```

5.3 Experiments

At the start of experiments we started by selecting few files from our data as a representative of their corresponding file types like different files generated by Microsoft office, PDF files and PS files.. The file types vary in size from few kilo bytes to several megabytes. These files were used as carrier of malware. Similarly the experiments were repeated with different type and size of shellcodes ranging from 20 bytes to 350 bytes.

In the following charts the results were obtained by using a same shellcode in different type carrier files. The shellcode used is “Win32 Download and Execute Shellcode Generator (browsers edition)” ,written by Yag Kohha , the size of shellcode was 275 bytes.

5.3.1 SHELLCODE Used for Following Test files

Win32 Download and Execute Shellcode Generator (browsers edition)

Size: 275 bytes + loading_url

Author: Yag Kohha (skyhole [at] gmail.com)

Usage: ./sco http://remote_server/loader.exe

Text File.

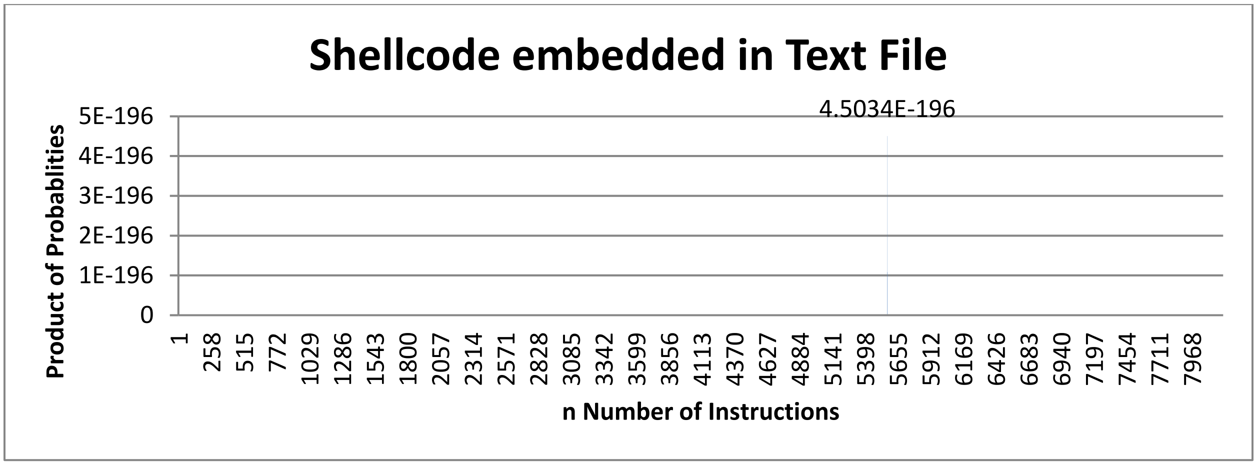


Fig 5.3.1 The Rfc3236.txt file is of size 17 Kb and shellcode 1443 of length 110 bytes is embedded at 22684th Index of file.

Docx File

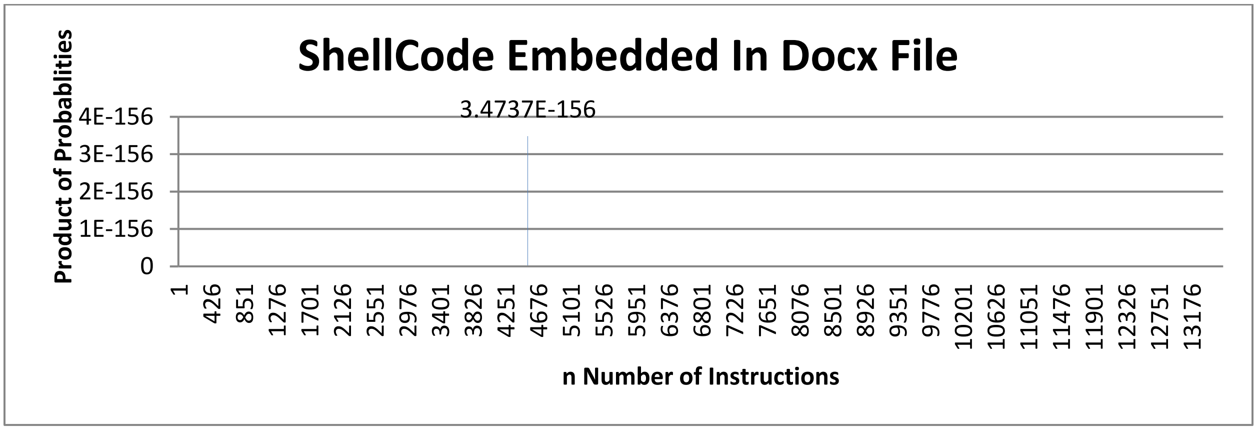


Fig 5.3.2. The PRIMER OF THE PHILIPPINE INDEX OF SPORTS TOURISM .doc file is of size 28 Kb and shellcode 4073 of length 124 bytes is embedded at the 18596th Index

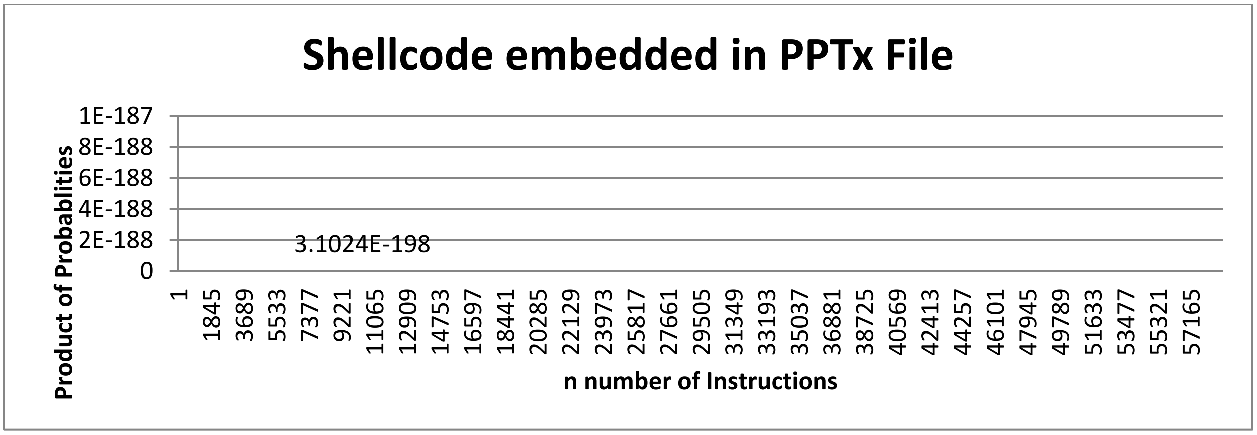


Fig 5.3.3. The Model Hockey Program.pptx file is of size 124 Kb and shellcode 1275 of length 318 bytes is embedded at 169233th Index.

PS File

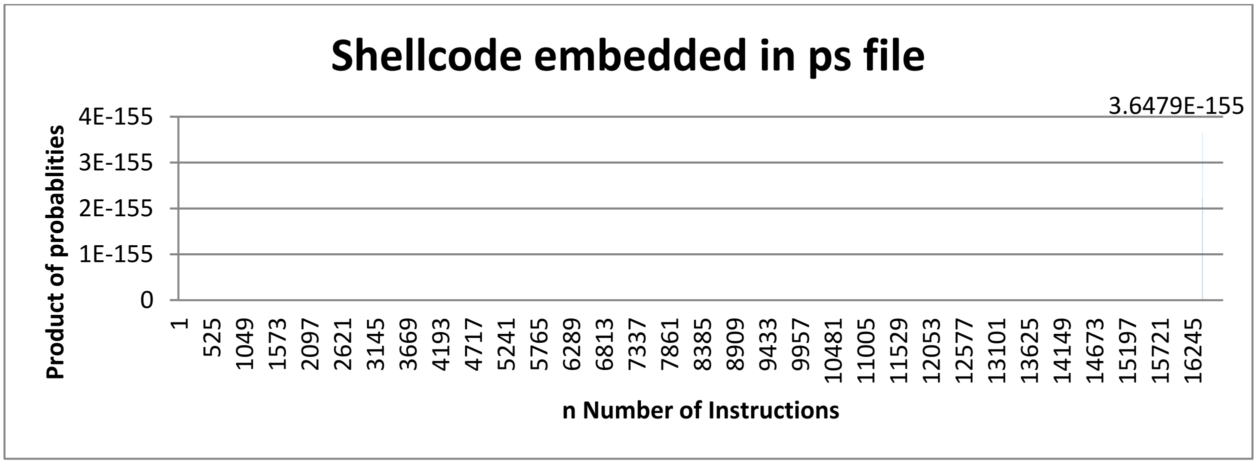


Fig 5.3.4. The Roots_of_lisp.ps file is of size 209 Kb and shellcode 1444 of 104 bytes is embedded at the 331386th Index.

PPT File

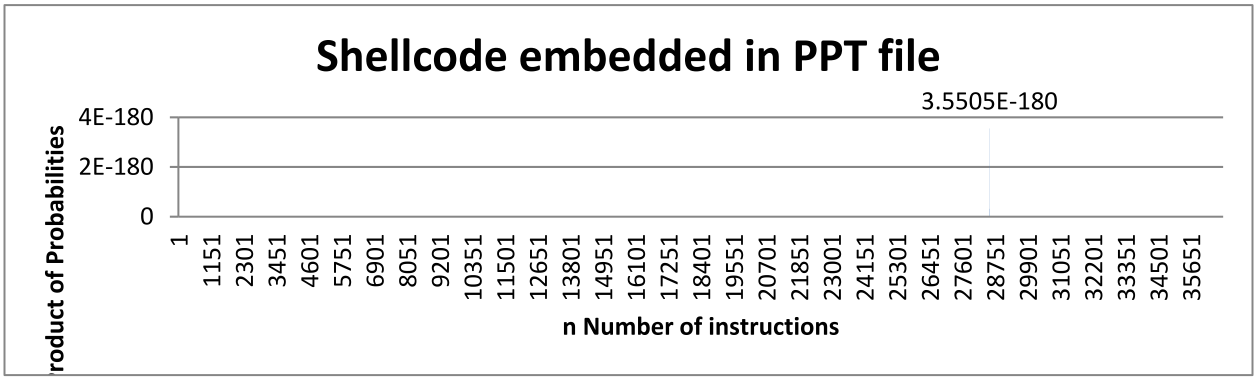


Fig 5.3.5. The InternetGovernance.ppt file is of size 105 Kb and shellcode 1155 of length 249 bytes is embedded at 174790th index.

