Buffer Overflow Security techniques in Windows: Development, Verification and Validation

A Thesis submitted
for the award of the degree of
Master of Engineering
in
Software Engineering

by

Parul Oberoi
(801131018)

under the supervision of

Dr. Maninder Singh
Associate Professor

Computer Science and Engineering Department
Thapar University
Patiala – 147 004, INDIA

June, 2013

Contents

List of Figures

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "**Buffer Overflow Security techniques in Windows: Development, Verification and Validation**", in partial fulfilment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Maninder Singh and refers other researchers work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Parul Oberoi

Parul Oberoi

(801131018)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Maninder Singh

(Associate Professor)

Countersigned by

Maninder Singh (Ph.D.)
Associate Professor
Computer Science and Engineering Department
Thapar University
Patiala-147004, India

S. K. Mohapatra (Ph.D.)
Sr. Professor (Mech. Engg)
Dean of Academic Affairs
Thapar University
Patiala-147004, India

Acknowledgement

I would like to express my deepest appreciation to Dr. Maninder Singh, my mentor and thesis supervisor for his constant support and motivation. He had been instrumental in guiding me throughout the thesis with his valuable insights, constructive criticisms and interminable encouragement.

I would like to thank all the faculty members and staff of Computer Science and Engineering department who were always there at the need of the hour and provided all the help and facilities, which I required, for the completion of this work.

I offer my deepest gratitude to my family for their support and affection and for believing in me always. I also want to thank my colleagues, who have given me moral support and their relentless advice throughout the completion of this work.

Abstract

With the ever increase in use of information technology based devices and gadgets, the development of softwares has become apparent. The functions which the software performs basically exhibits the needs and requirements of the user. Thus emphasis on the *software's well being is crucial.*Software building involves a lot of factors like functionality,ease of use, security, methods involved etc. One don't have to be a genius to deliver software on time, all the developer needs is to fulfill the requirements stated by the user. Keeping these requirements in mind any programmer can successfully develop and test the software.

But the problem arises when security of any software is audited. Many programmers do not keep security as the important aspect while developing these software which may create vulnerabilities. These vulnerabilities badly affect the system and may open back door for the intruders. Buffer overflow vulnerabilities continue to pose serious threats to network and computer security. By exploiting these vulnerabilities, a malicious hacker can strategically overwrite the return address of a procedure call, obtain control of a system, and subsequently launch more virulent attacks. Directly playing with the registers and memory management of the system the attacker can hijack the victim's system easily.

This thesis work exhibit the various Software countermeasures to safeguard the applications and systems against various forms of buffer overflow. At various levels the defense methods are created for such intrusions to be avoided. These may be modifications to applications, compilers, and operating systems. The various techniques are stack guard, SEH, SafeSEH, SEHOP etc. A DLL plugin to counter check an executable file is presented, which verifies the presence of secure buffer overflow technique usage in the application.

Despite the availability of these defenses, many systems remain vulnerable to buffer overflow attacks, which occur due to programming errors like bound checking, overlays, insecure software component usage etc. Utmost emphasis must be given by the pro-

grammers to produce safe and secure Software and to enhance, verify and validate these security measures, work presented in this thesis could be utilized.

Chapter 1

Introduction

The 21st century is the age of information. In every sphere of life, whether industries, businesses, educational institutes, social media etc using, sharing and processing information according to one's needs and usage has become vital. As information has become an important strategic resource in people's lives, its security is also crucial to adopt. The information obtaining, processing and security guarantee capability are playing critical roles in national power, and information security is adopted to improve the national security and social stability [39]. Information security refers to the processes and methodologies which are designed and implemented to safeguard print, electronic, or any other form of confidential and sensitive information or data from any unauthorized access, use, misuse, destruction or any type of modification. In short "Information security means protecting information and information systems from unauthorized access, use, exposure, disruption, modification, or destruction. With the ever increase in use of Internet and various applications over the Internet, securing information has become more important. The terms information security and computer security are frequently used interchangeably [46]. These are interrelated and share the same goals of protecting the integrity, confidentiality, availability, authenticity of information. Many different techniques are primarily used with different approaches to the methodologies used, and the areas of concentration. The approaches mainly include the Data Security, Devices Safety, Content Security and Behavior Security.

Information Security and Internet In today's world,Internet is inevitable source of information sharing. Hence transfer of information from one machine to another is very common affair. Internet is not a single network, but a collection of loosely connected networks globally that are accessible by individual computer systems in a variety of ways. These computer connections also involve gateways, routers, dial-up connections, and Internet service providers. The Internet is accessible easily to anyone with a computer and a network connection. Hence the transfer of information travels through a lot of devices so it requires serious concern to see the information is being transferred safely to the authorized user without the information being tampered in between. Every time a transfer of information occurs on the systems it may so happen that any intruder is planning to hijack that information and use it in a destructive way. It may be a case that the authorized user doesn't receive the information or receives distorted information hence accomplishing the intruder's motives. So to avoid the occurrence of such activities, security and vulnerability checks must be a crucial issue of concern.

Hence securing the resources required during the communication whether its hardware or software is important. But it may be possible that vulnerability exists in any resource which could hinder the security then some type of fix or patch must be provided and implemented temporarily but removing that vulnerability must be the permanent solution. Proper security checks and inspections may help in finding if any vulnerability exists or may occur in near future but that requires effort and time thus it should be followed that in the first go itself the vendors and programmers should create and provide vulnerability free software.

Elements of information security on internet

Security is associated with four core areas, which can be conveniently summarized by the acronym "CIAA". Any hacking event will affect any one or more of the essential area.

1. Confidentiality: Ensures that information is not accessed by unauthorized persons i.e. the data is only revealed to parties who have legitimate need. It means that information that should stay secret stays secret and only those persons authorized to

access it, may receive access. Unauthorized access to confidential information may have devastating consequences. Main mechanisms of protection of confidentiality in information systems are cryptography and access controls. Various threats to confidentiality are malware, intruders, social engineering activities, insecure networks [19].

2. Integrity: Ensuring trustworthiness of data or resources in terms of preventing improper and unauthorized changes. It ensures that the data and information send over the network is not compromised and tampered and hence the receiver gets the correct information. It aims at ensuring that information is protected from unauthorized alteration, modification or deletion.

3. Authenticity: Ensuring that a sender and a receiver of information are the persons they claim to be. There should not be any case that the attacker acts as one the communicator. It is an assurance that a message, transaction, or any type of exchange of information is from the source expected to be from.

4. Availability: Ensuring that the data or computing resources needed by appropriate personnel are both reliable and available in a timely manner. It aims to ensure that information is readily accessible to authorized users whenever required.

Impact of Software on Internet In the early usage of internet it was confined only to a few machines and people. The evolution of internet started with web 1.0 when the internet just took off. It was also known as - the Read Only Web. At this point of time internet was not at the all the same as its now. Web 1.0 has static web sites those were not interactive, were updated infrequently [14]. Visitors can only visit these sites could not impact or contribute to the sites. Most organizations had profile pages that visitors can look at but not impact or alter. The failure of web 1.0 because of its slow and clunky nature led to the emergence of web 2.0 and further web 3.0 and 4.0.

The working of internet requires Protocols, computers and communities, Information, servers and browsers, a packet-switched network using TCP/IP, various software etc. These softwares could be used to either set up Internet or to use Internet through various

softwares. Hence softwares are valuable part of internet setup and usage. Starting from setting a connection to the downloading and uploading of any page, softwares are the main ingredient. Software's usage has also increased day by day [48] . Not only in Internet but in every application of computers and other machines softwares are used like various music softwares, photo shop software, documentation software, research softwares, financial softwares etc. not only on personal computers, softwares are used at a high level institutes like research and development organization of any country, various defense organizations, corporate sectors, educational institutes, and various public and private sectors.

This means that softwares are even used in the areas where high precision and accuracy is required. For this the first and foremost need is to produce accurate softwares. A small error or vulnerability in the code may lead to bad results. For example consider a case that softwares are used in launching of a satellite, now this activity involves various tasks like to calculate the angle at which the launching is to be done, softwares are used to calculate the fuel required for combustion and many more tasks which are to be performed by softwares. Last but not the least the launching itself is done by softwares. So all these activities require utmost accuracy as a foreign body is to be placed and launched in the solar system. So any mistake of even minute degree or so may lead to some fatal results. So the software's accuracy is a must. Just imagine case that there is minor bug in the software, it may lead such bad effects which are beyond human imagination like as it happened in crashing of ARIANE-5.

Various activities on internet are performed and implemented with softwares itself. Hence the credibility of any software must be checked so as it doesn't contain any vulnerability of any sort. Vulnerability refers to the Existence of a weakness, design, or error on implementation that can lead to an unexpected, undesirable event compromising the security of the system. Through these vulnerabilities various intruders may try to or even get control of the machines being used a network. But it has been observed that programmers neglect the accuracy of software while developing it and run for the number of softwares created and developed in a particular span of time which affects the quality of software and further affects the security of Internet. If ever any software contains

any type of vulnerability it could harm the system in a very bad way. Various software vulnerabilities exists which are neglected many a times not by choice but by chance. For example, buffer overflow, integer overflow, memory exploitation, format string attacks, race condition, cross-site scripting, cross-site forgery and SQL injections [1]. Today, buffer overflow vulnerability face the majority of the exploits done by the intruders.

These vulnerabilities may provide remarkably easy gain of unauthorized access to intruder in an insecure networked environment. Even if users have nothing stored on the computers that is to be considered important, that computer can act as a "weak link", allowing unauthorized access to the organization's main systems and information. Information that intruders find useful includes which hardware and software are being used, system configuration, software in use, type of network connections, phone numbers, and authorization and authentication procedures [27]. Security-related information can enable unauthorized individuals to get access to important files and programs, thus negotiating the security of the system. Various important information are passwords,control files and keys, personnel information, and encryption and decryption algorithms etc. and these could be learnt easily if the vulnerability is exploited by the intruder. Various organizations affected through these vulnerabilities include banks and financial institutes, insurance companies, brokerage houses, consultants,defence, government departments, hospitals and medical laboratories, network service providers, utility organizations, the textile companies, universities and colleges, and wholesale and retail trades.

The consequences of a break-in cover a broad range of possibilities: a loss of time in recovering from the problem, a downfall in productivity, a significant loss of money or staff-hours, a huge loss of credibility or market opportunity, a business no longer able to fight with the present market, legal liability, and the loss of life. Thus developing good software is crucial.

Most of the above vulnerabilities exist because of the poor programming norms which programmers follow. The improper usage of various functions lead to many exploits to occur. It is known that the C programming language is the most popular language for system's programming and is used for the development of operating systems, embedded

systems, web systems and many other projects that require low-level access to hardware. It is a widespread programming language still used nowadays for numerous projects. Unfortunately, due to the lack of security awareness at the time of its development, there exist multiple functions of the C standard library exhibit serious security issues [42]. The most common security threat because of poor programming is buffer overflow or stack overflow. Although it may occur accidentally through programming error but buffer overflow is an increasingly common type of security attack on data integrity. In the list of "2011 CWE/SANS Top 25 Most Dangerous Software Errors", Buffer Overflow has bagged at the 3rd rank in software vulnerabilities [2] This shows that buffer overflow is very common and easy form of vulnerability which the intruders can conquer. Buffer overflow basically occurs because of Flaws in Software or Protocol Designs, Weaknesses in How Protocols and Software Are Implemented, Weaknesses in System and Network Configurations

Many techniques and tools have been used to overcome this vulnerability and many are still being under development. When vulnerabilities are discovered, computer vendors will normally develop patches to address the problem. However, it is up to the user, to get and install the patches, or properly configure the complete software to operate the system more securely [27]. It helps to understand that no single solution protects the system from a variety of threats. It always need multiple layers of security. If one fails, other layers still stand. Network security is accomplished through hardware and software. The software must be regularly updated and managed to protect from emerging threats. An effective network security strategy requires identifying threats and then choosing the most effective set of tools to combat them. But still all these tools and techniques come into action at a very later stage. The programmers must follow the security norms while writing code of any software so that there occurs no need of these tools to be used.

This thesis work presents the study and analysis of various Buffer Overflow techniques and showcases implementation of DLL to validate compile time usage of security switches and establishes malware detection using PE format.

Chapter 2

Literature Survey

In today's world of computers and computing, softwares are essential component to make the computer systems work. Since a computer system has to perform various tasks for the user so, many softwares are required. Each software has its known functionalities to execute. If a computer is to function, software can not be optional. Everything that a computer does,from the time, user turn the power switch on till the system shut down, everything is under the control of softwares. Hence it could be believed that softwares are prerequisites for the computer system to work as the way it does. Any Software is composed, created and developed in a particular programming language like C language, C++, java, .net platform etc. It could be decided upon by the programmer depending upon the type of field the software belongs or it may depend on the functionalities required for the software to perform. or it may be the language the programmer is comfortable in.

By far C programming language is the most popular language for system's programming and is used for the development of operating systems, system softwares, software applications and many other projects that require low-level access to hardware. This programming language is famous from a long time, and is still used in today's market.Unfortunately, Security awareness is the last focus of C language, Many functions of the C library display serious security concern [42]. If it is believed that the language possesses security issue then its more than obvious that softwares developed with that

programming language will also possess some type of vulnerability issues. But neglecting this aspect, still large number of vulnerable functions are used in many source codes of recent projects and system softwares. The most common security threat which occurs due to these vulnerable functions is buffer overflow or it could be referred as stack overflow. Small programming error even if occurred accidently may lead to many devastating results. Buffer overflow is an increasingly common type of security attack on data integrity. Buffer overflows cause in the order of 50 percent of all security alerts. Hence many techniques and tools have been and are being developed to overcome this vulnerability.

2.1 Significance of Secure Coding

The major problem which leads to such vulnerabilities is the careless attitude of the programmers towards the security of the code. Due to the deadlines and overburdened schedule, the programmers tend to finish the work haphazardly which gives birth to, so many software vulnerabilities. [4] Writing Secure Code must be the major emphasis while producing or developing any software. Many major faults and their evidences show that these source code vulnerabilities lead to major hazards. the software must be such that the users can rely on the integrity of the systems, and thus on the information that they hold and provide. Some of the insecure coding fatal examples are discussed below:

2.1.1 ARIANE 5 : Flight 501 Failure

On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure. Only after 40 seconds of flight's initiation,it crashed.Ariane 5 was designed by the European Space Agency (ESA) as a replacement for the successful Ariane 4 launcher. It was adopted from Ariane 4 itself. The intention was to create a reliable, high capacity, launch vehicle for ESA that could be used to support their contribution to the International Space Station as well as a range of other commercial and scientific launches. It took the European Space Agency 10 years and 7 billion dollar to produce Ariane 5, a giant rocket efficient

of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business [38]. All it took to explode that rocket less than a minute into its maiden voyage, scattering blazing debris across the mangrove swamps of French Guiana, was a small computer program trying to push a 64-bit number into a 16-bit space. One bug led to one big crash. Of all the careless lines of code recorded in the annals of computer science, this episode may stand as the most devastatingly efficient. From interviews with experts and an analysis prepared by the space agency, a it proved as clear path from an arithmetic error in software to total destruction. The Chain of Events which led to the failure:

1. The altitude and trajectory of the rocket are measured by a computer-based inertial reference system. This transmits commands to the engines to maintain attitude and direction.
2. The software failed in this system and the backup system also shut down.
3. Diagnostic commands were transmitted to the engines which interpreted them as real data and which swivelled to an extreme position.
4. Hence the rocket crashed.

The major cause of this was negligence in the coding of the software.Software failure occurred when an attempt to convert a 64-bit floating point number to a signed 16-bit integer caused the number to overflow. There was no exception handler associated with the conversion so the system exception management facilities were invoked. These led to the shut down the software [40]. The backup software was a copy and behaved in exactly the same way. Only a small inattentiveness led to such a crucial failure. thus it must be accepted that writing correct and secure code is the key to eradicate the software vulnerabilities.

The lesson learnt from such a case study is:

1. Don't run legacy software in critical systems unless it is actually needed.

2. As well as testing for what the system should do,it may also have to test for what the system should not do.

3. Do not have a default exception handling response in which system moves to shut-down state and have no fail-safe state.

