International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 06 | June - 2022 Impact Factor: 7.185 ISSN: 2582-3930

Buffer Overflow Vulnerabilities and Prevention

Shreenivas Hegde , Assistant Prof. Rakshitha Sumanth

Department of MCA

Dayananda Sagar college of engineering

Abstract

Either in terms of software error or being an attack itself, buffer overflow attacks have been one of the most important security problems accountable for a common vulnerability of software security and cyber risks. Buffer overflow vulnerabilities influence the zone of remote network penetration vulnerabilities, where an anonymous user tries to gain partial or complete control of the host. Buffer overflow attacks liable for some of the biggest data breaches in recent years. One of the notable data breach examples include Morris worm or internet worm of November 2, 1988. Morris worm is an ancient computer worm distributed via the internet, was noted for infecting around 6000 major Unix machines. This paper mainly focuses on buffer overflow attack vulnerabilities and the preventive measures to safeguard once system from being attacked.

Keywords – Buffer overflow attack, susceptibility, impediment

1. Introduction

In recent years, the buffer overflow threat paved the way for a severe form of vulnerability in software and became the reason for critical issues such as network security. Most of the buffer overflow issues are due to the poor programming practices. Almost all web servers, web application servers and web applications are receptive to buffer overflow attack. Applications that are written in interpreted languages, such as Java and Python, are immune to the attacks, with the exception of overflows in their interpreter. Buffer overflow is a software related coding error or loophole that can be exploited by a hacker to gain unauthorized access to the network systems. Buffer overflow occurs when the volume of the data exceeds the storage capacity of the memory location, causing overwriting of neighbouring memory locations. As a result, the program behaves unpredictably and generates false results leading to system crashes and memory access errors. Buffer overflow attack takes place when the attacker tries to manipulate the code execution path in order to compromise the affected system. For example, an attacker tries to introduce an extra code, consigning new instructions to the application to gain access to IT systems. If attackers know the memory format of a program, they purposely send input that a buffer cannot store and try to overwrite the executable code replacing it with their own code. This attack is common and most of the preventive measures are prone to error.

A buffer overflow vulnerability will predominantly occur when:

1. The code is contingent on users input which could control its behaviour.

2. The code is reliant on data properties that are enforced beyond data scope.
3. The code is complicated for programmers to predict its behaviour accurately.

In this process the anomaly forces to store the data in a buffer outside the programmer's designated memory and hence affects the program flow control. This malicious computer worm propagates themselves from machine-to-machine causing security breaches. This security breach has recorded more than 50% of the incidence compromising computer network security.

2. Literature survey

The problem arises from the basic fact that how the computer would manage memory. A stack is a continuous block of live memory that holds units of information which would be processed by performing operations. Within the stack are areas called buffers into which data can be input dynamically, as opposed to pre-programmed data units. Many operating systems and programming languages were developed at the time when security was not a primary concern. Unfortunately, these systems didn't have any mechanism to block or truncate oversized data. Excess data would overflow to the neighbouring memory location and benefit the attacker. A clever attacker would manipulate this flaw and try to execute malicious code where the system will run into alien instructions[7].

This thread was first marked in a US Air force planning study on computer security published in October 1972, where this flaw remained embedded in computer systems across the country. The situation was different in 1988 where the home computing Revolution and the rise of the internet vastly expanded the number of systems. That was the year where a major viral attack on the Internet took place. A self-replicating program called a worm copied itself across thousands of computers throughout the country within a span of hours infecting the hosts. Till now a staggering number of systems remained vulnerable to these attacks[7].

3. Types of Buffer overflow attacks

Different buffer overflow attacks use different Strategies and target different pieces of code. They are categorized as four major types:

3.1. Stack overflow attack

The stack follows LIFO fashion in a data structure, which means last in first out. This supports two operations called push and pop. Push operation is used to specify the value of the Steel whereas pop operation is used to extract the value of the Steel. If the data set on the stack is malicious, they will try to overwrite the adjacent memory locations and affect the data that or pointer that is stored by the other program. An attacker makes use of this vulnerability and try to exploit the system by manipulating data or creating a pointer to run hazardous code[6].

3.2. Heap overflow attack

The heap overflow takes place when part of memory is assigned to the heap and data is written to the memory without being checked. As a result, the critical data structures in the heap such as heap headers, dynamic object pointers can overwrite the virtual function table[6].

3.3. Integer overflow attack

As a result of an arithmetic operation, an integer result overflows and the result does not lie in the allocated memory space. This leads to integer overflow. For example, the space assigned for 32-

bit integer data type may be an unsigned integer between 0 and 4294 967 295 or a signed integer between - 2147 483 648 and 2147 483 647. When you try to calculate 4294 967 295 + 1 and try to store the resultant data that exceeds the maximum value for the whole data type, it eventually crashes the system and puts you at risk of attack. integer overflow attack simply leads to Inaccurate program behaviour and does not cause major vulnerabilities. But in some circumstances they cause severe damage[6].