PDF File

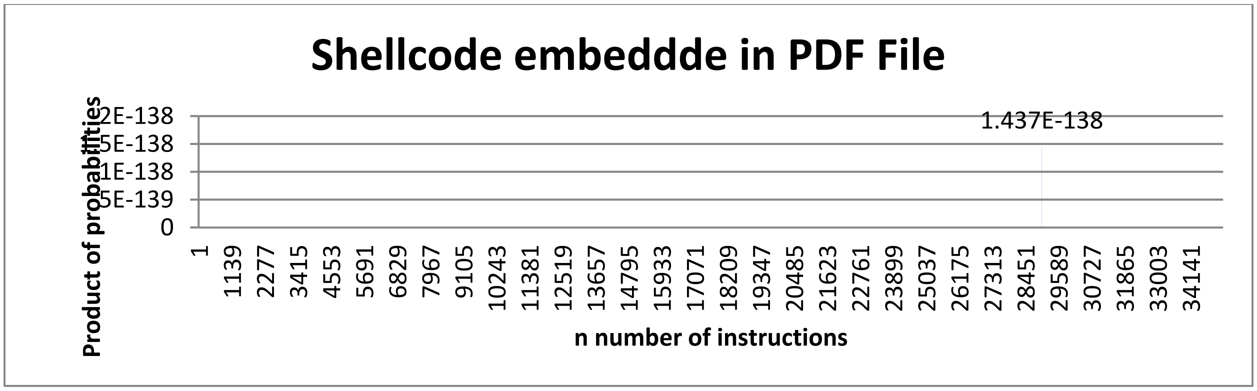


Fig 5.3.6. The File:Sybil attack.Pdf file is of size 98 kb and shellcode 7971 of length 57 bytes is embedded at 174366th index.

JPG File

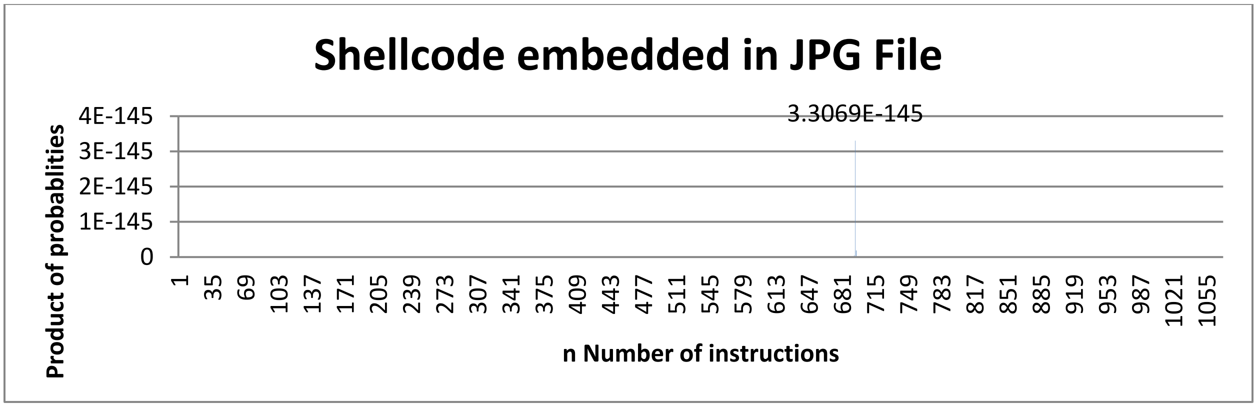


Fig 5.3.7 The English shaperd.jpg file is of size 3 Kb and shellcode 5251 of 275 bytes is embedded at 4189th index.

MP3 File

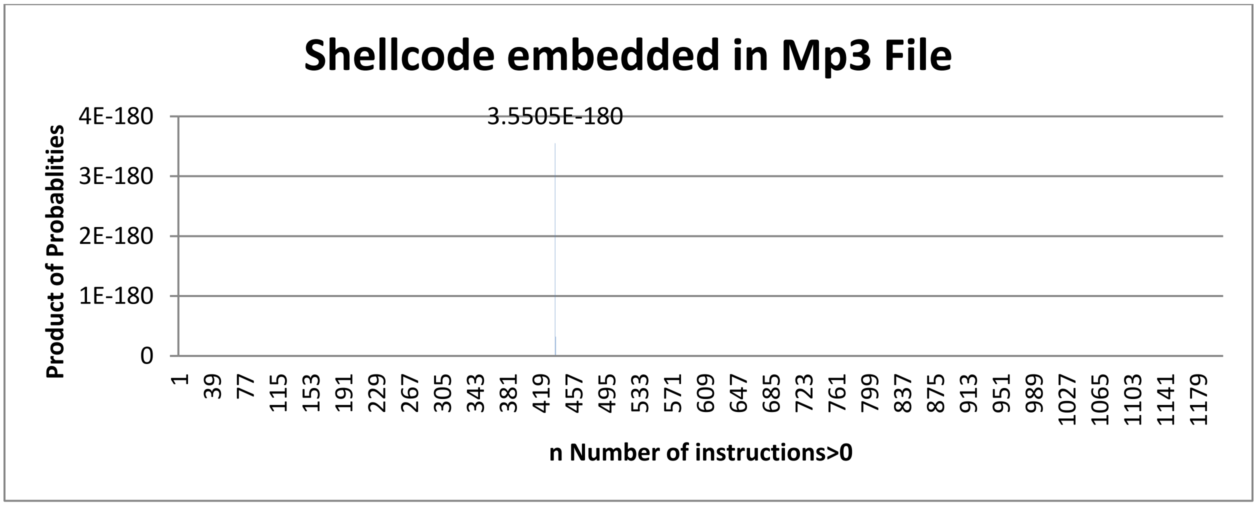


Fig 5.3.8 The File:B2k Jingle bell.mp3 file is of size 114 Kb and shellcode 1155 of 249 bytes is embedded .

