International Journal of scientific research and management (IJSRM)
||Volume||2||Issue||8||Pages||1201-1204||2014||
Website: www.ijsrm.in ISSN (e): 2321-3418

Buffer Overflow: Proof Of Concept Implementation

Pooja Rani[1], Dr.Sushma Jain[2]

[1]Computer Science Department
Thapar University
Patiala, India
pooja015.is@gmail.com

[2]Computer Science and Engineering,
Thapar University
Patiala, India
sjain@gmail.com

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 this 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.*

Keywords: BufferOverflow, Minishare, Ability Ftp Server, DEP

1. Introduction

When 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. The buffer can be any section of the memory which is allocated to keep anything from any character string to any array of the integers. The 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. Bufferoverflow can be of these types-

1.1 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 in Figure 1, 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. In the given code in the Figure. 1, 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.

```
#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;
}
```

Figure1 : Example for Stack BufferOverflow

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 addr 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.

1.2 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.

1.3 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 .

1.4 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 .

2. Types of BufferOverflow Attacks

2.1 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.

2.2 ASLR- Address Space Layout Randomization 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.

2.3 DEP- Data Execution Prevention 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. 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

2.4 Stack Shield- Stack Shield 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 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.

3. Related Work

Licker *et al.* proposed a prevention mechanism for the buffer overflow in the java smart cards [1]. The java smart cards used a sandbox based protection schema. Giannetsos and Dimitrio proposed Spy-Sense which was a spyware tool for the execution of the stealthy exploits in the sensor networks [2]. Spy-Sense allowed the injection of the stealthy exploits in the sensor nodes of the sensor network. Alone *et al.* proposed a protection technique for the stack buffer overflow which was based on the duplication and randomization [3]. . Gilbert and Ripoll proposed prevention technique for brute force type attack for canary protection in the networking server [4]. Islander *et al.* proposed a Runtime Intrusion Prevention Evaluator (RIPE) for the prevention of the buffer overflow attacks [5]. Wang *et al*. proposed a Signature-Free Buffer Overflow Attack Blocker known as Segre [6]. Day *et al.* proposed a technology to detect the return to lab buffer overflow attacks by using the Network Intrusion detain system [7]. Gadaleta *et al.* proposed a protection technique for the stack based buffer overflow attacks which were based on the instruction level [8]. Wu and Yongdong proposed protection for buffer overflow in visual studio [9]. Yunnan *et al*. proposed a technology for prevention of global and static type variables from the buffer overflow attack [10]. Francolin and Castelluccia analyzed buffer overflow attacks in the Harvard type of architecture devices [11]. Stojanovski *et al.* proposed technique of buffer overflow attacks by passing the Data Execution Prevention present in the Windows XP [12]. Kong *et al.* proposed protection from buffer overflow by using the taint checking at the instruction level [13]. They proposed an architecture that was based on the instruction level taint checking. Gupta *et al.* proposed dynamic coding instrumentation for detection and protection from the corruption of the return address [14]. Heywood and Kayaked proposed the prevention of the buffer overflow by using the genetic algorithm [15]. Corlis *et al.* proposed dynamic instruction stream editing (DISE) for the protection of the return addresses from the attacks [16]. Inoue proposed an energy security tradeoff for cache architecture to prevent the buffer overflow attacks [17].

4. Implementation

A. In the following work it has been demostrated that how can be bufferoverflow will be performed for the Ability Ftp Server and Minishare-

The buffer overflow in Ability Ftp Server requires

1. Window XP used as a victim machine
2. Ability ftp server installed on victim machine
3. Backtrack machine or any linux used as an attacker machine

The Ability ftp server has the vulnerability with using the STOR command. The given script will send STOR command along with data, which is more than that buffer can handle at the port number 21. Since, Ability Server works on port number 21 so that along with this given script shell code can also placed. This script can also work as proof of concept for buffer overflow attacks on Ability Server.

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

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 buffer= '\x41'*966 + '\x0a\xaf\xd8\x77' + '\x42'*16 + '\x42'*1014

print "\nSending evil buffer..."
s.connect(('ip address of victim machine',21)) # Hardcoded IP Address and port
data = s.recv(1023)
s.send('USER ftp\r\n') # Hardcoded FTP username.
data = s.recv(1024)
s.send('PASS ftp\r\n') # Hardcoded FTP password.
data = s.recv(1024)
s.send('STOR ' + buffer + '\r\n')
s.close()
```

Figure 2: Script for Ability Ftp Server

B. BufferOverflow attacks in Minishare requires

1. Attacker machine i.e. Backtrack or Linux machine
2. Victim machine i.e. Windows
3. Minishare installed on Windows machine

The Minishare has the vulnerability with using the GET command. The given script will send GET command along with data, which is more than that buffer can handle at the port number 80. Since, Minishare works on port number 80 so that along with this given script shell code can also placed. This script can also work as proof of concept for buffer overflow attacks on Minishare.

```
#! usr/bin/python
import socket, sys
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer="GET"
buffer += "\x90"*1788
buffer += "\x65\x82\x85\x75"
buffer+= "\x90"*16
sock.send(buffer)
sock.close()
```

Figure 3: Script For Minishare

References

[1] 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.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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. 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

Author Profile

1. Pooja Rani is currently pursuing her Mtech under Computer Science Department in Thapar University, Patiala under the supervision of Dr. Sushma Jain . Her specialization is in information and Network Security