4. Wherever possible, use real equipments and not simulations.

5. Improve the review process to include external participants and review all assumptions made in the code

Thus it is very important to check and recheck the code of any software so that no such incident like above happens in future as it is wastage of resources, time, man power, and lastly the software itself becomes a trash.

2.1.2 THERAC Failure

Therac-25 till date has been the most disastrous case of human error relating computer controlled radiation and human death to date. It called off 6 lives just because of a silly mistake done in the software of the system. The Therac-25 was a medical linear accelerator, a linac, developed by the AECL (Atomic Energy of Canada Limited) and CGR, a French company. It was the newest version of their previous models, the Therac-6 and Therac-20. These machines accelerated electrons that created energy beams that destroyed tumours. For the slight tissue penetration, the electrons were used; and to reach deeper penetration, the beam was converted into x-ray form depending on the dose requirement. The Therac-25 was a million dollar machine built to give radiation treatments to cancer patients [22]. This high energy radiation machine was controlled by a computer from a different room to protect the operator to perform any unnecessary doses of radiation but it behaved differently. Patients usually came in for a series of low energy radiation treatments to gradually and safely remove any remaining cancerous growth.

The Therac-25 had two main modes of operation: a low energy mode and a high energy mode which were used depending on the type of dosage to be given to the patient.

Therac-25 worked properly till some abrupt changes were made during its functioning like there appeared a case where the The technician by mistake typed "x" into the computer, which represented x-ray beam, then immediately realizing the error, changed the "x" into an "e" for electron beam, and hit "enter", the machine showed the ready state and hence it was started for treatment. This sequence occurred in less than 8 seconds and later the technician gave the beam command and the computer started to give dose to the patient. But the machine stopped thrice which the technician believed to be some startup error and still gave the dose. But the problem aroused when the patient felt some uneasiness and hence collapsed. It was thought the patient must be feeling bad because of the health conditions but the reason was, thrice starting the machine increased the dose three times, which took the life of the patient after 3 months [31]. As the commands were changed in such a short period of time, the computer did not respond properly leading to disastrous effects.

Their were many reasons for the therac's failure: poor understanding for properly assessing the old software when using for new machinery.The error and warning messages were not well designed. The system failed to fix or even understand the frequent recurring problems. Proper hardware machinery should have been installed to catch safety glitches. Manufacturer must have been accountable for any type of failure in the system. The terms and conditions required for the system must have been made clear in well advance.

If the above things were taken into consideration seriously then the therac's failure would never have occurred [23]. But still many recommendations were made to make therac-25 a success.

1. The source code of the system was corrected and test cases were checked.
2. All operators were informed not to restart machine without re-entering information.
3. Error messages will be made crystal clear to understand. Dose administered clearly shown to operator.
4. Taking these points into consideration therac was modified and used.

2.1.3 STUXNET

Stuxnet worm is the type of risk that is a threat to the national security as it causes harm to many activities deemed important to the basic functioning of modern society. The Stuxnet worm covertly attempts to identify and exploit equipment that controls a nation's critical infrastructure.Stuxnet is a large, complex piece of malware with many different components and functionalities. It is a threat that was primarily written to target an industrial control system like gas pipelines and power plants. Its final goal is to reprogram industrial control systems (ICS) by modifying code on programmable logic controllers (PLCs) to make them work in a manner the attacker intended and to hide those changes from the operator of the equipment. In order to achieve this goal the creators amassed a vast array of components to increase their chances of success. This includes zero-day exploits, a Windows rootkit, the first ever PLC rootkit, antivirus evasion techniques, complex process injection and hooking code, network infection routines, peer-to-peer updates, and a command and control interface.

Stuxnet was primarily developed to target the industrial nations like Iran, and its development institutes such as a gas pipeline or a power plant. The ultimate goal of Stuxnet is to sabotage that facility by reprogramming programmable logic controllers (PLCs) to operate as the attackers want them to response, and to make the system work in an unlikely fashion out of their specified boundaries. Stuxnet worm was discovered in June 2010 but it was believed that it existed even before that. It spreads over the network and in removable storage devices (USB) [34]. These targets are exploited through self-replication worms produced while using removable drives exploiting a vulnerability, allowing auto-execution which then interrupts in the proper functioning of the system. It also spreads in a LAN through a vulnerability in the Windows Print Spooler which has the printer overflow vulnerability. It copies and executes itself on remote computers through network shares and those running a WinCC database server. This shows that Stuxnet worm focuses on the basic functionalities and applications of any system which are every now and then used in any industry. It also contains a Windows rootkit that hide its binaries which are loaded to run the exploit. Its very critical feature is attempting to

bypass various security products.It also captures the Fingerprints of a specific industrial control system and modifies code on the Siemens PLCs to potentially demolish the system completely. It also hides modified code from the actual system .

The various vulnerabilities exploited by Stuxnet are: CVE-2008-4250 (MS-08-067) Windows Server Service NetPathCanonicalize() Vulnerability, CVE-2010-2568 (MS-10-046) - Windows Shell LNK Vulnerability, CVE-2010-2729 (MS-10-061)Windows Print Spooler Service Vulnerability, CVE-2010-2743 (MS-10-073) Windows Win32K Keyboard Layout Vulnerability etc [21]. The major exploit was made in Iran Gas pipeline and nuclear power plant. Stuxnet has also increased awareness of the vulnerability of industrial control systems, which haven't been the target of many attacks. This resulted in becoming more hardened against these attacks as at any point of time the attacks can be made.

All the above vulnerabilities and corresponding exploits depict that softwares are the most targeted component of any system. Whether the vulnerability occurs due to insecure source code functions or due to creation of new worms over insecure networks or due to any 3rd party attack, the main target which gets destroyed is the system.

In the above cases 3 different fields like Medicine, Rocket Science and nuclear plant faced 3 different challenges related to the software vulnerability. In all the above softwares get destroyed but the main target becomes the mankind. Thus it becomes very important to safeguard the systems by safeguarding the softwares.

2.2 Buffer Overflow Vulnerability

As the name suggests buffer overflow deals with the buffers and various arrays used in any programming language. It primarily occurs in memory components like registers. Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and

the storage for controls (e.g. return addresses). An overflow in the data part can affect the control flow of the program, because an overflow can change the return address [10]. Buffer overflow is ever increasing these days as the previously created softwares which contain Buffer Overflow vulnerability, face the challenge to eradicate this problem in their counter components. The new techniques being developed could be implemented in the softwares being created fresh as in old softwares it requires modifying all its components to remove this vulnerability, but it is a tedious task to do. Buffer overflows have been the most common form of security vulnerability for the last ten years. More over, buffer overflow vulnerabilities dominate the area of remote network penetration vulnerabilities, where an anonymous Internet user seeks to gain partial or total control of a host. If buffer overflow vulnerabilities could be effectively eliminated, a very large portion of the most serious security threats would also be eliminated [4].

The reason of buffer overflow vulnerability popularity is always wondered. It could simply be that Attackers are getting cleverer and are defeating ever more clever countermeasures. Attacks are getting easier to do, by script kiddies. Its not difficult to find any shell script and implement just the attacker must know some of the know how about registers and to manipulate them while taking into consideration the programming language under usage. This is the main reason that the Programs written in C are particularly susceptible to buffer overflow attacks. Space and performance were more important design considerations for C than safety and security. Hence, C allows direct pointer manipulations without any bounds checking [8]. The standard C library includes many functions that are unsafe if they are not used carefully. Nevertheless, many security-critical programs are also written in C these days to avoid such vulnerability to occur.

2.3 Memory Map of a Process

To thoroughly understand the concept of Buffer Overflow it is important to grasp the memory management of a program whenever any application is compiled and run. The various terms used in Buffer overflow must also be clear.Buffer overflow principle must be properly comprehended so as to eradicate the problems by applying various counter

measures. Buffer Overflow occur at register level of any computer system, so the basic hardware and memory architecture of the system must be clear in all respects .It is also referred that Low level software security starts by eradicating Buffer overflows. The various terms and their usage involved in buffer overflows are explained in brief below:

Text and data are the two logical parts in which any application or program can be divided. Text is the actual read-only program code in machine-readable format where instructions are to be performed and data is the information on which the various text operations are executed. Text data resides in the lower areas of a process's memory allocation [12]. Several instances of the same program can share this memory area. The content in the text area is not manipulative.

Data can be divided into the three logical parts of static, stack, and heap data. The distinction between these types is varied upon when and how the memory is allocated, and where the data is stored or located. When an executable is first loaded by the operating system, the text segment is loaded into memory first. The data segments then follow. Figure 2.1 represents the basic structure of a program's memory.

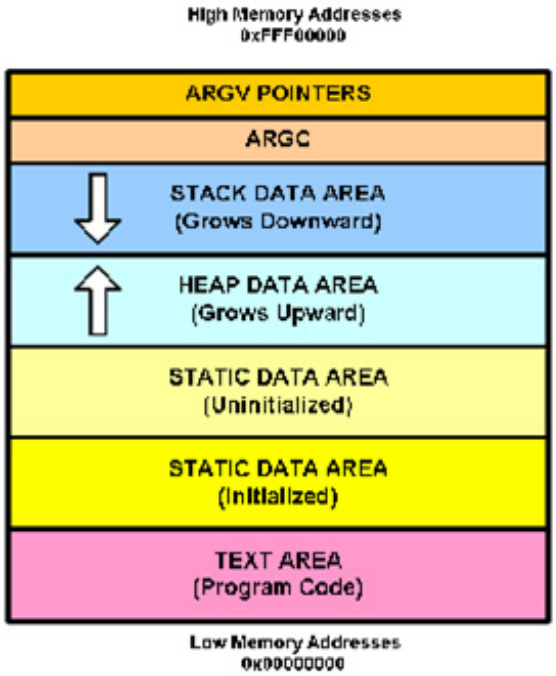

Figure 2.1: Memory Management [12]

Text Area: Code segment contains the code executable or code binary. This segment is loaded into the memory first at the lowest address. Static data, located above and

adjacent to the text data, is pre-known information whose storage space is compiled into the program. This memory area is normally reserved for global variables and static C++ class members. Static data can either be in an initialized or uninitialized state. Heap data, located above and adjacent to static data, is allocated at runtime by the C language functions malloc() and calloc(), and by the C++ new operator. The heap grows up from a lower memory address to a higher memory address.

Heap: When program allocate memory at runtime using many dynamic functions like malloc, Calloc, or new operator then memory gets allocated in heap. When new variables are provided memory then the heap grows upward as shown in Figure 2.1.The heap is simply the memory used by programs to store global variables. Elements of the heap (variables) have no dependencies with each other and can always be accessed randomly at any time. the memory provided to stack is much more than heap. Accessing this memory is bit slower.

Stack: A stack is an abstract data type used in computers. A stack works on the principle of LIFO (last in first out) that the objects placed last on the stack will be removed first. Two of the most important operations performed on the stack are PUSH and POP. PUSH adds an element at the top of the stack. POP, in contrast, reduces the stack size by one by removing the last element at the top of the stack. Stack is used in the computer systems whenever any calculations and operations on various variables are to be performed because of which the size of stack keeps on changing every now and then. Stacks are used like in the conversion of postfix to infix etc [12]. It stores local variables and is used for passing arguments to the functions along with the return address of the instruction which is to be executed after the function call is over. Whenever any new function is called and executed a new stack frame is created. A register called the stack pointer (SP) points to the top of the stack. The bottom of the stack is at a fixed address. Its size is dynamically adjusted by the kernel at run time.

Registers:Registers are either 16 or 32 bit high-speed storage locations directly inside the CPU, designed for high-speed access. These are basically used in the processing and execution of the data while some program is in progress. Registers can be grouped into

the four categories of Data, Segment, Index, and Control. Each register has their own significance.The segment registers CS, DS, ES, and SS are used as base locations for program instructions (text data), data (static and heap data), and the stack.The index registers EBP and ESP contain offset references to the code, data, and stack registers. They are, in effect, a compass or positioning service that allow the program to keep track of exactly where all of its data and instructions are located. The data registers contain actual data bits and are used for the movement and manipulation of this data. EBX is used for holding the address of a function or variable [12]. EBX plays a vital role in the exploitation of a buffer overrun. The control registers are bit-wise storage units used to alert the program or CPU of critical states or conditions, within the data or the program itself. EIP is of special importance in that it contains the address of the next instruction to be executed. Again, this is a crucial element in the exploitation of the buffer overrun.

Vulnerability:It refers to the presence of a weakness, in design, or implementation error that can lead to an unexpected, undesirable results compromising the security of the system. Security is a state of well-being of information and infrastructures in which the possibility of successful yet undetected exploitation, distorting, and disruption of information and services is kept low or tolerable. This security is affected if any type of vulnerability exists. This leads to the misuse and tampering of data and systems in use.

Exploit:A defined way to breach the security of an IT system is through vulnerability.An exploit is a chunk of software, a piece of data, or some sequence of commands that takes advantage of a bug, or vulnerability present in the program in order to perform unintended or unanticipated activities on computer software or hardware resources. Such behavior readily includes things like gaining control of a computer system or allowing privilege escalation or a denial-of-service attack. The exploits can either be local or remote [3].Maximum number of exploits are designed to gain root-level access to a computer system. The intruder creates such exploits which are easy to implement and provide higher access level of the system. However, it is also possible to use several exploits, firstly to attain low-level access, then to escalate privileges till the intruder could behave like a root user.

Shell code: Whenever the exploit is to be done to attain privileges, the code that has some pretty specific characteristics to make the exploit work is created. Since this code is being inserted into a program is used during its execution directly by the systems CPU. The shell code provided has to be machine code which is specific to the particular type of CPU used. In addition, the code will be sticking into memory and is executed from a location that is not known to the user, so the code provided must be position independent in other words it must be able to execute correctly regardless of where it ends up [35]. Code with these features is used as part of an exploit and is commonly referred to as shell code, so called because this type of code is written to provide the intruder with the shell access of the exploited systems.

Exe- The Portable Executable (PE) format is a file format for executables, object code, DLLs, Font files, and others used in 32-bit and 64-bit versions of Windows operating systems. The PE format is a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code. The inclusion of various dynamic library references for linking, API exports and import tables, resource management information and thread-local storage (TLS) data is very crucial. Windows uses the Portable Executable Format to store executable files, also known as an "image" of an executable. Although the PE file contains all the information required to "run" a program, the PE file must first be parsed, processed and loaded into memory. This process involves allocation of memory, relocations, imports, etc. Thus, the PE file is simply an "image" of the executable, the executable being referred to the program in memory. Any application run on windows system is exe file.

Debugger- A debugger is a computer program that is used to test and debug other programs (the "target" program). They also perform functions such as running a program step by step (single-stepping or program animation), stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint,tracking the values of variables, Object file scanning-locates routines from object files and libraries [43]. Debuggers have the ability to modify program state while it is running. Debuggers offer two modes of operation-full or partial simulation-to limit this impact. One of the debugger used it this thesis is OllyDbg. It is a 32-bit assembler-level

analyzing Debugger for Microsoft windows with good interface, very useful if the user is debugging an application and does not have the source code. It shows the complete detail of an exe file that how its registers are used, how is the stack stored, the various commands being used etc.

It emphasis on binary code analysis makes it particularly useful in cases where source code is unavailable. Its main features are : Intuitive user interface, no cryptical commands. It helps in code analysis like tracing registers, working with procedures, loops, API calls, switches, symbols tables, constants and strings. It can directly load and debug DLLs. It does object file scanning, locating routines from object files and libraries, allowing user-defined labels, comments and function descriptions citeOLLY. It understands debugging information in the Borland format. It Saves patches between sessions change, which is written back to executable file and updates fixups. It has a Open architecture and many third-party plugins are available. It shows the dissembler, debugger View, Memory View, Registers View, File Patcher, Strings List.Figure 2.2 shows a screen of ollydbg.