5.4 Results

As it is obvious from the charts that in few file types like text, doc, docx, pptx the results seems promising to meet our goal of identifying and locating the exact position of embedded malware in the file .But at the same time the results are not satisfactory for PS and PPT files. The results were more ad less similar when we changed the carrier files and embedded shellcodes. The results seems to highly dependent upon type of carrier files.

The results could be explained by the sum of conditional probabilities of a set of trigrams were supposed to be high at the point where it matches the code model, and lower at the points in the file where it is significantly different from the code model.

In our experiments it can be noted that the sum of conditional probabilities of a set of trigrams in the file exceeds 0.01 at code points. In the plain text files it can be located simply because this value is around 0.005 for whole file. But if we carefully look at chart for the Doc file this sum is around 0.01 for the whole file and above 0.01 at beginning and at the end of the file. So if the shellcode code is embedded at some points in that region then it would be obviously difficult to trace, but could be traced easily if it were embedded in the middle. This shows that irrespective of the data contained in that files these files retains same trend for the sum of conditional probabilities of a set of trigrams irrespective of the data contained in the file, so at some points in the file structure these files coincides with the code model or it can be said as these portions of files are more near to code as compared to data. That’s the reason the results were highly dependent on file types.

For the ppt file most of the portion of the file the sum of conditional probabilities of a set of trigrams in the file touches the 0.015, higher than that for code, so the trend cannot be obvious in this situation.

5.5 Revising Code Model

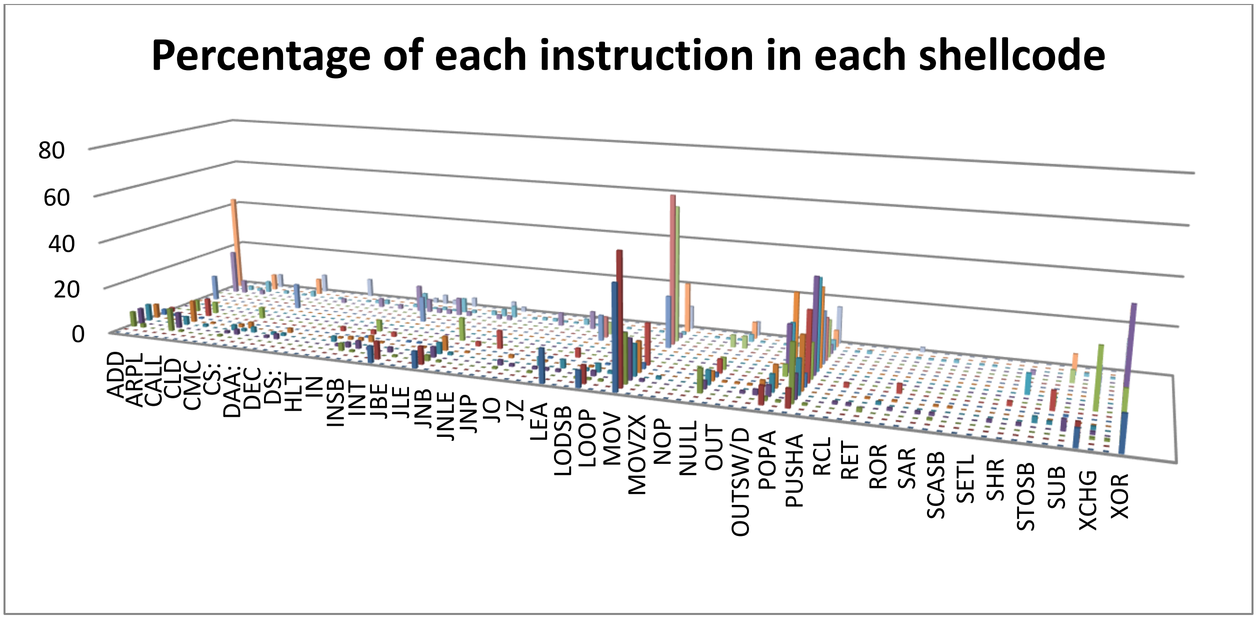
During the course of that experiments we observed that shellcodes seems slightly different from executable files. So we decided to build our code model from shellcodes instead of previous code model. Model was constructed in same way as before. Same set of experiments were repeated but I did not offer any improvement over previous results, mainly because the data obtained from shellcodes was not sufficient for building code model because of small sizes of shellcodes. As more than 80 instructions were choose for building code model and a simple matrix on order nxn need 6400 values. The data was not sufficient to build higher order matrix. So this does not seems a feasible solution.

