Hacking computer memory with C, Buffer Overflow Attack

Article · January 2023

CITATIONS
0

READS
280

2 authors:

Amirhossein Najafizadeh
Amirkabir University of Technology
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Asal delkhosh
Amirkabir University of Technology
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Hacking computer memory with C, Buffer Overflow Attack

Amirhossein Najafizadeh
Amirkabir University of Tehran, Iran
December 2022
najafizadeh21@gmail.com

Abstract

Hacking computer memory[1] with C programming language, but how is this even possible? In this article we are going to hack our system memory with C programming language to generate a buffer overflow attack in the system, but before doing that we are going to talk about what is buffer overflow and how it can make vulnerabilities for our applications or system. We are going to talk about what is actually happening when buffer overflows and then we are going to talk about its consequences. In the next parts we are going to access to other parts of memory with C programming language by giving an example with real code and then talk about how does C programming language allow us to access every part of memory? How we can secure our system against this problem? What will happen if we don't come with a solution to control this vulnerability? Why can't we access to any part of memory in other programming languages like Java and Go[2]? What makes a programming language safe against buffer overflow attack? After reading this research you will know everything about buffer overflow attack and how to stop it.

[1] In here or any other place of this research memory is a reference to RAM (random access memory), since our program data is being stored on RAM.
[2] Go programming language (Golang).

1. Buffer Overflow

Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.

For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary.

Buffer overflows can affect all types of software. They typically result from malformed inputs or failure to allocate enough space for the buffer. If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes.

1.1. Buffer Overflow Example

In the following part, we are going to give an example of buffer overflow. If you run the sample file, you can see that in this program we can access memory parts out of array allocated space in memory. The program will execute without any errors or warnings. Allow us to access any element out of array allocated space.

```c
// let's see what happens if we update the 11th element.
#include <stdio.h>

int main() {
  // creating an array of size 10.
  int memory[10];

  // now trying to access the 11th element.
  printf("%d\n", memory[10]);

  // let's see what happens if we update the 11th element.
  memory[10] = 10;

  // let's see what happens if we access the 11th element again.
  printf("%d\n", memory[10]);

  return 0;
}
```

1.2. What is actually happening?

What is happening in the computer is we are accessing the memory parts that we are not allowed by increasing the pointer address. The array pointer is pointing at a part of memory and you can move it to the next house of array or give an index and it will sum up the current position with your index and returns the value of the result memory house. But we are increasing the pointer more than we should, this is where we get Buffer Overflow.

Our system does not have any idea that your programming is going beyond its allowed limits, hens it cannot stop it from accessing the forbidden memory locations. But how this can cause a problem for the system?

1.3. Consequences

This access may subtly corrupt other memory used by your program[3], or may cause an immediate segmentation fault depending on how that particular array is laid out in memory. Buffer overflows can affect all types of software. They typically result from malformed inputs or failure to allocate enough space for the buffer. If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes. It can also cause buffer overflow attack.

2. Buffer Overflow Attack

In a Buffer Overflow attack, the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.

Attackers exploit Buffer Overflow issues by overwriting the memory of an application. This changes the execution path of the program, triggering a response that damages files or exposes private information. For example, an attacker may introduce extra code, sending new instructions to the application to gain access to IT[4] systems.

[3] In here we are talking about programs that can access to memory parts.
[4] Information Technology

If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite a pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program.

2.1. Types of Buffer Overflow Attacks

There are two types of Buffer Overflow attack, Stack[5]-based and Heap[6]-based.

Stack-based buffer overflows are more common, and leverage stack memory that only exists during the execution time of a function.

Heap-based attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations.

2.2. Which programming languages are more vulnerable?

C and C++ are two languages that are highly susceptible to buffer overflow attacks, as they do not have built-in safeguards against overwriting or accessing data in their memory. Mac OSX, Windows, and Linux[7] all use code written in C and C++.

Languages such as PERL, Java, JavaScript, and C# use built-in safety mechanisms that minimize the likelihood of buffer overflow.

[5] Stack memory is used only by one thread of execution.
[6] Heap memory is used by all the parts of the application.
[7] All distributions

3. How to Prevent Buffer Overflows?

Although handling the buffer overflow seems hard, but there are ways to prevent our systems from buffer overflow attacks.

3.1. Bound indexing

Since you didn't show any code, the answer can only be a general one: Stay inside the bounds of the array.

Apart from accessing at some wildly out of bounds position, one particular case is more common: If you have an array with a size of 10 then 10 isn't a valid index. Because arrays in C and C++ are 0-based. So, in this case, valid indices are 0 to 9.

3.2. Use dynamic lists

You must not go out of bounds, the C/C++ developer has to be precise. That said, you could use *std::vector* instead of a plain array: it provides the *std::vector::at* method that throws an exception[8] if you try to make an out-of-bounds access.

3.3. Address space randomization (ASLR)

Randomly moves around the address space locations of data regions. Typically, buffer overflow attacks need to know the locality of executable code, and randomizing address spaces makes this virtually impossible.

3.4. Data execution prevention flags

These flags certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region.

[8] Exceptions are runtime errors.

3.5. Structured exception handler overwrites protection[9]

Helps stop malicious code from attacking Structured Exception Handling[10], a built-in system for managing hardware and software exceptions. It thus prevents an attacker from being able to make use of the SEH overwrite exploitation technique. At a functional level, an SEH overwrite is achieved using a stack-based buffer overflow to overwrite an exception registration record, stored on a thread's stack.

Security measures in code and operating system protection are not enough. When an organization discovers a buffer overflow vulnerability, it must react quickly to patch the affected software and make sure that users of the software can access the patch.

[9] SEHOP
[10] SEH

References

[1] Anderson, D. (2022, July 1). *Buffer Overflow Attack with Example*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/buffer-overflow-attack-with-example/

[2] Lefevre, S. (2016). *Avoid Out Of Bounds Array*. CodeProject.

[3] MacKeveer, H. (2020). Buffer Overflow Attack. *Imperva*, 5.

[4] TheWMoe. (2016, July 17). *Array allows out of bounds access in C*. Retrieved from StackOverflow: https://stackoverflow.com/questions/38416792/array-allows-out-of-bounds-access-in-c