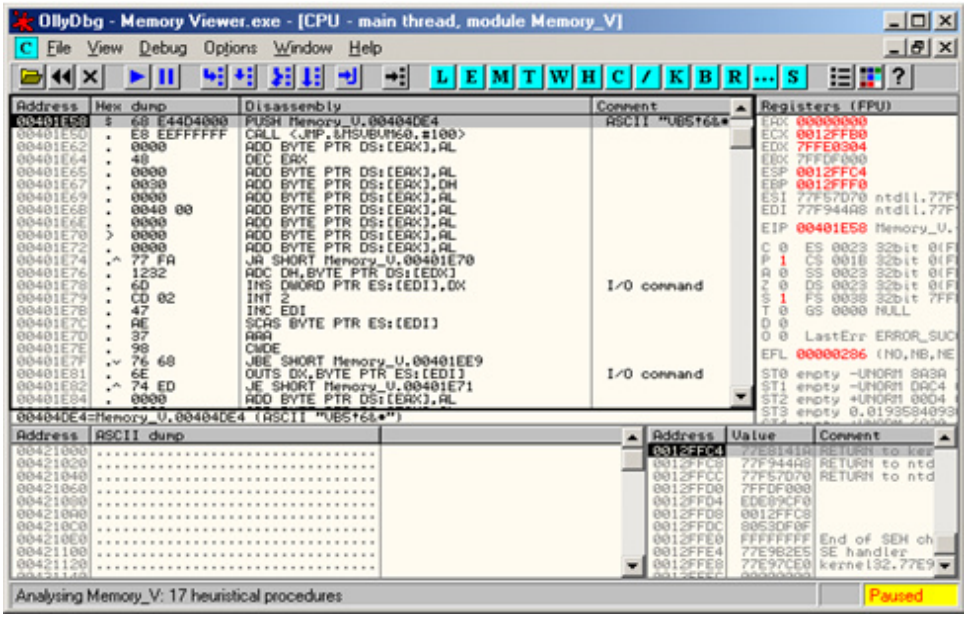

Figure 2.2: OllyDebugger

NOPs: NOP is an abbreviation used for No Operation in an assembly language instruction, computer programming language statements or register commands that effectively does nothing at all."NOP" Instructions are normally used at the start of the buffer to make the shell code land smoothly. After NOPs, the attacker places the shell code,

then the address of this shell code. As it is known before, since it is difficult to know the exact address of the Shell Code, some NOP instructions are padded to the beginning of the buffer to increase the likelihood that the execution jumps to some location near to the shellcode.It is usually used to delay execution for purposes of timing [25]. Taking its advantage and filling half of the overflow buffer with NOPs brightens the chances of attacker to attain the control of the victim system. Placing the shell code at the center, and then following the return addresses makes the attacker to create a good shell code. If by luck, the return address points anywhere in the string of NOPs, its execution will just reach to the code thus fulfilling the attacker's purpose. In the Intel architecture, the NOP instruction is one byte long and it translates to 0x90 in machine code.

2.3.1 Buffer Overflow Principle:

A buffer overflow occurs when a program or process tries to store more data in a buffer (temporary data storage area) than it was supposed to hold. Since buffers are created to contain a finite amount of data, the extra information if added, has to go somewhere, can overflow into adjoining buffers, corrupting and overwriting the valid data held in them. A buffer overflow attack can be performed by any malicious user to exploit an unchecked boundary conditions of a buffer in a program and then overwrites the program code with malicious data. If the program code is overwritten with new executable code, the result is, it changes the program's operation as directed by the attacker [12]. If overwritten with other data, the likely effect is to cause the original program to crash.

Buffer overflow attacks form a substantial portion of all security attacks simply because buffer overflow vulnerabilities are so common and so easy to exploit [4]. However, buffer overflow vulnerabilities particularly dominate in the class of remote penetration attacks because buffer overflow vulnerability present the attacker with exactly it needed. The ability to inject and execute attack code. The injected attack code runs with the privileges of the vulnerable program, and allows the attacker to bootstrap whatever other functionality is needed to control the host computer. The first buffer overflow or stack smashing was observed and conducted by Aleph One in 1988 in the paper titled : Stack

Smashing for fun and profit" [35]. A buffer overflow is the result of stuffing more data into a buffer than it can handle. Thus the main cause of Stack-based buffer overflows are by programs that do not verify the length of data being copied into a buffer.

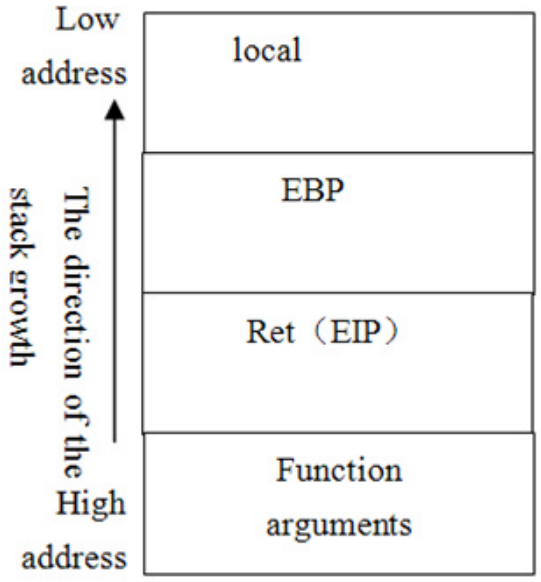

Figure 2.3: Stack during execution of function [15]

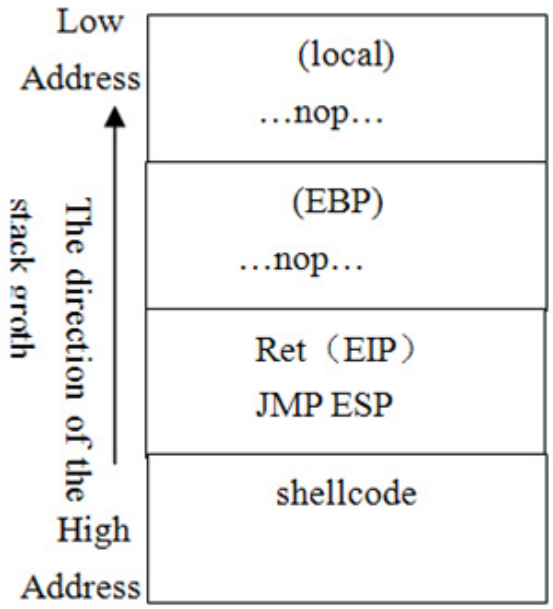

Figure 2.4: Stack after Buffer Overflow [15]

During a function call being executed, it uses various types of registers to execute and process the data. The registers used include EAX, EBX, EIP, ECS etc. Each register has its own significance. Whenever any function is under threat by the attacker, the attacker always targets the register values, tries to manipulate them and thus enjoys the

control of the system through the hijacked privileges from the targeted application. The overwritten data changes the value of the EIP register which further changes the flow control of the program. This always corrupts the working of the stack, that often causes the program to crash or operate incorrectly [15]. Stack overflow attack is to corrupt the stack in such a way so as to inject executable code (Shell Code) into the running program, which will overwrite the critical adjacent data, such as the function return address, the function pointer and so on, and to take control of the process. This is one of the oldest and most reliable methods for attackers to gain unauthorized access to a computer. Figure 2.3 elaborates the stack during a function call and Figure 2.4 shows the changed values after buffer overflow is executed through JMP ESP technique.

2.4 Types of Buffer Overflow

The overall goal of a buffer overflow attack is to subvert the function of a privileged program so that the attacker can take control of that program, and if the program is sufficiently privileged, the attacker holds the system [4]. Typically the attacker is attacking a root program, and immediately executes code same as "exec(sh)" to get a root shell, but not always. To complete the attack, the attacker must achieve two sub-goals:

1. Arrange for suitable code to be available in the program's address space.
2. Get the program to jump to that code, with suitable parameters loaded into registers and memory.

There are many forms of buffer overflow, each with different causes and results, but most forms are potential security threats to a computer system connected to a network as hackers can (and often do) use the side-effects of the overflow to attempt to take control of the system.

Stack Buffer Overflow The stack is where the computer declares and initializes the variables used in a computer program. In a stack buffer overflow, large amount of data

is written to the stack than it can handle, causing the stack to be overwritten, including the "return pointer" that tells the computer where to go once it finishes processing the stack [20]. The attacker tries to control the EIP as it tells the address of next instruction to be executed. Hence the hacker controlling the EIP can therefore use a stack overflow to rewrite the return pointer and direct the computer to malicious code. Stack smashing, a technique first described in detail in the mid 90s by hackers as Aleph One which is implemented by these two steps: changing the flow of control to execute attacker-supplied code.

The basics of a stack overflow is simple: Consider there are two buffers, a source buffer containing faulty input (assuming to be the attacker), and a destination buffer(assuming to be victim) that is comparatively smaller than the attack input. The second buffer is present on the stack and is adjacent to the function's return address on the stack. The false code does not check that the source buffer is too large to fit in the destination buffer. It copies the attacking code to the destination buffer, overwriting crucial information on the stack (such as the function return address) [36]. When the function returns, the CPU unwinds the stack frame and pops the modified return address from the stack.Control does not return to the original function as expected. Instead, malicious code created by the attacker when crafting the initial input is executed, which provides some type of extra facilities to the attacker to access the destination system.

There are a number of counter measures that are used to prevent stack smashing vulnerabilities. The major root cause for this vulnerability is programming flaws, removing those may avoid the problem to occur. Some points to be taken under consideration:

1. Security must be a top priority of the development efforts, even if it conflicts with the features planned to develop and implement under the specified schedule.

2. Appropriate code review techniques must be implemented to ensure the quality of code and try to improve the quality of the software development process for the in depth achievement of security.

3. Safe Programming languages like Java and Ada95 must be preferred which are

not vulnerable to buffer overflows, for a new product, or to port your existing application to such languages [33].

4. Perform proper bounds checking on all input data and in loops processing data.

5. On top of using, appropriate technical means like libsafe, avoid the implicit problems of C-functions like strcpy() and strcat() by properly using their bounds checking equivalents, in this case strncpy() and strncat()

6. Various techniques like stack guard, SEH, safe SEH, SEHOP are also used to safeguard the stack.

Heap Buffer Overflow A heap buffer overflow occurs when too much data is written to the portion of memory allocated to software for storing the software's data while it is running. Heap buffer overflows often leads to a system crash due to data corruption, as the software is overwritten with new values while it is running, or due to the execution of malicious code which is written into the heap buffer during the overflow and has thereby bypassed the computer's standard security system [16].Heap attacks are typically harder to perform than a Stack based attack because the presence of an overflow is not the only factor that determines the success, as the complete corruption of information on the heap is treated as its overflow. Only overwriting the data does not fulfills the intruder's purpose of overflow.

The two main ways by which heap overflow could be performed are: Overwriting the data stored in heap or by overwriting the pointer. Heap Overflow in practical example could be, changing things like the CD Key stored in a variable to a value that must be known so as to find out the granted access of a user or it could be changing things like a pointer that points to some specific code that is executed and that could be used to change the actual path to follow the malicious code specified by the attacker.

When the Heap memory is allocated, the programmer first determines the amount of space needed to load and execute the program. Heap header is used to store this information and rest of the data that may be required for the specific operating system, is also written on the Heap header. After this the heap is filled with the necessary and

required data [16]. If the programmer under estimates the amount of data to be placed into the specified space, the adjoining heap header and/or heap could be overwritten, it will result in a crash.This is most common way with which heap could be exploited to give an attacker control over a computer.

The counter measures of Heap overflow include the same protection techniques that are used for stack safeguarding. Other than this the PaX kernel patch for various systems stands out, since it offers not only a non-executable stack, but can also render the heap non-executable [33]. Though, that being able to overwrite a function pointer of suitable type and being able to tamper with the parameters used to call this function may still compromise a system hardened with PaX in a return-into-libc like fashion.

Off-by-One Errors Another very common programming error is, known as off-by-one error,which is to exceed the buffer size by just one byte. Typically this happens in loops that try to process all buffer elements. So this can duly be regarded as the minimal possible buffer overflow. An off-by-one error is a specific type of buffer overflow that occurs when a value is one iteration off what it is expected to be. This can often be due to miscounting the number of times a program should call a specific loop of code [33]. Programmers often forget that a string must include a null byte which terminates the string. Many common string operations, although bounded, does not add this character by default, thus allowing the string to enter in some other buffer's boundary on the stack, with no separation. If this string is used again later, it may treat both buffers as one, as it is expecting a null character at the end of buffer. The error may result in rewriting of one digit in the return pointer in the stack, which allows a hacker to guide the pointer to an address containing the malicious code. An "off-by-five" error was reported for sudo in 2002 (CVE-2002-0184), but that is more like a "length calculation" error.For example : This problem may be encountered in scanf function [7]. If the string is exactly 256 bytes long, the null byte terminator will be placed as first character in the specified buffer thus affecting the buffer.

The countermeasures against this attack are almost the same as for stack smashing attacks but it is at least one order of magnitude more difficult for programmers to take

precautions or to spot vulnerabilities while reviewing code. Note that the technical countermeasures StackGuard, Stack Shield, Microsofts /GS compiler switch and non-executable stack are effective against this attack as well. Incidentally, in this case the minor difference of canary/cookie setting between StackGuard and the /GS switch will result in earlier detection of vulnerability.StackGuard places a canary word next to the return address on the stack [33]. Once the function completes its execution, the protection instrument checks to make sure that the canary word is unmodified before jumping to the return address. If the original canary word is compromised, it shows something fishy occurred and thus the program will terminate.

Arithmetic Overflow In the C programming language it is possible to create a signed or unsigned variable. For example, the Calculator on Windows platform provides a perfect example of how a program will not always do what it is expected to do. To illustrate, open up the calculator, click 'View then Scientific', to ensure the necessary options are displayed. Now, type in '0 1' and hit '='. it should show a '-1' value. With the '-1' value in the results window, hit the 'Hex' radio button on the left [16]. It must now display the long string of 'FFFFFFFFFF'. In the programming world, it demonstrates what a signed value of '-1' looks like in Hex. The gist is that when the calculator is switched back to 'Dec' mode, or decimal format, the result is not a '-1'. Instead get a very large number of '18446744073709551615'. All this happened because the Windows calculator switched the calculation from a signed value to an unsigned value. The main point here is that programmers can make the same error. The result is a negative number quickly becomes a very large value that is enough to overflow a buffer. The results are that an attacker can easily control the size of the inputted data and create a dynamic buffer overflow condition.

Buffer Overrun A buffer overrun occurs when too much data is sent to the small block of buffer memory used by CD and DVD burners. These buffers exist to provide a steady flow of information from the computer to the device. Data is read from the buffer at a specific speed and must be fed into the buffer at the same speed, otherwise data is overwritten before it is used [20]. This results in file corruption and unsuccessful burning.A buffer overrun attack was one of the mechanisms reportedly utilized to deploy

the malicious agents used on the Solaris-based servers in the recent DDoS attacks.The key find out buffer overrun is to use static analysis. it is tedious but very effective method. One of the primary replication mechanisms was exploitation of a buffer overrun vulnerability in the fingerd daemon.

There are three main actions to resolve the problem. First is to utilize the /GS compile option. This option creates a cookie between the stack overrun and the return address. This allows the system which helps to prevent buffer overruns, by changing the stack layout [11]. The second action is to use the ¡strsafe.h¿ library. This library has buffer overrun safe functions that will help with the detection of buffer overflows. Finally, the last action is to perform extensive code reviews of string functionality and indexes utilized within your application.

Format String Attack A format string attack occurs when a program reads input from the user, or other software, and processes the input as a string of one or more commands. If the command that is received is different from what is expected from it, like being longer or shorter than the allocated data space, the program may crash, quit or make up for the missing information by reading extra data from the stack and allowing the execution of malicious code [20].Format String Attack allows to dump the stack. Stack contains interesting information: data, code pointers, stack addresses of the format string, format string's stack offset location. These values are enough to let the attacker write the exploit. Using these values the stack will be dumped until format string is found. Locating the pointer address of format string is the main task. Then Choose the overwritten address on the stack, point format string at overwritten address and write address of shell code to the end of string. Adjust offsets for the Address of format string based on its length. Format string needs its own address for reference where the exploit is then injected which may lead to system crash.