5.6 Further investigation of Shellcodes

Form the results shown above we concluded that the information obtained by the above method is not sufficient to achieve the desired result. Then we started to look for the behavior of other elements of code other than just instruction sequence like the sequence of registers, more patterns of code in assembly language. May be any other feature could prove helpful.

5.6.1 Percentage of each instruction in each shellcode

The frequency of instructions found in shellcode was calculated**.** Mov ,push ,pop and xor are the most frequent instructions.


Percentage of each instruction in each shellcode
80
60
40
20
0
ADD
ARPL
CALL
CLD
CMC
CS:
DAA:
DEC
DS:
HLT
IN
INSB
INT
JBE
JLE
JNB
JNLE
JNP
JO
JZ
LEA
LODSB
LOOP
MOV
MOVZX
NOP
NULL
OUT
OUTSW/D
POPA
PUSHA
RCL
RET
ROR
SAR
SCASB
SETL
SHR
STOSB
SUB
XCHG
XOR

5.6.1 Percentage of Registers

The other most prominent element in assembly language codes others then instructions are registers. After considering the instruction sequences we focused on the frequency, type and sequence of registers.

The following charts showing the frequencies of individual instructions in shellcode set.

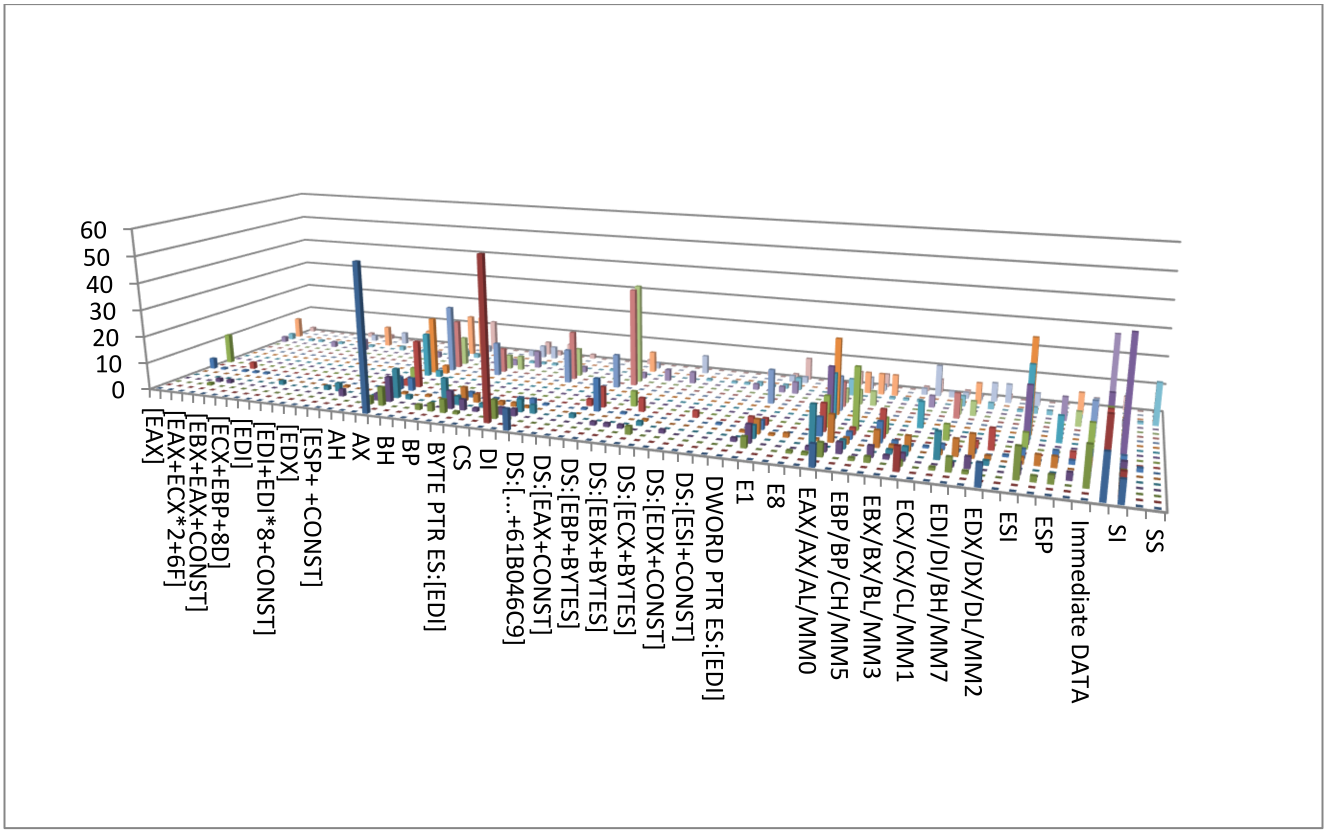


Fig 5.6.1 (a)Percentage of Registers

5.6.2 Ratio of Operands

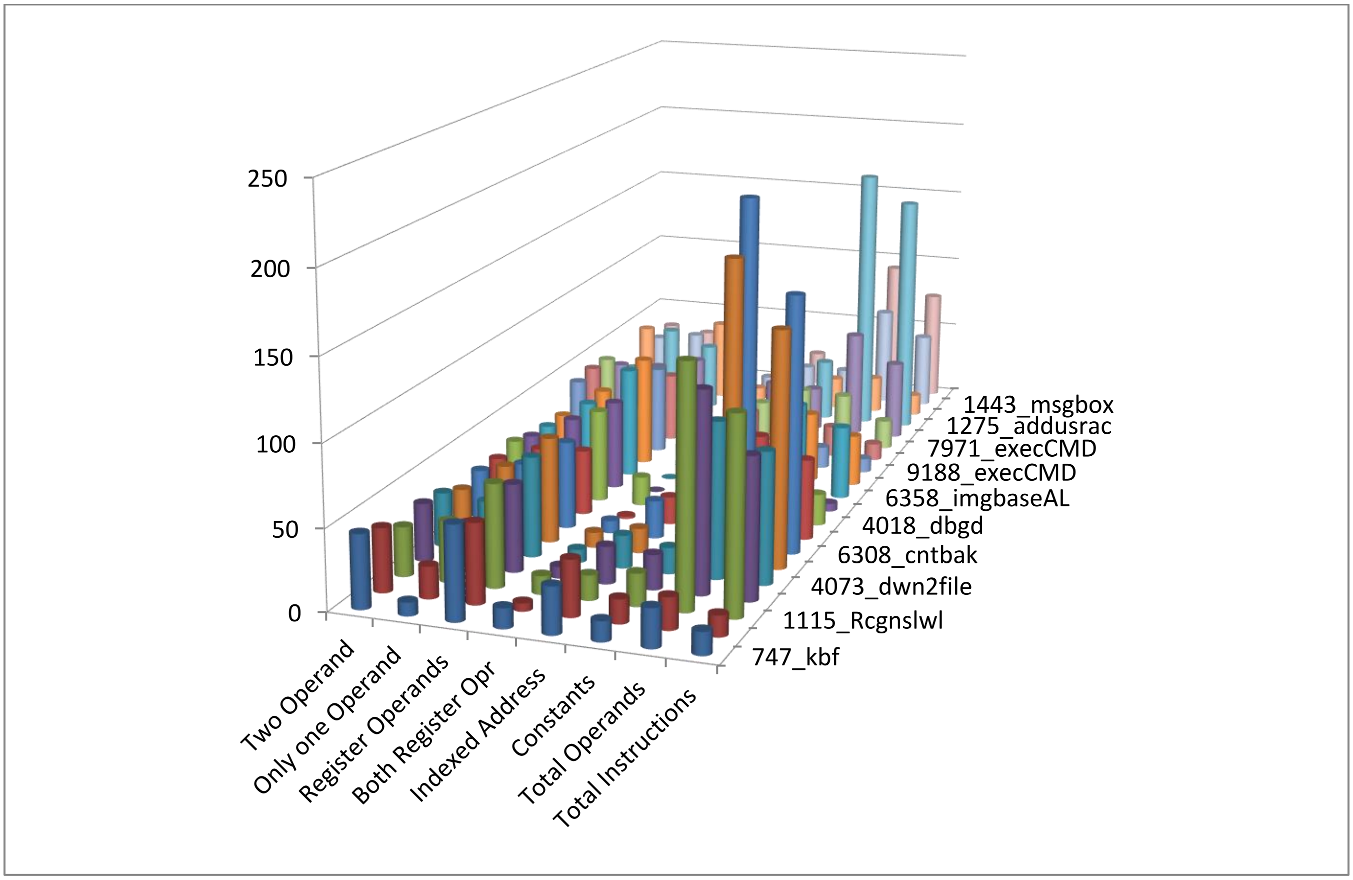


Fig 5.6.2 Ratio of Operands

Chapter 6

Conclusion and Future work

6.1 Conclusion

- According to “ A Fast Static Analysis Approach To Detect Exploit Code Inside Network Flows “by Ramkumar Chinchani1 and Eric van den Berg2 ,The self-correcting property of Intel binary disassembly is interesting because it tends to converge to the same instruction stream with the loss of only a few instructions. This appears to occur in spite of the network stream consisting primarily of random data and also when disassembly is performed beginning at different offsets. Convergence occurred in every instance but with different number of incorrectly instructions, ranging from 0 to 4 instructions. Also note that not all start bytes will lead to a successful disassembly and in such an event, they are decoded as a data byte.

  But in our experiments as we are using very small size shellcodes so their detection is somewhat position dependant, because due to small size they sometime fail to converge in original assembly so may produce wrong results few times, Although this problem does not exist when this method will be applied on actual data because attacker must take care of position of embedded code while embedding in benign document so that it could converge to right assembly.

6.2 Future Work

“Modern documents and the corresponding applications make use of embedded code fragments. This embedded code is capable of indirectly invoking other applications or libraries on the host as part of document rendering or editing. For example, a pie chart”.

“One should expect to see any kind of code embedded in a document. Since malcode is code, one cannot be entirely certain that a piece of code detected in a document is legitimate or not, unless it is discovered and embedded in an object that typically does not contain code.” [2]

- Modern documents and the corresponding applications make use of embedded code fragments.This feature needs to be investigated further to study the nature and features of these embedded codes .