3.4. Unicode overflow

Unicode strings have been widely used across the world to ensure that any language can be transcript without a problem. For example, Korean characters are different from English characters. On realizing the fact such characters could not be converted according to the ASCII codes. Hence the Unicode strings are introduced where it allows the user to take advantage of the program by typing Unicode characters in input that expects ASCII characters. It simply provides input that surpasses the maximum limit to make a buffer overflow with uncertain characters of Unicode where the program is expecting ASCII input[6].

4. Buffer overflow with example

A buffer is a temporary storage area. When considerably more data gets placed by processor then the extra data overflows and leak out into other buffers. The exceeding data unusually holds specific instructions for actions intended by an hacker[3]. Let's take up an example where we don't insert any malicious code but describe how buffer can overflow.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argument1, char *arg[])
{
        char buffer[5];
        if (argument1 < 2)
        {
                printf("strcpy() NOT executed....\n");
                printf("Syntax: %s <characters>\n", arg[0]);
                exit(0);
        }
        strcpy(buffer, arg[1]);
        printf("buffer content= %s\n", buffer);
        printf("strcpy() executed...\n");
        return 0;
}
```

Input1:

```
buffer content= 12345678
strcpy() executed...
```

Input2:

```
buffer content= 12345678919
strcpy() executed...
*** stack smashing detected ***: terminated
```

The vulnerability is present since, the buffer could be overflowed if the user's input (arg[1]) is bigger than 8 bytes. For the system with 32-bit , we must fill up a double word that occupies (32 bits) memory. Character occupies 1 byte, so if we request buffer having 5 bytes, the system will assign 2 double words (8 bytes). Hence, when you input more than 8 bytes; the buffer will be overflowed. Similar standard built-in functions that are technically less vulnerable, such as strncpy(), strncat(), and memcpy(), can also be used in the programming. It depends upon the programmer to provide reasonable size of the buffer and not on the compiler[3].

5. Impediment

The code injection and the exploit need not have to happen in one shot. The hacker can insert code into one buffer without leaking it into the adjacent memory locations, and overflow a different buffer to exploit the code pointer. There are certain defensive mechanisms Against buffer overflow vulnerabilities and attacks. Some are mentioned below.

5.1. Writing the correct code.

Some programming languages like C which is error-prone idioms such as night terminated strings which favours the performance except correctness. But there are some tools that help programmers to write a code that is less likely to contain buffer overflow vulnerabilities. One of such methods is to grep the source code for highly vulnerable library calls messages strcpy() and sprintf() which does not check the length of the arguments, these functions make the program vulnerable[1]. In order to combat these issues, more advanced debugging tools such as fault injection tools have been developed, which includes inserting buffer overflow randomly to search for vulnerable components in the program.

5.2. Non-executable buffers.

The concept here is to make the victims programs address space non executable, making it impossible for attackers to execute the code they inject into the victim's programs buffer[1].

5.3. Array bound checking.

If arrays cannot be overflowed, then array overflows cannot be used to exploit adjacent programming states. To implement array bounds checking, then all inputs to arrays needs to be checked to make sure that they are within range. The simple and direct method to eliminate the issue is by implementing optimization techniques into these checks. There are certain approaches to implement array bound checking such as Compaq C Compiler, jones & kelly's array bound checking, type-safe languages etc[1].

5.4. Code pointer integrity checking.

The code pointer integrity checking seeks to Detect that a code point that has been corrupted before it is dereferenced. So, when that succeeds in corrupting a code pointer, the corrupted code pointer will never be used due to its detection before each use[1].

5.5. System compatibility and performance.

The virtual methods make indirect function calls common place, yet may access arrays more often than virtual methods, depending on the application. Hence this system performance must be considered along with implementation effort where array references are no longer simple pointers but become pointers to buffer descriptors[1].

6. Conclusion

This was a detailed presentation and analysis of A buffer overflow vulnerability, attacks and preventive measures. Buffer overflows are worthy to be considered before developing any system or constructing network a first. They cause major penetration security vulnerability issues. The buffer overflow attack technique is both unusual in contemporary, also not easily defended against existing attack methods. But considering the above-mentioned preventive measures could Possibly reduce the chance of being prone to buffer overflow attack.

References

1. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole," Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade".
2. Samah Muhammed S. ALHusayn ,"The Buffer Overflow Attack and How to Solve Buffer Overflow in Recent Research".
3. https://www.geeksforgeeks.org/buffer-overflow-attack-with-example/
4. https://www.fortinet.com/resources/cyberglossary/buffer-overflow
5. https://www.imperva.com/learn/application-security/buffer-overflow/
6. https://www.wallarm.com/what/buffer-overflow-attack-definition-types-use-by-hackers-part-1
7. https://media.thinkbrg.com/wp-content/uploads/2021/03/19180113/33_Sleepy-History-of-Buffer-Overflow_Oct2020.pdf