For example: When a string value is passed into a program it will be read as one of several types of data. It could be a String, Integer, Word, etc. In any well written program, a programmer will specify the type of and length of data handled by the string function (e.g. printf, wsprintf, etc.). Since a string can contain almost any character,

so an attacker has to provide it with a format token and the entered string will explode into a much longer value. For example, if we assume a = 'C', we could print it using printf('C'), It will work, but an attacker could enter the value of 'Cmod6' which tells the computer to print the value as a double word, thus filling up much more space on the stack than a simple string value.

Another type of formatting error can occur when a programmer forgets to allocate space for values that change from one format to another. For example, when a string enters a system from a network it is in one long row of characters (e.g. bbbb). Programmers have to check the length of this string and allocate space for it on the stack [16]. However, it is required to change the string type to Unicode, which uses twice as much space than the string, and but attempts to use the same length to allot more space on the stack. The problem is, the programmer must keep in mind to allocate twice as much space as required earlier. As a Windows Mobile bug hunter, this error is seen frequently. Most programmers seem to forget that this operating system requires Unicode values to function. Hence it leads to a crash.

2.5 Countermeasures of buffer overflow

Defending against buffer overflow is very important because it causes serious issues with the security of a network. Different approaches and procedures lead to control buffer overflow. These approaches may use manual techniques by the programmers, some software approach or some hardware approach which ever suits the best.

2.5.1 Software level defense

Writing Correct Code: "To err is human, but to really foul up requires a computer." – Anon. Despite a long history of understanding of how to write secure programs vulnerable programs continue to emerge on a regular basis. Thus some tools and techniques have evolved to help novice developers write programs that are less likely to contain buffer overflow vulnerabilities. To combat the problem of subtle residual bugs, more ad-

vanced debugging tools have been developed, such as fault injection tools [4]. The idea is to inject deliberate buffer overflow faults at random to search for vulnerable program components. There are also static analysis tools emerging that can detect many buffer overflow vulnerabilities.

Non-Executable Buffers: The general concept is to make the data segment of the victim program's address space non-executable, making it impossible for attackers to execute the code they inject into the victim program's input buffers. More recent UNIX and MS Windows systems have come to depend on the ability to make dynamic code into program data segments to support various performance optimizations. Thus one cannot make all program data segments non-executable without sacrificing substantial program compatibility.

Array Bounds Checking: While injecting code is optional for a buffer overflow attack, the corruption of control flow is essential. Thus unlike non-executable buffers, array bounds checking completely stops buffer overflow vulnerabilities and attacks. To implement array bounds checking, all reads and writes to arrays need to be checked to ensure that they are within range. Many tools like Compaq c compiler, array bound checking through GS patch, purify-memory access checking etc. are used for array bound checking.

Type safe languages: All buffer overflow vulnerabilities result from the lack of type safety in C. If new, security-sensitive code is to be written, it is recommended that the code be written in a type-safe language such as Java or ML. There are millions of lines of code invested in existing operating systems and security-sensitive applications, and the vast majority of that code is written in C. So its prevention is utmost important.

Runtime instrumentation: Compile-time techniques like Stack Guard and Return Address Defender (RAD) insert code to check for return address modification [9]. Stack Guard places a canary before the return address and checks the canary's value when the function returns. RAD creates a Return Address Repository global array and copies the return address to it in the function prologue. It then checks for modifications in the function epilogue. These approaches are not completely foolproof because attackers can

alter the return address indirectly by using a pointer.

Static and dynamic code analysis: Researchers have proposed algorithms for selecting susceptible buffers, creating buffer overruns, and based on the result, analyzing the application for susceptibility [9]. This technique identifies locations that call unsafe library functions on local buffers and non library functions that read or copy user input. It then calculates the return address that attackers would overwrite to insert an attack string.

2.5.2 Hardware based defense

Hardware assisted approaches improve on software schemes and result in low performance overheads, but require architectural modifications, often to the CPU and instruction set, thereby requiring a significant buy-in from chip manufacturers.

Protecting the return address by hardware: This approach requires extra backing store for every process to save the duplicate stack to and introduces some complexity to the processor in handling stack overflows. An interesting challenge that must be solved for these approaches is as follows: This approach must keep the protected return address stack in sync when dealing with constructs that arbitrarily pop off several stack frames like setjmp/longjmp [32]. When the code returns from a call, the processor checks to make sure that the return value matches the value stored on the duplicate stack. They explore both hardware and operating system initiated save-on-overflow.

A Compiler-Hardware Technique for Protecting Against Buffer Overflow Attacks: New secure hardware module called a Guard, The Guard, sits between the cache and main memory, can be implemented as reconfigurable logic such as a field programmable gate array (FPGA) that augments the CPU functionality, or simply as a gateway chip that interfaces the CPU with the system bus. It models the simulations with the FPGA method. Note that recent work has shown the effectiveness of using FPGA architectures to address specific security threats. The return address is copied to a memory region in the hardware Guard that is inaccessible through any direct memory calls. This mem-

ory region is called the return address (RA) stack, and it is automatically managed by a modified compiler [13]. The compiler inserted instructions to manage the RA stack synchronize the state of the RA stack with the program stack, independent of code locality and caching policies. On each function call (CALL), the return address is stored in the RA stack, and at each function return (RET), the Guard checks if an overflow has occurred and ensures that the correct address is returned to the processor. Thus, the Guard maintains a copy of the valid return address and ensures that the correct address is always returned regardless of any buffer overflows.

2.6 Windows Usage and its Popularity

Windows operating system has been the most popular and widely used platform in the computer world. Whether its any financial organization, medical organizations, educational institutes usage of windows has always been on the top of charts. Windows has made a household name through its Personal Computers. Yet it only really took off in the mid 90's and with the release of Microsoft's Windows 95 operating system there was no going back. After windows 95, windows 2000, windows XP, windows vista, windows server, windows 7 windows 8, all the versions of windows have been immensely used in every sphere of life [24].A survey conducted by Net Market Share.com of 1.5 billion desktop and portable PCs Windows sweeps the show Figure 2.5 depicts the monopoly of Windows platform.

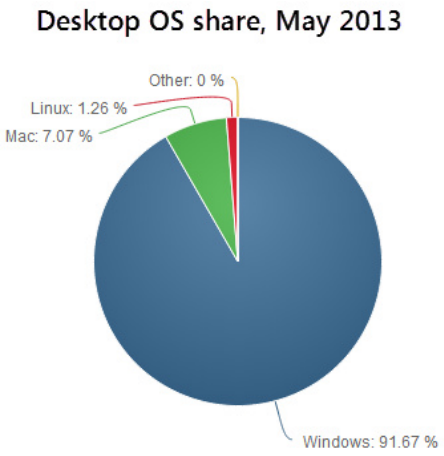

Figure 2.5: Statistics of usage of Operating Systems of May,2013 [49]

The main reason behind this popularity is its ease to use. Ease to use refers that how easy it is for a lay man to understand and use the computer system without much of technical knowledge. With loads of innovative functionality for better user experience provided by the operating system and new means of communication a new life style is born where people are experiencing a smooth life while using computers. But if the security triangle is analyzed then it could be easily seen that windows has emphasized on ease of use the most and security is neglected as shown in Figure 2.6. Security of triangle depicts the three major aspects of any application created, these are Functionality, Ease of use and Security as referred by Eric Cole in his book" Network Security Bible".

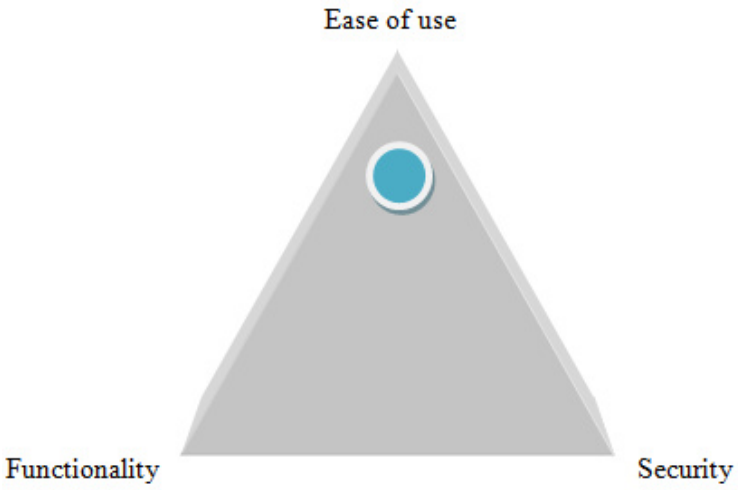

Figure 2.6: Security Triangle

Ideally the ball is supposed to be in the center, but in windows the ball is tilted towards ease of use. Because of which Security and functionality suffers. Security is must if the protection of personal information from unwanted prying eyes and malicious programs is required.

Computer worms are self-replicating computer programs. These can take over the computers without any knowledge, consuming our network connectivity and create havoc on our machines. The rates of infection of worms in computers observed in the past are just astonishing. As discussed in the last section there have been many attacks on the Windows platform also. These attacks include the hardware, software applications, databases, protocols, etc. Hence it could be concluded that security of any windows system is always under question. Security of any Windows applications is hindered if

any type of vulnerability exists. This vulnerability can be any hardware or any software vulnerability. Most of the insecure applications exist because of the software vulnerability. The most common operating system "Windows" contains the most common software vulnerability that is "Buffer overflow".

2.7 Buffer Overflow Attack on Windows

As it is seen above that Windows platform is very user friendly and focuses more on functionalities rather than on the security aspect of the system. Thus many vulnerabilities exist which are every now and then exploited by the hackers and intruders. There exist Hardware, Software and Application vulnerabilities which hinder the security of many Windows platforms like Windows 98, Windows 2000, Windows XP etc. Some of these vulnerabilities are discussed below:

2.7.1 Microsoft Windows Print Spooler Buffer Overflow Vulnerability

Microsoft windows print spooler service ('spoolsv.exe') manages printing process including the identification of the correct printer driver loading the printer driver, spooling high-level function calls into a print job and scheduling the print job for printing.The Windows Print Spooler manages the printing process, and loads files to memory for later printing. Microsoft Windows operating systems use the Server Message Block (SMB) protocol to support services such as file and printer sharing [29].The main responsibilities of print spooler include:

1. Finding whether a print job should be handled locally or remotely.
2. Accepting a data stream created by GDI, in conjunction with a printer driver, for output on a particular type of printer.
3. Spooling the data to a file.

4. Selecting the first available physical printer in a logical printer queue.

5. Converting a data stream from a spooled format (such as enhanced meta file (EMF)) to a format that can be sent to printer hardware (such as printer control language (PCL)).

6. Sending a data stream to printer hardware.

7. Maintaining a registry-based database for spooler components and printer forms.

8. Rendering print jobs on the client computer instead of on the print server. Client-side rendering eases the print server workload, is transparent to the print driver, and is enabled by default in Windows Vista.

2.7.2 The architecture of Print Spooler:

The primary components of the Microsoft Windows 2000 and later print spooler are illustrated in the Figure 2.7

Application:The print application creates a print job by calling GDI functions.

GDI:The Graphics Device Interface (GDI) includes both user-mode and kernel-mode components. The user-mode component, Microsoft Win32 GDI, is used by Win32 applications that require graphics support. The kernel-mode component, the graphics engine (or graphics rendering engine), exports services and functions that graphics device drivers can use [28].

Winspool.drv Winspool.drv is the client interface into the spooler. It exports the functions that make up the spooler's Win32 API, and provides RPC stubs for accessing the server. (GDI is the primary client, but applications also call some of its Win32 functions.)

Spoolsv.exe Spoolsv.exe is the spooler's API server. It is implemented as a Windows 2000 (or later) service that is started when the operating system is started. This module exports an RPC interface to the server side of the spooler's Win32 API. Clients of

Spoolsv.exe include Winspool.drv (locally) and Win32spl.dll (remotely). The module implements some API functions, but most function calls are passed to a print provider by means of the router (Spoolss.dll).

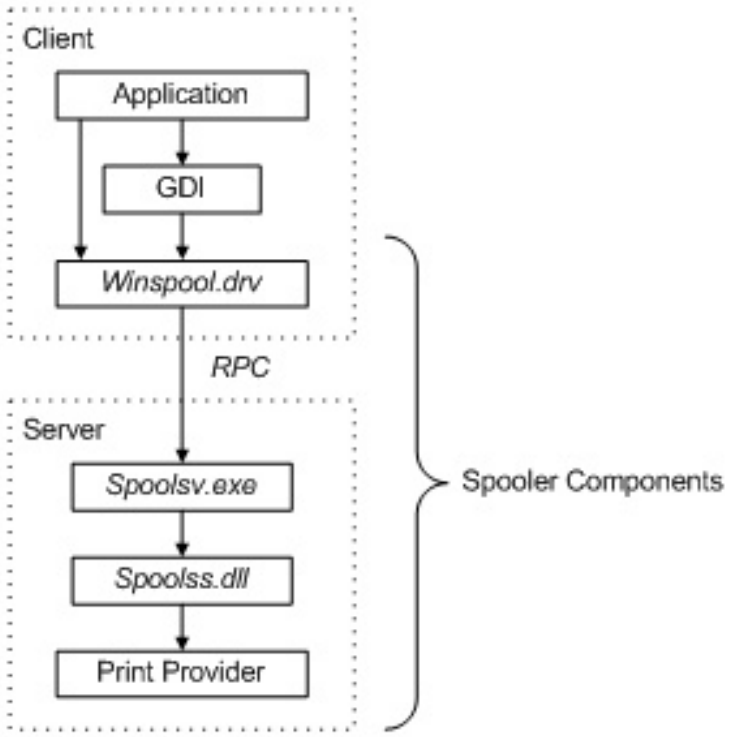

Figure 2.7: Architecture of Print Spooler [28]

Router The router, Spoolss.dll, determines which print provider to call, based on a printer name or handle supplied with each function call, and passes the function call to the correct provider.

Print Provider The print provider that supports the specified print device.

Print Monitor If printer hardware is local to the system on which the application is running, the "client" and "server" are the same system.

But Microsoft Windows Print Spooler service is prone to a buffer overflow vulnerability. The Problem occurs when the intruder send a large size of message to print that the print spooler cannot handle thus it causes the printer buffer overflow. It is prone to this vulnerability as the software fails to perform boundary checks before copying user-supplied data into sensitive process buffers.A buffer overflow vulnerability has been reporting in the handling of some malformed SMB requests.It occurs when service

handles malformed messages containing excessive data. These messages are passed to a finite sized buffer, triggering an overflow condition and facilitating memory corruption. A successful attack may result in arbitrary code execution which can allow an attacker to gain system privileges.

This vulnerability exists in the Enumerate Print Shares function in win32spl.dll. The vulnerable function does not correctly validate the length of the printer server's response [44]. When a malformed response is received from the printer server, the stack buffer can be overflowed, resulting in an exploitable condition.

This vulnerability facilitates local privilege escalation and unauthorized remote access depending on the underlying operating system. A remote unauthenticated attacker can exploit this issue on Windows 2000 and Windows XP SP1.

Windows XP SP2 and Windows Server 2003 require a remote attacker to authenticate to a computer or have local interactive access for successful exploitation. Remote exploitation on these operating systems is possible only if a user has shared a printer or has attempted to connect to a shared printer. Furthermore, this issue can trigger only a denial-of-service condition in Windows XP SP2 and Windows Server 2003 [47].This vulnerability is referred as CVE-2009-0228 since 01/20/2009. This Vulnerability has been referred for patches under the Vulnerability list as Microsoft Security Bulletin MS05-043.

2.7.3 Microsoft Office Web Components ActiveX control buffer overflow:

The Microsoft Office Web Components ActiveX control is vulnerable to a heap-based buffer overflow. By persuading a victim to visit a specially-crafted Web site that passes invalid parameters for the ActiveX method, a remote attacker could overflow a buffer and execute arbitrary code on the system or cause the application to crash [18]. This security update is rated Critical for all supported editions of Microsoft Office XP, Microsoft Office 2003, Microsoft Office 2000 Web Components,Microsoft Office XP Web Components, Microsoft Office 2003 Web Components, etc. Microsoft Office Web Components are

ActiveX controls that provide Microsoft Office functionality, such as spreadsheets, tables, and charts. These ActiveX controls are provided by the file MSOWC.DLL. Several of the ActiveX controls provided by MSOWC.DLL are marked Safe for Scripting and Safe for Initialization, which means that they can be controlled by a web page in Internet Explorer.An attacker could exploit this issue by tempting a victim to visit a maliciously crafted web page. Successful exploits will allow the attacker to execute arbitrary code within the context of the affected application that uses the ActiveX control (typically Internet Explorer). Failed exploit attempts will result in a denial-of-service condition.