Bibliography

[1] Clift, Neill,Wijeratna, Thushara K. ,“ Identifying malware that employs stealth techniques”, United States Patent :7743418, June 22, 2010 , http://www.freepatentsonline.com/7743418.html

[2] Vulnerability (computing), http://en.wikipedia.org/wiki/Vulnerability_(computing)

[3] Shellcode, http://en.wikipedia.org/wiki/Shellcode

[4] Ronald Volgers ,”Limits of network-level shellcode detection” In *10th Twentieth Student Conference on IT, Enschede,* 2009 [5] Wei-Jen Li, Salvatore Stolfo, Angelos Stavrou, Elli Androulaki, and Angelos D. Keromytis, “A Study of Malcode-Bearing Documents” In *Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment,* Lecture Notes In Computer Science Vol. 4579 , pp. 231 – 250, Springer Berlin ,2007

[6] Kevin Borders ,Atul Prakash , Mark Zielinski ,” Spector: Automatically Analyzing Shellcode” In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pp.501-514, Ieee Xplore,2008

[7] R. Chincahil and E. van den Berg, "*A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows*". In Proceedings of Recent Advances in Intrusion Detection (RAID 2006), Lecture Notes in Computer Science, Vol. 3858, pp. 284-308, Springer Berlin, 2006.

[8] Leyden, J,"*Trojan exploits unpatchedWord vulnerability*", The Register ,May 2006

[9] Evers, J,"*Zero-day attacks continue to hit Microsoft. News.com*" ,September 2006

[10] Kierznowski, D," Backdooring PDF Files",September 2006

[11] James Newsome, Brad Karp, and Dawn Song," *Polygraph:Automatically generating signatures for polymorphic worms*", In SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy, pages 226–241,Washington, DC, USA, 2005. IEEE Computer Society.

[12] Mihai Christodorescu and Somesh Jha,"*Testing malware detectors*", In ISSTA '04: Proceedings of the 2004 ACMSIGSOFT international symposium on Software testing and analysis, pages 34–44, New York, NY, USA, 2004.

[13] Carey Nachenberg" *Computer virus-antivirus coevolution*", Commun. ACM, 40(1):46–51, 1997.

[14] Wang, K., Parekh, J., Stolfo, S.J.: Anagram,"*A Content Anomaly Detector Resistant to Mimicry Attack*", In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, Springer, Heidelberg ,2006

[15] Li, W.-J., Wang, K., Stolfo, S.J., Herzog, B.,"*Fileprints: Identifying File Types by n-gram Analysis*", In: 2005 IEEE Information Assurance Workshop ,2005

[16] Stolfo, S.J., Wang, K., Li, W.-J.,"Towards Stealthy Malware Detection", In: Jha,Christodorescu, Wang (eds.) Malware Detection Book, Springer, Heidelberg ,2006

[17]Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.,"*Data Mining Methods for Detection of New Malicious Executables*". In: IEEE Symposium on Security and Privacy,Oakland, CA (May 2001),

[18] Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.,"*Detection of New Malicious Code Using N-grams Signatures*", In: Proceedings of Second Annual Conference on Privacy, Security and Trust, October 13-15, 2004

[19] Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.," *N-gram-based Detection of New Malicious Code*", In: Proceedings of the 28th IEEE Annual International

Computer Software and Applications Conference, MPSAC 2004. Hong Kong.September 28–30,2004

[20] Karim, M.E., Walenstein, A., Lakhotia, A.,”*Malware Phylogeny Generation using Permutations of Code*”,Journal in Computer Virology ,2005

[21] McDaniel, M., Heydari, M.H.: Content Based File Type Detection Algorithms.In: 6th Annual Hawaii International Conference on System Sciences (HICSS’03) (2003)

[22] Noga, A.J.: A Visual Data Hash Method. Air Force Research report (October 2004)

[23] Goel, S.: Kolmogorov Complexity Estimates for Detection of Viruses. Complexity Journal 9(2) (2003)

[24] Jedidiah R. Crandall, Zhendong Su, S. Felix Wu, and Frederic T. Chong,”*On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits*”, In CCS ’05: Proceedings of the 12th ACM conference on Computer and communications security,pages 235–248, New York, NY, USA, 2005. ACM

[25] Yingbo Song, Michael E. Locasto, Angelos Stavrou,Angelos D. Keromytis, and Salvatore J. Stolfo ,” *On the infeasibility of modeling polymorphic shellcode”,* In CCS ’07: Proceedings of the 14th ACM conference on Computer and communications

security, pages 541–551,New York, NY, USA, 2007. ACM.]