By convincing a user to view a specially crafted HTML document (e.g., a web page or an HTML email message or attachment), an attacker may be able to execute arbitrary code with the privileges of the user.An attacker who successfully exploited these vulnerabilities could gain the same user rights as the local user. Users whose accounts are configured to have fewer user rights on the system could be less impacted than users who operate with administrative user rights.This vulnerability is exploited remotely from any system.

Its is referred in the list of common vulnerabilities and exposures as CVE-2009-1534 and its defect number in Microsoft is MS09-043.The credit for finding its existence goes to Sean Larsson of VeriSign iDefense Labs. Almost every system and application of Windows is affected by this vulnerability hence many patches are available which are recommended to upgrade while using these Windows web components.Its referred as Microsoft Security Bulletin MS09-043 in the security patches.

2.8 Techniques to avoid buffer overflow

With the first Buffer Overflow discovered by aleph one in 1988, since then there have been many techniques and tools developed for the same [35]. With each growing attack on different tools applications and protocols etc Microsoft has been working to control this vulnerability. Till now it is clear that buffer overflow is not hard to achieve but the exploitation it causes is hard to control. Because of this reason controlling buffer

overflow or stack overflow is quite important. Many techniques have emerged to safe guard the buffer or stack while running any application. Each technique has its own characteristics according to their usage are used and implemention is done. The buffer overflow protection is applicable at 2 different categories of applications : application level and operating system level.Figure 2.8 shows various techniques developed till date.

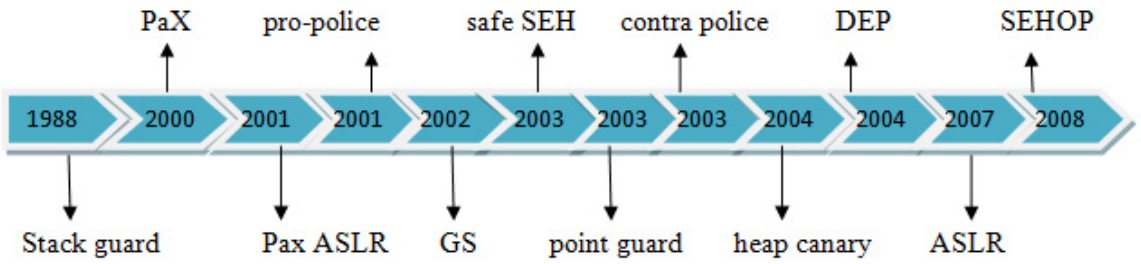

Figure 2.8: Techniques developed to safeguard Buffer Overflow

The above diagram shows the various techniques emerged over the years to protect buffers and prevent its overflow. These are various techniques being developed for different platforms like windows and linux etc. Each technique had some issues because of which new technique was developed. The next section elaborates the significance and use of the techniques being applied in windows platform.

2.8.1 Stack guard:

Stack Guard is a compiler that emits programs hardened against stack smashing attacks.Programs that have been compiled with Stack Guard are largely immune to stack smashing attack. Protection requires no source code changes at all a simple compiler technique that virtually eliminates buffer overflow vulnerabilities with only modest performance penalties. Stack Guard is a simple compiler extension that limits the amount of damage that a buffer overflow attack can inflict on a program. Stack Guard thwarts this class of attack by effectively preventing changes to the return address while the function is still active. If the return address cannot be changed, then the attacker has no way of invoking the injected attack code, and the attack method is thwarted. Stack Guard prevents changes to active return addresses by either detecting the change of the return address before the function returns, or by completely preventing the write to the

return address. Detecting changes to the return address is a more efficient and portable technique, while preventing the change is more secure.

Stack Guard supports both techniques, as well as adaptively switching from one mode to the other [5]. Stack Guard does this by placing a "canary" word next to the return address on the stack. When the function returns, it first checks to see that the canary word is intact before jumping to the address pointed to by the return address word. The buffer overflow attack method exploits the fact that the return address word is located very close to a byte array with weak bounds checking, so the only tool the attacker has is a linear, sequential write of bytes to memory, usually in ascending order. Under these restricted circumstances, it is very difficult to over-write the return address word without disturbing the canary word. Hence if the canary gets overwritten the user knows that buffer overflow occurred. But the main problem with this technique is, it prevents only buffer overflow attacks that target the saved EIP, so there are various ways to bypass these defenses.

Microsofts protection has a big difference, it does protect the frame pointer, placing a random canary between frame pointer and local variables [37]. When using this protection, if a buffer overflow is used to change the frame pointer or the return address, the random canary would be inevitably altered. But stack guard can be controlled by the intruder in some cases and thus gaining full control of the system even if it is protected by safe guard. The various attack can be through Functions arguments control,Returning with an altered frame pointer, showing More control over local variables,Pointing callers frame to get the local variables. The stack guard can be protected in the same way as the frame pointer is protected. The main solution to protect stack guard is the introduction of random canary. Using a random canary may give some higher level of security, but its not a full proof solution too. So this led to device other methods to protect stack from overflow.

2.8.2 GS option:

The Buffer Security Check option, known by its flag name GS is used to mitigate buffer overflow vulnerabilities in C and C++ code that allow an attacker to overwrite important stack data and seize control of the program. The primary goal of GS protection is to detect corruption of a functions return address that is stored on the stack and abort execution if corruption is detected. The GS feature also provides some other protections by careful layout of stack data. GS prevents the attacker from using an overwritten return address on the stack. Its main task is to insert a stack cookie between the local variables and return Address. It Checks the cookie at the function epilogue. Thus prevents the attacker from overwriting other local variables or arguments. If by any chance the value of cookie is different from the value earlier specified it indicates the system that some sort of corruption has occurred and thus stops the further execution of the program. String buffers go above other variables and arguments are copied below local variables thus it tells that some fault occurred.

When ever an application starts, the very first function that comes under execution is the C RunTime (CRT) entry points such as main CRT Startup. The first action taken by these functions is to call security init cookie, which is responsible for initializing the cookie that will eventually end up in every qualified function's stack frame.The Visual Studio C++ compiler mainly supports this Buffer Security Check option, known by its flag name, GS. This option causes the compiler to add checks that protect the integrity of the return address and other important stack meta data associated with procedure invocation [45]. The GS protections do not eliminate vulnerabilities, but rather make it more difficult for an attacker to exploit vulnerabilities..

A cookie value created is copied from a program-wide master cookie and placed on the stack in between the functions return address and any space allocated for local variables.The master cookie value that is copied onto the stack in the prologue and compared against in the epilogue is a global value initialized by the C runtime (CRT). While the program is starting up, the security init cookie function is called to initialize the master cookie value and store its value in the security cookie variable. The primary goal

of security init cookie is to generate a nondeterministic value for the security cookie. To accomplish this, a number of environmental values are captured, including these values like System Time,Current Process ID,Current Thread ID,Static value in the PE,Current Tick Count, Performance Counters .There is a cost involved in implementing the GS feature. Additional code is added to every protected function, and additional stack storage is used to store cookie values. For this and other reasons, GS protection is not necessarily applied to all functions even if the GS option has been selected. Still it is believed that GS option is not the final solution to safeguard the stack as it also has some limitations.

2.8.3 Structured Exception Handling:

SEH is an acronym for Structured exception handling. Exception handling is built into the Windows operating system and helps make applications more robust. When a problem occurs such as a memory access violation or divide by zero error an exception handler can be coded to deal with such situations and allow the process to recover and continue execution as normal [24]. Even if the program developer has not set up any exception handling every thread of every process has at least one handler that is setup on thread initialization. Information about exception handlers is stored on the stack in an EXCEPTION REGISTRATION structure and a pointer to the first EXCEPTION REGISTRATION structure is stored in the Thread Environment Block.

The SEH is a mechanism in Windows that makes use of a data structure called "Linked List" which contains a sequence of data records. When an exception is triggered the operating system will travel down this list. The exception handler can either evaluate it whether it is suitable to handle the exception or it can tell the operating system to continue down the list and evaluate the other exception functions.Since the Windows stack grows downward it will be seen that the order of the records is reversed in the linked list[nSEH]...[SEH]. When an exception occurs in a program's function ,the exception handler will push the elements of it's structure to the stack since this is part of the function prologue to execute the exception. At the time of the exception the SEH will

be located at esp+8.

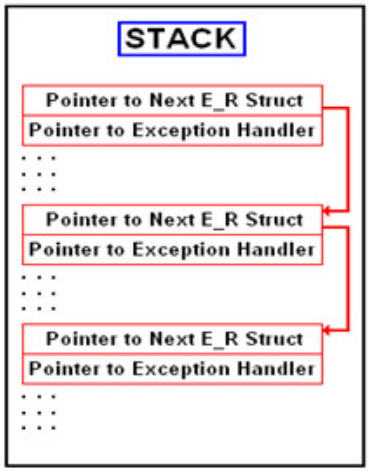

Figure 2.9: SEH Handler [26]

As it can be seen in Figure 2.9 the EXCEPTION REGISTRATION structure has two elements, a pointer to the next EXCEPTION REGISTRATION structure and a pointer to the exception handler.Structured exception handling is a mechanism for handling both hardware and software exceptions. Therefore, the software code will be able to handle hardware and software exceptions identically. Structured exception handling enables you to have complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and machines.

Working of SEH: The various blocks of code have to be properly encapsulated, with each block having one or more associated handlers. Each handler must specify some form of filter or exception condition to handle [26]. When an exception is raised by code in a protected block, the set of corresponding handlers are to be searched in order, and the first one with a matching filter condition will be executed. A single method can have multiple structured exception handling blocks, and the blocks can also be nested within each other. If in any case no exception handler could manage the exception condition, the handler will make the program to stop executing.

There is also a Thread Information Block (TIB) is a data structure in Win32 that stores information about the currently running thread. At the position FS:[0x00] the current exception handler is found and it is the position from where the exception handler starts at assembly level. Various Exception handlers form a linked list chain on the stack,

and sits relatively close to the bottom of the stack.When an exception occurs, Windows retrieves the head of the SEH chain walks through the list and tries to find the suitable handler to close the application properly.

But there are many chances that an attacker would try to by pass this protection method to gain control of the system through buffer. Unfortunately it is possible as SEH is also flawed mechanism.When overflowing stack based buffers on Windows platforms one of the mechanism available to the attacker to gain control of the process is to overwrite this EXCEPTION REGISTRATION structure,setting the pointer to the handler to a block of code or an instruction that will redirect the flow of execution back to the buffer. In case if the handler is overwritten by any chance then the attacker can get control of the buffer easily, leading to exploitation. The attacker will follow a simple method ,that is to overwrite SEH with a pointer to a POP POP RETN instruction (the POP instruction will remove 4-bytes from the top of the stack and the RETN instruction will return execution to the top of the stack). It is to be considered that the SEH is located at esp+8 so if the attacker increments the stack with 8-bytes and returns to the new pointer at the top of the stack then attacker will be executing nSEH. Then using nSEH bytes to write some opcode to jump to an area of memory that could be controlled easily the attacker can place the desired shell code [17]. So it is recommended to use other safe guard methods superior than SEH.

2.8.4 Safe SEH:

Safe seh was introduced in early 2003 to overcome the gap of SEH. Among other improvements in Windows XP SP2 and Windows Server 2003 Microsoft introduced the concept of "safe structured exception handling." The general idea is to collect handler's entry points in a designated read-only table and have each entry point verified against this table for exceptions prior to control being passed to the handler. In order an executable is to be created with a safe exception handler table, each object file of the linker command line must contain a special symbol '@feat.00'. If by any chance the object file passed to the linker does not contain this symbol, then the exception handler table is deleted from

the executable and thus the run-time checks will not be performed for the application. When /SAFESEH is specified, the linker will only produce an image but if it can also produce a table of the image's safe exception handlers that will be more useful [41]. This table specifies the operating system which exception handlers are valid for the image.The safeseh directive instructs the assembler to produce appropriately formatted input data for the safe exception handler table.

Safe seh stores the valid exception handlers which the system can execute when any exception occurs.If an application has a safe exception handler table,and is attempting to execute any unregistered exception handler, it will result in immediate program termination. Thus the attacker has lower chances to by pass this technique as the attacker can not create and register its known handler's image. It is important to register each exception handlers entry point with the safeseh directive. Saving the SEH overwriting involves [36] making changes to the compiled versions of code such that executable files are made to contain meta data that the platform would need to properly mitigate this technique.

Microsoft pursued this approach and released a functional mitigation with Visual Studio 2003. Unfortunately, the need to rebuild executables in combination with the inability to completely handle cases where an exception handler is pointed outside of an image file make the SafeSEH approach less attractive. There are various ways by which the Safe SEH could be by passed. These can be:

If the vulnerable application is not compiled with safeseh and one or more of the loaded modules (OS modules or application-specific modules) is/are not compiled with safeseh, then the attacker can use a pop pop ret address from one of the non-safeseh compiled modules to make it work. In fact, its recommended to look for an application specific module (that is not safeseh compiled), because it would make the exploit more reliable across various versions of the OS.

If the only module without safeseh protection is the application/binary itself, then you may still be able to pull off the exploit, under certain conditions. The application binary will (most likely) be loaded at an address that starts with a null byte. If the

attacker can find a pop pop ret instruction in this application binary, then that address could be used (the null byte will be at the end), however the attacker will not be able to put the shell code after the SE handler overwrites because the shell code would not be put in memory and the null byte would act as string terminator. So in this scenario, the exploit will only work if the shell code is put in the buffer before nseh/seh are overwritten.

But still Safe SEH also has some limitations and gaps because of this the latest technique under usage is SEHOP.

2.8.5 SEHOP:

The purpose of the SEHOP(structured exception handler override protection) mitigation is to prevent an attacker from being able to make use of the Structured Exception Handler (SEH) overwrite exploitation technique. SEH overwrites are also commonly used by exploits that target the increasing number of browser-based vulnerabilities. At a high-level, the SEH overwrite technique uses a software vulnerability to execute arbitrary code by abusing the 32-bit exception dispatching facilities provided by Windows. At a functional level, an SEH overwrite is generally accomplished by using a stack-based buffer overflow to overwrite an exception registration record that has been stored on a thread's stack. An exception registration record is composed of two fields: a next pointer and an exception handler function pointer. The next pointer is used to link an exception registration record to the next record in the singly-linked list of registered exception handlers [6]. The exception handler function pointer is called by the Windows exception dispatcher when an exception occurs. The definition for an exception registration record can be seen below:

Typedef struct EXCEPTIONREGISTRATIONRECORD struct EXCEPTIONREGISTRATIONRECORD Next;

PEXCEPTIONROUTINE Handler;

EXCEPTIONREGISTRATIONRECORD, PEXCEPTIONREGISTRATIONRECORD;

SEHOP involves adding dynamic checks to the exception dispatcher that do not rely

on having metadata derived from a binary. At a high-level, SEHOP prevents attackers from being able to use the SEH overwrite technique by verifying that a thread's exception handler list is intact before allowing any of the registered exception handlers to be called. This mitigation technique is made possible because of an implicit side effect of an SEH overwrite. When the majority of stack-based buffer overflows occur, an attacker will implicitly overwrite the next pointer of an exception registration record prior to overwriting the record's exception handler function pointer. Since the next pointer is corrupted, the integrity of the exception handler chain is broken. SEHOP achieves this functionality in two distinct steps.

1. The first step involves the insertion of a symbolic exception registration record as the tail record in a thread's exception handler list. This step occurs when a thread first begins executing in user mode. Since exception registration records are always inserted at the head of the exception handler list, the symbolic record is guaranteed to be the final exception registration record.

2. The second step consists of walking the exception handler list at the time that an exception is being dispatched to ensure that the symbolic record can be reached and that it is valid as in Figure 2.10. This step happens when the exception dispatcher is notified that an exception has occurred in user mode. If the symbolic record cannot be reached, the exception dispatcher can assume that the exception handler list is corrupt and that an SEH overwrite may have occurred. The exception dispatcher is then able to safely terminate the process as depicted in Figure 2.11.

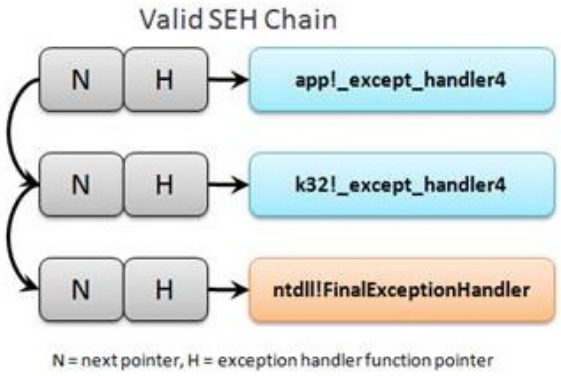

Figure 2.10: SEHOP Handler [6]

This shows that whenever any exception occurs the SEH handler starts traversing the list to find out the appropriate handel to the exception occured, but if the handle is not found then the chain is terminated.

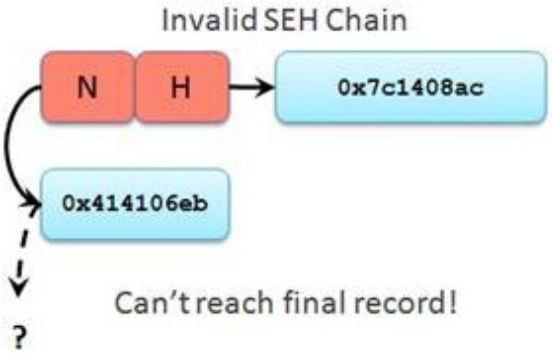

Figure 2.11: SEHOP invalid handler [6]

SEHOP is enabled by default on Windows Server 2008 and disabled by default on Windows Vista SP1. All the above techniques are accountable to provide security to various applications and programs used in windows platform.

Chapter 3

Problem Formulation

As it is seen from the above study that Windows Platform is badly affected by Buffer Overflow Vulnerability and could be exploited by a little know how of the memory structure and the registers.The main reason for Windows buffer overflow is the inclination towards the ease of use because of which many attackers try to gain the root privileges.Moreover because of the popularity of Windows platform amongst the masses make the attackers to hijack the applications and get control of victim's system.The main point of concern is many of the programmers focus on creating large functionalities on one software rather than giving emphasis on the security of the software.

Starting from the Windows 98, Windows 2000, Windows XP, etc, no platform is completely secure from the buffer overflow vulnerability. Many techniques like Stack guard, GS option, SEH etc. have been developed to protect windows applications but still these are not completely able to do that. Many higher version of Windows Operating systems like Windows Vista, Windows 7, Windows 8 are provided with new techniques to safeguard the systems which are successful at protecting these systems. The techniques like Safe SEH, SEHOP are some of the techniques useful in safeguarding the windows platform.

Besides of the presence of above techniques still there exist some methods through which these techniques could be bypassed and the attacker can gain the root level privileges.This can happen by manipulating the registers at a deeper level than done earlier

and with some well planned attack the latest technique like SEHOP can also be circumvented and advantage could be gained.

This Thesis works Showcases:

1. The methods by which the above techniques could be circumvented and how by different means this could be stopped and hence the Windows platform could be saved from Buffer Overflow vulnerability.

2. The usage of dll to exhibit the various security techniques implemented on the application using its executable file which helps in finding out whether the application under use is vulnerable or not.

Chapter 4

Objectives

1. To Study and analyze Buffer Overflow techniques specific to Windows Operating System.

2. To Implement Structured exception handling based Buffer Overflow methods on various Operating environments.

3. To Verify and Validate the results by implementing dll and integrating same with debugger at run time.

Chapter 5

Implementation and Results

Besides following various norms of development still many Softwares are developed with vulnerabilities in them. The main reason for this could be that the programmers give more attention to planning and development rather than on testing the security of the software. Due to this reason a lot many small bugs are not raised as a problem but later these bugs become the reason of big exploits.

These days many companies are adopting new approach to software development which states 'Building Secure Softwares'. These Companies and Organizations are hiring new experts for creating a vulnerability free software. Earlier the Software development team involved project managers, programmers, developers,testers etc but with the demand for secure software has added new team member which is known as a Security Analyst whose main job is to critically find out that by any case Is the Software under development is secure or vulnerable?, only after the right signal the software is released in market.

But still there are many softwares which are prone to one or the other vulnerability. These may be because of the legacy code or because of the carelessness of the development team. And since Windows platform is the most widely used Operating System because of its ease of use, its really shows a bad picture. The most common Vulnerability which exists in its softwares is Buffer Overflow. It was believed that Buffer Overflow does not exist any more but latest trends of vulnerabilities break those myths. It still prevails in

majority. The Figure 5.1 supports the existence of buffer overflow and its latest exploit.

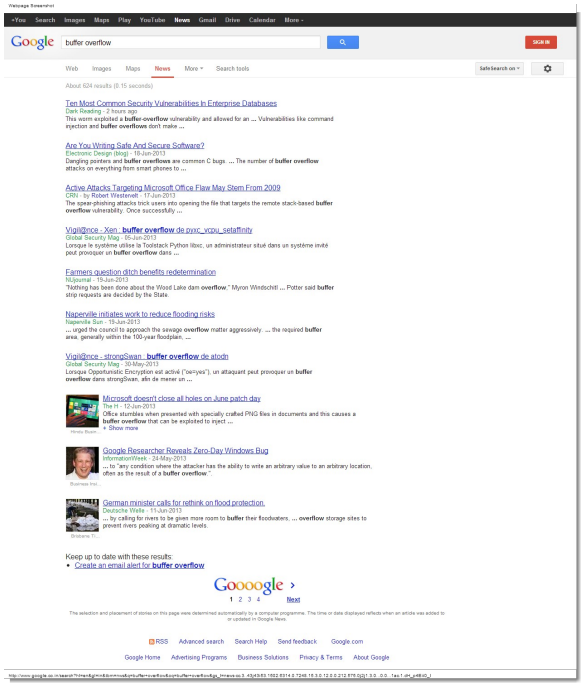

Figure 5.1: Google News showing existence of Buffer Overflow

This implementation displays the various test beds created to analyze that how with and without the security techniques the any exe file behaves. This enables the user to find out from the exe file only that what types of security features are implemented on the compiled file.

The C source code on which these test bed is implemented is written below. The main task of this code to just copy the values from the arguments provided and print them with hi and bye. The main point to notice is here is whenever the use wants to find that whether the particular application is vulnerable or not if the source code is provided then the best way is to analyze the input. If the input is to be provided from command line then there are chances of the application to be vulnerable. It is not compulsory but chances of the vulnerability may exist. The C source code is shown in Figure 5.2:

To analyze this code during its execution, a debugger called as ollydbg is used as discussed in Chapter 2.

```
#include<stdio.h>
display(char *temp1, char *temp2)
{
char name[400];
strcpy(name, temp2);
printf("Hi %s %s\n", temp1, name);
}
int main(int argc, char *argv[])
{
display(argv[1], argv[2]);
printf("Bye %s %s\n",argv[1], argv[2]);
}
```

Figure 5.2: C Source code used for test beds

5.1 Compilation Options

5.1.1 Zi Switch

Any programming file whenever compiled with the cl compiler on the command prompt creates a exe file. This exe file has particular size. But if the source code is compiled with an extra switch with cl that is 'Zi', the size of exe file is much larger than the previous file as shown in Figure 5.3 and Figure 5.4

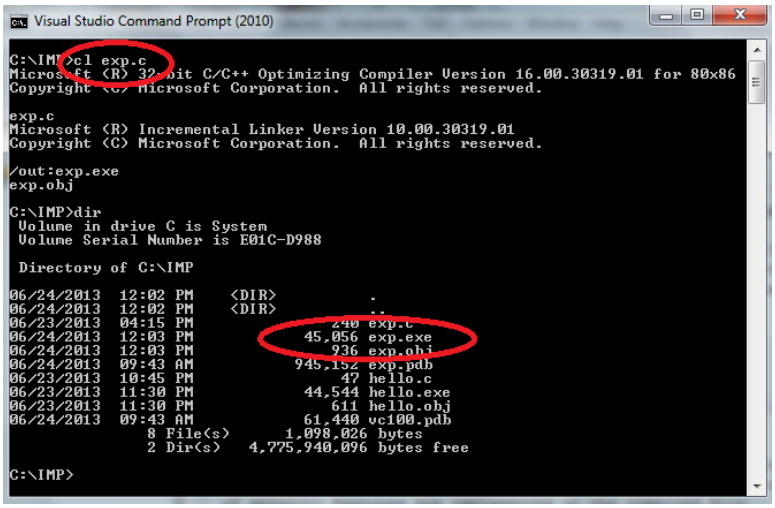

Figure 5.3: exe file size when complied with cl only

Now the purpose of using Zi switch is to generate the symbol table and various other information about the flow of the program being executed. This flow when studied by the testers and debuggers enables to catch the vulnerabilities which my affect the security of the application. It produces a program database (PDB) that contains symbolic debugging information for use with the debugger. The symbolic debugging information includes the names of the variables, functions and their working. The following Figure 5.5 shows the

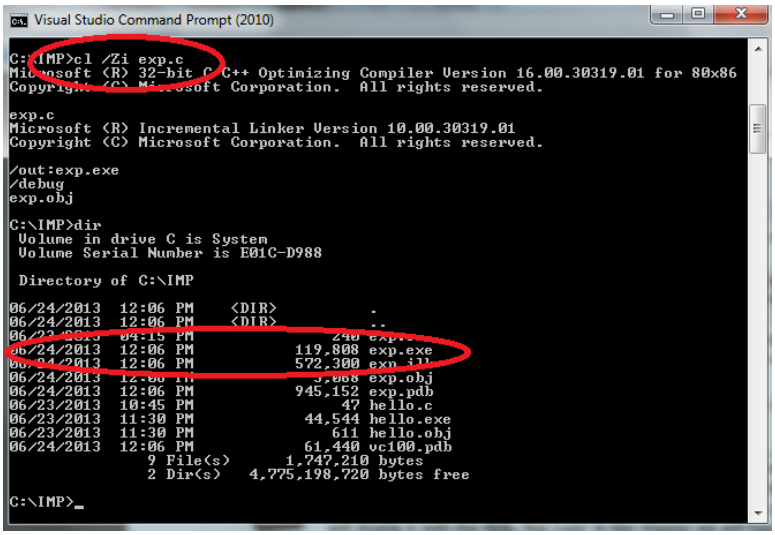

Figure 5.4: exe file size when complied with cl with Zi

various functions of the C source code like strcpy,src(source), dest(destination),formats, printf etc.

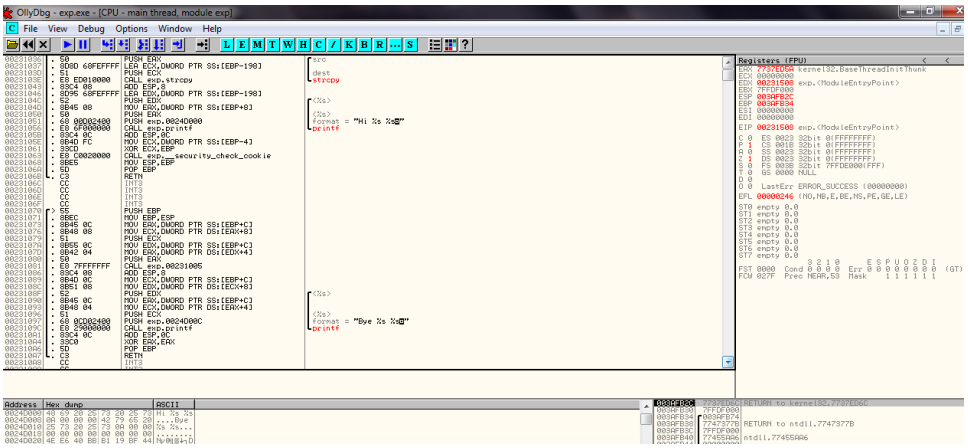

Figure 5.5: Functions and Symbols generated with Zi switch

When the source code is executed completely with all the arguments used in the function the process gets complete and terminates successfully showing the correct address at the EIP Figure 5.6.

5.1.2 GS switch

Guard stack or called as a GS switch as discussed in chapter 2. It is basically used to protect the source code from the Buffer Overflow attack as it creates a cookie at the start of the execution which safeguards the application. If by any means any attacker

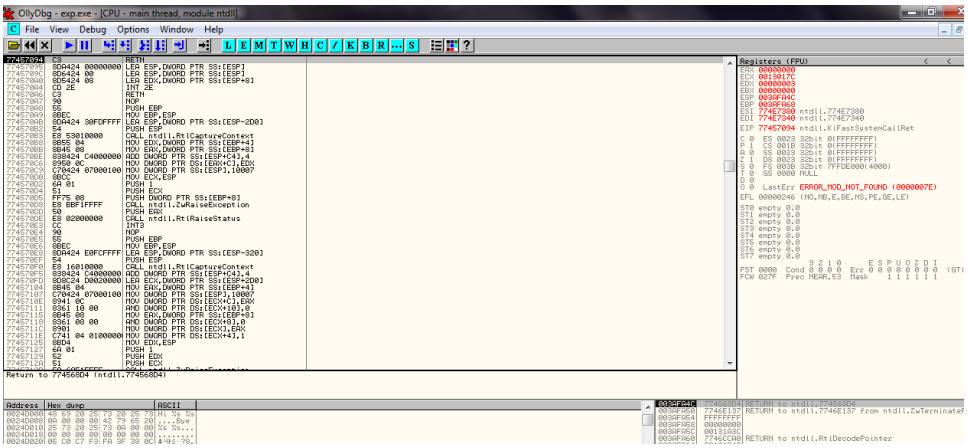

Figure 5.6: Successfully Termination and Execution of the code

tries to exploit the application, the generated cookie stops the exploit and terminates the program then and there.