[26] M. Zubair Shafiq, Syed Ali Khayam, and Muddassar Farooq,” Embedded Malware Detection using Markov n-grams “, *In Lecture Notes in Computer Science* , *Detection of Intrusions and Malware, and Vulnerability Assessment* Vol. 5137/2008 , pp. 88-107 ,Springer,2008

[27] Michalis Polychronakis1, Kostas G. Anagnostakis, and Evangelos P. Markatos,” Network- Level Polymorphic Shellcode Detection Using Emulation” In *Lecture Notes in Computer Science*, D*etection of Intrusions and Malware & Vulnerability Assessment*, Vol. 4064/2006, pp.54-73 ,Springer,2006

Appendix A

Shellcodes

Shellcode ID	Description	Size
534	windows/XP-sp1 portshell on port 58821	116 bytes
747	PEB method for Windows 9x/NT/2k/XP	35 byte
777	C PEB kernel base location method works on win9x-win2k3 no null bytes,	31 bytes
1122	PEB way of getting kernel32 imagebase. Gives kernel32 imagebase in eax. only eax/esi used.	29 bytes
1155	Reverse Generic Shellcode without loader(no null byte)	249 bytes
1275	win32 useradd shellcode	318 bytes

1386	win32 download & exec shellcode	202 bytes
1443	Pops up Message Box Under Windows Xp SP2	110 bytes
1444	Relocate able dynamic runtime assembly code example using hash lookup	104 bytes
1675	win32 Beep Shellcode (SP1/SP2)	35 bytes
4018	Sets PEB->BeingDebugged to 0. IsDebuggerPresent()/BeingDebugged bypass	39 bytes
4073	win32 download and execute	124 bytes
5251	win32 Download and Execute Shellcode Generator (browsers edition)	275 bytes
6358	ImageBase Finder (Alphanumeric)	67 bytes
6359	PEB Kernel32.dll ImageBase Finder (Ascii Printable)	49 bytes
7971	Execute Cmd.exe Tested Under Windows Xp SP2	57 bytes
8078	Execute Cmd.exe Tested Under Windows Xp SP2	32 bytes

8103	Uses PEB method to determine whether a debugger is attached to the running proccess or not.	14 bytes
8122	payload:add admin acount & Telnet Listening	111 bytes
9188	Execute Cmd.exe Tested Under Windows Xp SP	23 bytes

Appendix B

Data Set of Carrier Files

File Name	Description	Size
The_Earth_seen_from_Apollo_17.jpg		100 kb
The_Scream.jpg	One of several versions of the painting "The Scream"	90 kb
Cat.jpg	A cat wearing watermelon shell	26 kb
English shaperd.jpg	A cute English shepherd	3kb
new_dash.pdf	Text ,Images, Tables, Multi Colored,64 pages	979 kb
Empsit.pdf	Text ,Graphs, logos ,Tables, Colored, 38 pages	207 kb

thehost_chapter4.pdf	Text ,punctuation, black, 18 pages	18 pages
sybil attack.pdf	Text ,formulas, figures ,black, 6 pages	98 kb
Foot.ppt	Text ,images, photographs, multi colored	1.8 Mb
Invisiblewarrior.ppt	Text ,images, multi colored	563 kb
The Periodic Table.ppt	Text , Table , multi colored	78 kb
The Need for Speed.pptx	Text ,Images, logos, screenshots, multi colored	3.71Mb
TL16.pptx	Text , Figures , multi colored	1.5 Mb
Cutting_trees_and_cutting_lemma_final.pptx	Text , Images, Figures, multi colored	329 kb
The Model Hockey Program.pptx	Text , Logo, multi colored	124 kb
Stateapp.doc	Text , Text boxes, lines, tables logos	366 kb

Ioannes_a_Cruce,_The_Dark_Night,_EN.doc	Text , Hyperlinks	466 kb
research_papers.doc	Text	34 kb
The Nine Satanic Statements.doc	Text	20kb
ResourceGov.docx	Text, Logos, images, tables, hyperlinks, screenshots, multi colored	499 kb
PRIMER OF THE PHILIPPINE INDEX OF SPORTS TOURISM. docx	Text	28 kb
Recycling for the Future-1_8-18-09.docx	Text	24 kb
The Commercial Agency Law.docx	Text	21 kb
abbott.txt	Text	71 kb
howl.txt	Text	24 kb
rfc3236.txt	Text	17 kb

1000.txt	Text	8 kb
Essence.ps	Text	192 kb
Iamc.ps	Text	2.15 Mb
roots_of_lisp	Text	209 kb
B2K_Jingle Bells.mp3	Music	114 kb
Backstreet Boys_Larger Than Life.mp3	Music	119kb
Bleedinglove.mp3	Music	936 kb

Appendix C

List of Executables used for code model

youtube_downloader_hd_setup1.9	2.16 MB
X12-30263	291 MB
windows-kb890830-v3.6	9.57 MB
winamp5572_full_emusic-7plus_en-us	10.4 MB
Vuze_Installer	8.06 MB
Vcredist	2.06 Mb

SkypeSetupFull-Beta	22 Mb
Registrybooster	3.88 Mb
PowerDVD9.1501D(Trial)_DVD081031-03	92.3 Mb
Opera1053	12.4 Mb
msgr10us10001267	409 KB
InternetDownloadManager	3.05 Mb
googletalk-setup	1.53 Mb
freecell-1.0-setup	438 Kb
FoxitReader30_enu_Setup	3.56 Mb

driverscanner	7.30 Mb
crt_x64	1.57 Mb
bittorrent6.4.18775	2.78 Mb
BeautyGuide-English	3.17 Mb
avg_avwt_stf_all_90_819a2842	103 Mb
AllInOneKeylogger	4.11 Mb