By default GS switch is provided on Windows 7 Operating system. Thus Whenever a program is complied as in Figure 5.3 it displays the following screen as output which has the first command as the security init cookie being generated.Figure 5.7

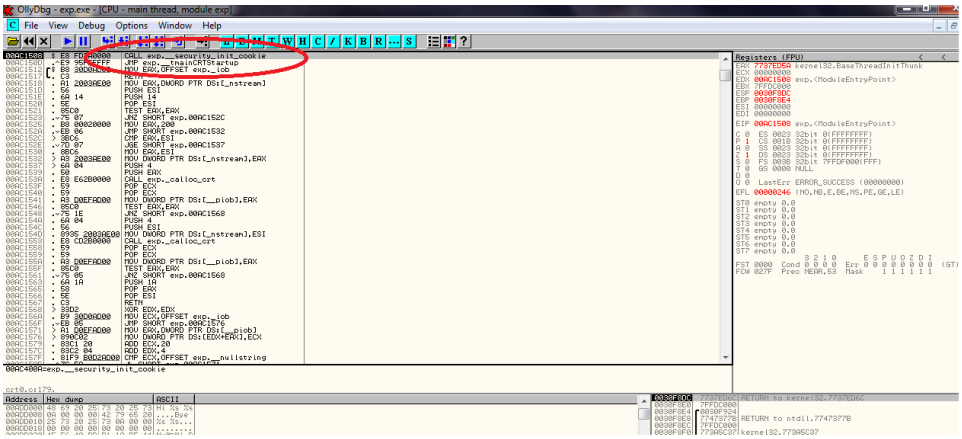

Figure 5.7: Creation of Security init cookie of the GS switch

But the main problem arises if the source code is complied with out the GS switch, then the application is under threat if some type of vulnerability exists. It happens as the GS option is disabled, the Security init cookie is not generated thus the application becomes handicapped, and taking its advantage the intruder can hijack the application. The command to disable the GS switch is: Figure 5.8

When the GS switch is disabled then no security init cookie is generated which makes the program insecure to use. Figure 5.9

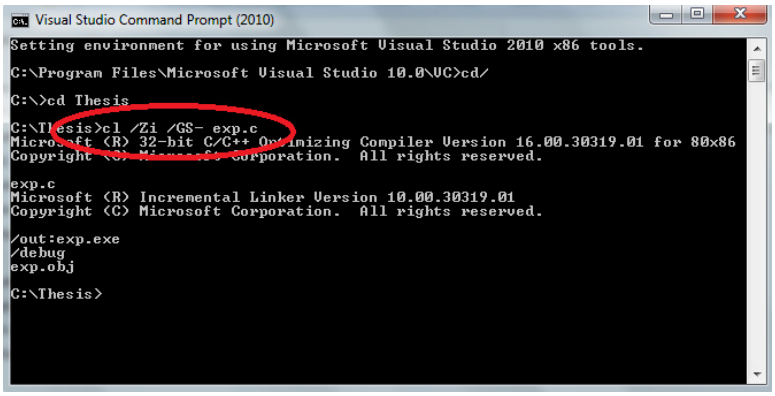

Figure 5.8: Disabling GS switch during compilation

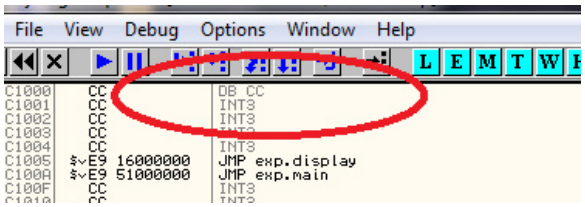

Figure 5.9: Program without security init cookie after disabling GS

Once the GS switch is disabled the attacker has full chances to try the vulnerability of the of the application under usage. The buffer overflow vulnerability could be checked by sending more data than the stack can handle.The following code sends 408 bytes data to 400 bytes buffer Figure 5.10

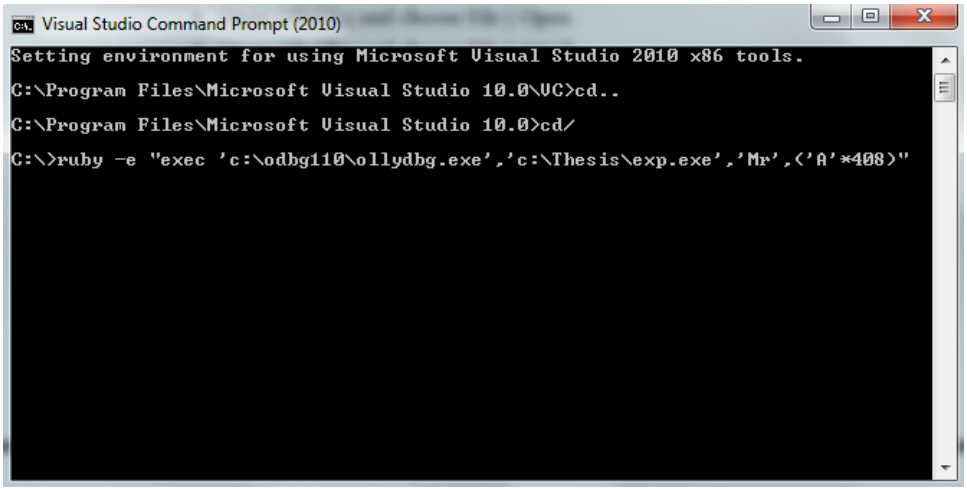

Figure 5.10: ruby command to send 408 bytes to buffer

and ollydbg shows how the program reacts to it as displayed in Figure 5.11 and Figure 5.12

The Figure 5.11 shows the filling up of the stack with A's send to its buffer as it does not contain the GS switch. The Figure 5.12 below shows the stack completely full with

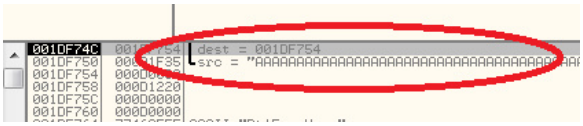

Figure 5.11: Stack receiving 408 A's

408 A's send to attack the bound checking vulnerability and the attacker got success.

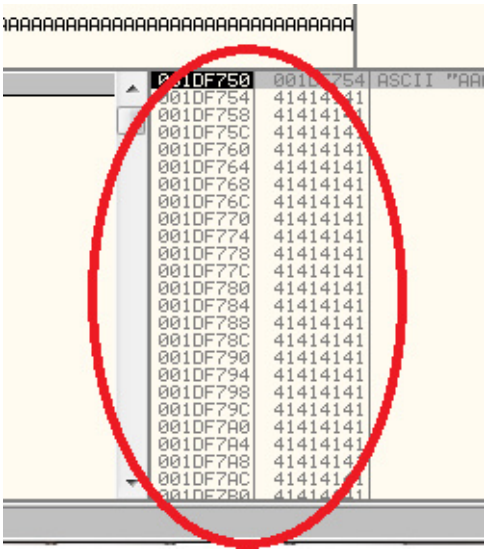

Figure 5.12: Stack Full with 408 A's

Now as soon as the program will be executed firstly the EBP value changes and then the EIP, which becomes 41414141. Hence displaying that the program has crashed. Figure 5.13 and Figure 5.14 display the program crashing.

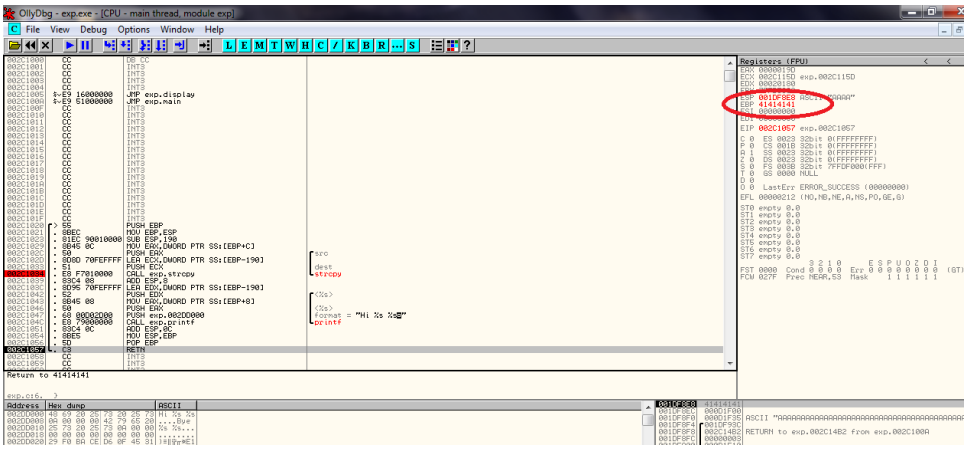

Figure 5.13: EBP value changed

In the above figure ebp value got changed because of the attack on the stack which shows the attacker's malicious code got the control over the registers.

The change in EIP value in the below Figure shows that the attack is complete and

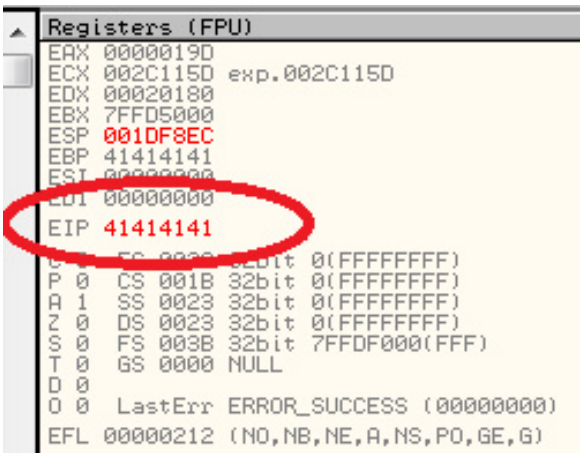

Figure 5.14: EIP value changed to 41414141

now by sending the malicious code the attacker may get control of the victim's system.

Now the above experimentation only involves the exe file to know that whether the application is vulnerable or not or what types of security mechanisms are applied to safeguard it. Thus without knowing the source code of the application its safety measures could be checked.

5.2 SEH complied Applications

As it is already known that SEH complied applications are safe from buffer overflow vulnerability. The SEH handler catches the exception whenever any attacker tries to overflow the buffer. A simple source code in C is used to show case the security of SEH complied applications. In this Thesis a DLL file is created which finds out whether the application under observation is SEH complied or note. This enables the user to find out during the purchase of any software how safe it is to be used. Whether the software is created with the right motives or does it contain any back doors or attacking motives. This could be checked and proves to be very beneficial in various fields of software usage.

A dynamic-link library (DLL) is a module that contains functions and data that can be used by another module it may be an application or some other dll. Its a certain type of file that contains instructions and commands which other programs can call and use to do certain things. This means multiple programs can share the abilities of a single

programmed single file [30].The use of DLL file helps promoting the modularization of code, reuse of code, efficient memory usage, and reducing disk space.

Creating a DLL file involves many steps and proper know how of the system. The basic steps involved are:

1. declspec(dllexport) in the source code
2. An EXPORTS statement in a .def file
3. An /EXPORT specification in a LINK command

To create a DLL file includes the creation of its source file which contains the functions and methods the DLL is going to implement. It involves the internal functions, the external functions and entry point function of the DLL. Various Windows based tools could also be used to develop the DLL. Once the various internal and external functions of the DLL are created its time to link those functions with the linker. Any DLL file contains the export table which includes the names of all functions present in the DLL which are further implemented by other executables. This exporting of DLL file could be done by either creating a .def file or by using the keyword declspec(dllexport) in the function's definition.Further stdcall is made as a final step.

Once the export table is completed, its time to create an import library. The significance of using an import library is to create a file which contains information the linker requires to resolve external references to exported DLL functions, so the system can locate the specified DLL and exported DLL functions at run time [30]. It uses the keyword declspec(dllimport). Then using the Import library during the usage of the DLL enables the executable to use the functions an methods present in the DLL and perform certain task.

The need of explaining creation of a DLL here involves the development and usage of DLL created in this thesis to be used with Ollydbg. The DLL created produces a plugin which enhances the functionality of Ollydbg debugger. The main task of the plugin is to check the presence of SEH handler in the provided executable of an application. It could

be used by copying the appropriate DLL file to the Ollydbg folder and then restarting the Ollydbg provides a plugin option in the menu bar. This plugin contains the SafeSEH option which enhances the functionality of ollydbg by checking the presence of Safe SEH handler in the executable.

The template to create DLL is provided below:

```
#ifndef EXP_STL
#define EXP_STL
#endif
#ifdef EXP_STL
#   define DECLSPECIFIER __declspec(dllexport)
#   define EXPIMP_TEMPLATE
#else
#   define DECLSPECIFIER __declspec(dllimport)
#   define EXPIMP_TEMPLATE extern
#endif
EXPIMP_TEMPLATE template class DECLSPECIFIER CdllTest<int>;
```

Now when the DLL is used in Ollydbg to check an executable, its basic working produces the following results:

Firstly the application is compiled with cl compiler available for Windows application and with a Zi switch and an exe is produced.Figure 5.15

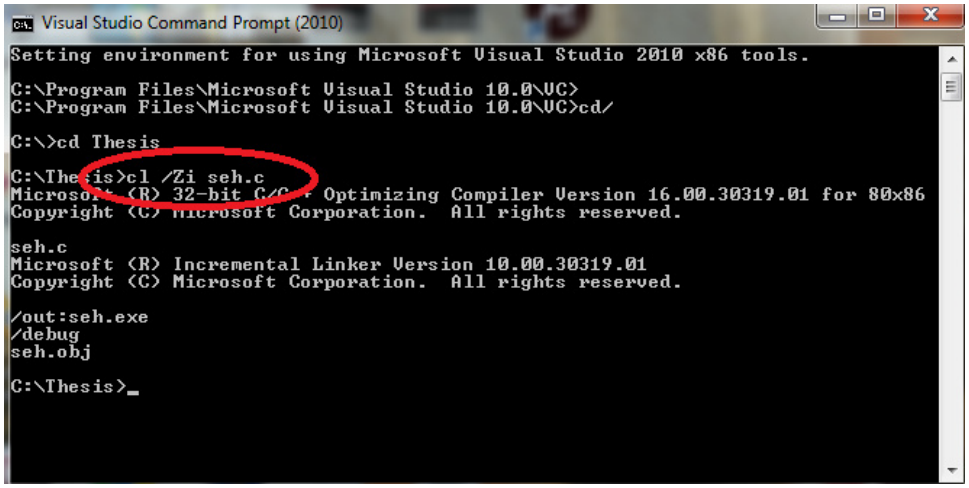

Figure 5.15: Compilation of application to produce its exe

Once the exe is created open it in Ollydbg is observe various characteristics.Figure 5.16. The various register values, various functions, methods, security calls could be studied when the application is opened in Ollydbg. This application contains a basic try and catch method to make the system handle the exception of buffer overflow if occurs. This try and catch method lets the program to call the stack exception handler if required.

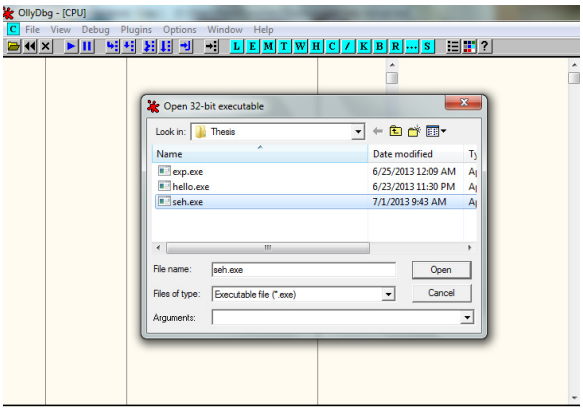

Figure 5.16: Open the exe file of application in Ollydbg

When the application is observed it could be easily seen that SEH handler is active in the application and the SEH handler's init cookie is also initialized. A chunk of addresses in the system are provided to the SEH handler.

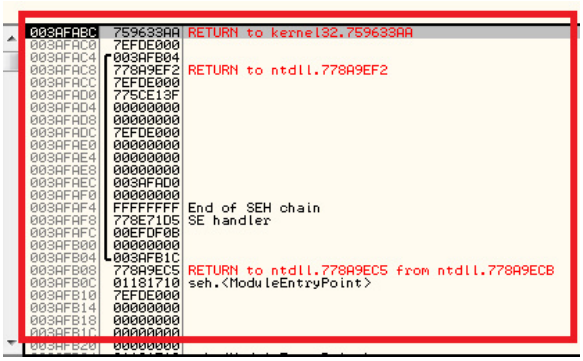

Figure 5.17: SEH handler present in Stack

Figure 5.17 displays the presence of SEH handler in the stack for the protection of the registers. The return to various modules like kernel, ntdll etc. could be seen.

Figure 5.18: Call of SEH security init cookie

Figure 5.18 displays the calling of the SEH security init cookie present in code for security of the application.Once the exe is executed the EIP register also shows the presence of SEH in the application. Figure 5.19

Now here comes the use of the dll created in this thesis work to find out only from

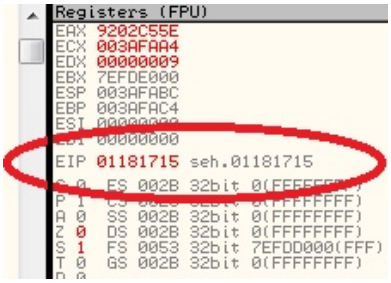

Figure 5.19: EIP register and SEH value

the exe that whether the application is SEH complied or not. When the exe is checked under the dll it shows that whether the SEH switch is on or off. as shown in Figure 5.20

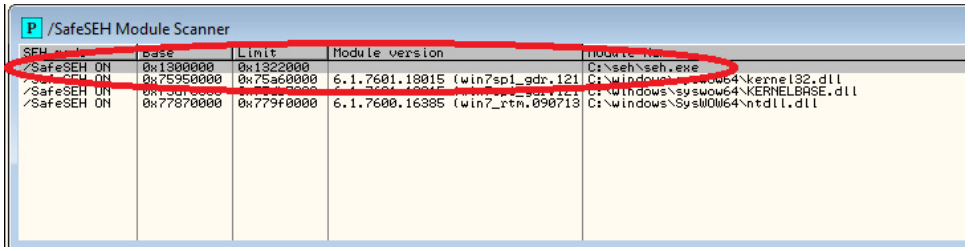

Figure 5.20: SEH switch on in seh.exe

Thus the Figure 5.20 shows that the application is safe to use and does not contain any malicious acts of the attacker.

This could be also be seen and observed in Windbg debugger which is by default present on Windows platform. When the exe is read in this debugger it also enables to dump the SEH handler and see all the values present in stack. Figure 5.21shows application seh.exe debugged in Windbg.

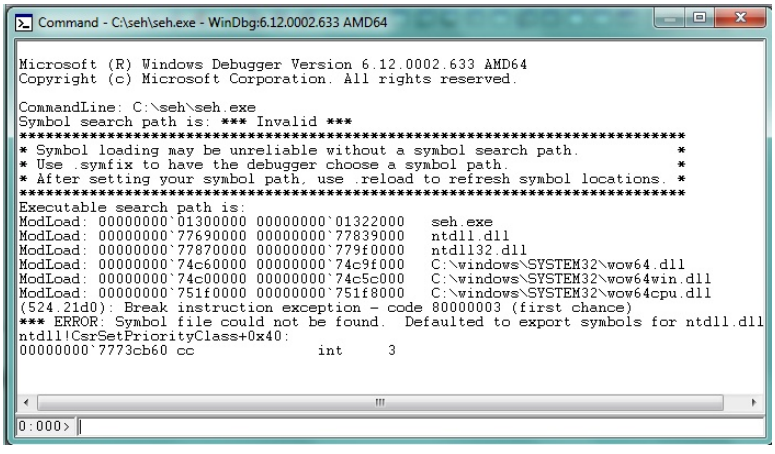

Figure 5.21: seh.exe open in Windbg

Any SEH compiled application contains a signature '64 A1'. This signature is a

proof that SEH handler is present in the exe file under observation. So finding this signature using the command "s 'starting address' l 'last address' '64 A1'" shows the stack containing SEH handler. shown in Figure 5.22

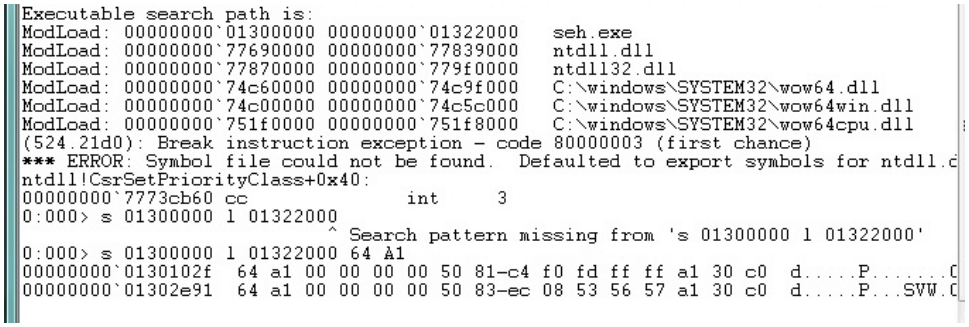

Figure 5.22: Presence of 64 A1 signature in exe file

Using the dump command whole of the SEH stack could be dumped and observed. The main point to notice here is the 'ffffffff' memory address which shows the end of the SEH handler stack in Figure 5.23

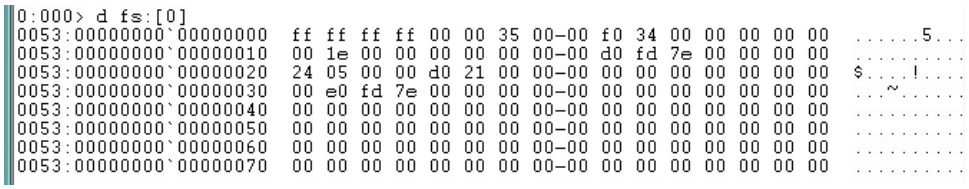

Figure 5.23: ffffffff showing the end of SEH handler

Now it is obvious to learn that if the application is not SEH complied then how it is known. At this point of time the dll plugin created is very useful. First of all just compile the application without the SEH handler. Figure 5.24 shows the required command.

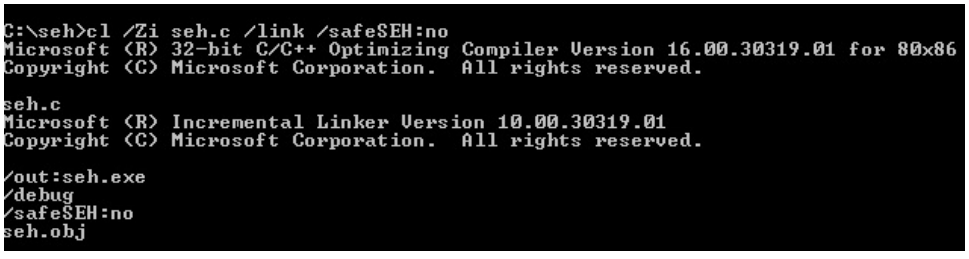

Figure 5.24: Application compiled without SEH

Now this leads to no calling of seh security cookie and no SEH handler is present in the stack and the dll shows the SEH switch as off which leads to a conclusion that even if the user does not have the source of code of the application, only from the exe of the

application its security could be judged and then user may decide whether to use the particular software or not. Figure 5.25, Figure 5.26show the application without SEH.

Figure 5.25: Application without SEH handler

There is no SEH handler present in the above Figure which leads to a conclusion that this application could be vulnerable and may lead to threats in future by the attackers.

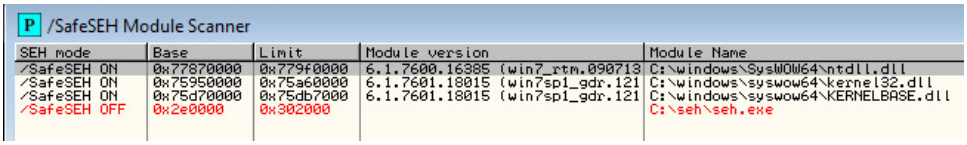

Figure 5.26: Safe SEH Off for the application under use

Safe SEH Off means the attacker has full chance to exploit the application being used. The malicious code send could even provide the attacker with the root privileges and the shell code of the system. This may further lead to destruction of the whole network. So to avoid these exploits, the user must be aware of the various security issues and must use a secured application.

The above Implementation was done on various Platforms of Windows like Windows 7, Windows XP, Windows 2000. Hence it could be concluded that its very important for any user to find out whether the application under use is safe or not, whether it is vulnerable with respect to various threats or not. This lets the user to safeguard the system and thus the network.

Chapter 6

Conclusion and Future Scope

This thesis work shows different techniques helpful in protecting various Windows application from Buffer Overflow. Buffer Overflow is observed to be one of the most common form of attack since many of the programmers producing different source codes does not consider security an important aspect but the latest trends in the attacks by the intruders show the need to emphasize on secure programming skills. Various vulnerable functions and methods used in the source codes must be eradicated.This thesis showcases the implementation of various switches to observe the change in the security of an application with and without the use of various options like GS and Zi.

The main work of this thesis contain the usage of a dll file which tells that whether the application under consideration is SEH complied or not. This enables the user to check the application before hand while purchasing or using it. This checking of the application helps out conclude whether the vendor has properly configured the security switches in the application or not. If yes, the application is safe to use. If not then the user may not use the application as it may contain some malicious code, back doors, and it may provide an ease to the attacker to hijack the root privileges of the system by triggering an attack on it.

Work presented in this thesis automates detection of secure software development technique(s) usage by application vendor prior to installation of the software. It also helps the various programmers working on legacy systems to check if or not the safeguarding

techniques are implemented on the softwares to be used in new systems. The main reason behind this is if the programmers do not check the vulnerabilities present on the legacy systems and use these systems as such then the vulnerability gets lingered on even to the new system, hence making another unsafe system for use.

Moreover the various test cases may be produced by the testing team and the security analyst based on the information provided by the Zi switch. This leads to saving time, resources and money of the organization while producing any software. In Future this work can be augmented to develop software which could enable the user to directly check the various safeguard techniques applied to the executable. As this work is researched for Windows only but can be extended to various other operating platforms.

References

[1] Tony Chan Andrew Cencini, Kevin Yu. *Software Vulnerabilities: Full, Responsible, and Non-Disclosure.* u.washington.edu, 2005.

[2] Alan Paller Dennis Kirby Bob Martin, Mason Brown. *2011 CWE/SANS Top 25 Most Dangerous Software Errors.* Common Weakness Enumeration by SANS and MITRE.org, 2011.

[3] Eric Chien and Peter Szor. *Blended Attacks Exploits,Vulnerabilities and Buffer-Overflow Techniques in Computer Viruses.* Symantec Security Response, 2002.

[4] Calton Pu Steve Beattie Crispin Cowan, Perry Wagle and Jonathan Walpole. *Buffer Overflows:Attacks and Defenses for the Vulnerability of the Decade.* http://www.cse.ogi.edu/DISC/projects/immunix, 2000.

[5] Dave Maier Heather Hinton Jonathan Walpole Peat Bakke Steve Beattie Aaron Grier PerryWagle Crispin Cowan, Calton Pu and Qian Zhang. *StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks.* Ryerson Polytechnic University, 1998.

[6] Lance Cruise. *Preventing the Exploitation of Structured Exception Handler (SEH) Overwrites with SEHOP.* http://blogs.technet.com/b/srd/archive/2009/02/02/preventing-the-exploitation-of-seh-overwrites-with-sehop.aspx, 2009.

[7] cwe.mitre.org. Cwe-193: Off-by-one error, 2013.

[8] David Evans David Larochelle. *Statically Detecting Likely Buffer Overflow Vulnerabilities.* http://lclint.cs.virginia.edu/usenix01.pdf, 2012.

[9] Feiyue Shi Desheng Fu. *BUFFER OVERFLOW EXPLOIT AND DEFENSIVE TECHNIQUES.* 2012 Fourth International Conference on Multimedia Information Networking and Security, 2012.

[10] Wenliang Du. *Buffer Overflow Vulnerability Lab.* Syracuse University, 2012.

[11] G. Andrew Duthie. *What is a buffer overrun?* http://weblogs.asp.net/ gad/ archive/ 2004/03/23/94996.aspx, 2013.

[12] Mark E.Donaldson. *Inside the Buffer Overflow Attack: Mechanism, Method, Prevention.* SANS Institute InfoSec Reading Room, 2003.

[13] Olga Gelbart Bhagirath Narahari Eugen Leontie, Gedare Bloom and Rahul Simha. *A Compiler-Hardware Technique for Protecting Against Buffer Overflow Attacks.* http://www.seas.gwu.edu/ simha/research/HWStack.pdf, 2013.

[14] Dr. Mike Evans. *The Evolution of the Web-From Web 1.0 to Web 4.0.* School of Systems EngineeringUniversity of Reading, 2007.

[15] LIU Feifei. *The Principle and Prevention of Windows Buffer Overflow.* ICCSE, 2012.

[16] Seth Fogie. *Security Reference Guide.* http://www.informit.com/guides/content.aspx g=security seqNum=154, 2003.

[17] fuzzysecurity.com. *Structured Exception Handler(SEH).* http:// www.fuzzysecurity.com /tutorials/ expDev/ 3.html, 2013.

[18] iss.net. *Microsoft Office Web Components ActiveX control buffer overflow (ScriptOWCBO).* http://www.iss.net/securitycenter/reference/vuln/ScriptOWCBO.htm, 2013.

[19] it.med.miami.edu. *Confidentiality, Integrity and Availability (CIA).* University of Miami, 2008.

[20] April Kohl. *Five Types of Buffer Overflow.* http://www.ehow.com/info8295664five-types-buffer-overflows.html, 2013.

[21] Jon Larimer. *An inside look at Stuxnet*. IBM Corporation, 2010.

[22] Nancy Leveson. *Medical Devices: The Therac-25*. University of Washingtion, 1993.

[23] Joanne Lim. *An Engineering Disaster: Therac-25*. Joanne Lim, 1998.

[24] David Litchfield. *Defeating the Stack base Buffer Overflow prevention mechanism of Microsoft Windows 2003 server*. bh asia-03-litchfiled.pdf, 2003.

[25] Hieu T. Luong. *Exploit stack-based buffer overflow using NOP-sled technique*. http://lthieu.wordpress.com/2012/11/10/exploit-stack-based-buffer-overflow-using-nop-sled-technique//, 2012.

[26] Brian Mariani. *Structured Exception Handler EXPLOITATION*. High-tech Bridge Information security solutions, 2013.

[27] Carnegie Mellon. *Security of the Internet*. Carnegie Mellon University, 1998.

[28] msdn.microsoft.com. *Introduction to Spooler Component*. http://msdn.microsoft.com/ en-us/ library/ windows/ hardware/ff551781(v=vs.85).aspx, 2012.

[29] msdn.microsoft.com. *Print Spooler Architecture*. http://msdn.microsoft.com/ en-us/ library/ windows/hardware/ff561105(v=vs.85).aspx, 2012.

[30] msdn.microsoft.com. *Dynamic Link Libraries*. http://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx, 2013.

[31] Clark Turner Nancy Leveson. *TheInvestigation of the Therac-25 Accidents*. IEEE Computer, 1993.

[32] George Varghese Nathan Tuck, Brad Calder. *Hardware and Binary Modification Support for Code Pointer Protection From Buffer Overflow*. 37th International Symposium on Microarchitecture,IEEE, 2004.

[33] Josef Nelien. *Buffer Overflows for Dummies*. SANS Institute InfoSec Reading Room, 2002.

[34] and Eric Chien Nicolas Falliere, Liam O Murchu. *W32.Stuxnet Dossier*. Symantec Security Response, 2011.

[35] Aleph One. *Smashing The Stack For Fun And Profit*. Phrack 49, 1988.

[36] owasp.org. *BufferOverflows*. https://www.owasp.org/index.php/BufferOverflows, 2013.

[37] Gerardo Richarte. *Four different tricks to bypass StackShield and StackGuard protection*. Stack guard.pdf, 2002.

[38] Ken Robinson. *Ariane 5 Flight 501 FailureA Case Study of Errors*. University of New South Wales, 2012.

[39] FENG DengGuo CAO ZhenFu HUANG JiWu SHEN ChangXiang, ZHANG Huang-Guo. *Survey of information security*. Springer-Verlag, 2007.

[40] Ian Sommerville. *The Ariane 5 Launcher Failure*. http://alandix.com/ academic /teaching/ CSC221/ariane.pdf, 2000.

[41] Andrew Soritev. *win32: Safe Structured Exception Handling*. http://www.tortall.net/projects/yasm/manual/html/objfmt-win32-safeseh.html, 2013.

[42] Raphael Tawil. *Eliminating Insecure Uses of C Library Functions*. Systems Group, Department of Computer Science, ETH Zurich, 2012.

[43] thegreycorner.com. *Stack based Windows buffer overflow tutorial*. http://www.thegreycorner.com/ 2010/01/beginning-stack-based-buffer-overflow.html, 2010.

[44] Inc Verisign. *Microsoft Windows 2000 Print Spooler Remote Stack Buffer Overflow Vulnerability*. http://www.verisigninc.com/en US/products-and-services/network-intelligence-availability/idefense/public-vulnerability-reports/articles/index.xhtml?id=806, 2009.

[45] Ollie Whitehouse. *Analysis of GS protections in Microsoft Windows Vista*. Symantec Advanced Threat Research, 2013.

[46] www.sans.org. *SANS Information Security Resources.* SANS, 2013.

[47] www.scip.ch. *Microsoft Windows Print Spooler EnumeratePrintShares buffer overflow.* http://www.scip.ch/en/?vuldb.3988, 2009.

[48] www.yourmaindomain.com. *Uses of Internet.* http://www.yourmaindomain.com /web-articles /use-of-internet.asp, 2008.

[49] www.zdnet.com. *Windows holding steady as PC sales drop.* http://www.zdnet.com /new-usage-stats-show-windows-holding-steady-as-pc-sales-drop-7000016211, 2013.

Publications

"Parul Oberoi, Maninder Singh", "Buffer overflow and Experimental Verification of its Countermeasures in Windows", "International Conference on Computational Intelligence and Information Technology-CIIT" to be held on Oct 17-18, Mumbai. Status-Communicated