Integer Pointers Program and Analysis

Raneem Kazma

Colorado State University Global Campus

Pointers are used to store the memory address of a variable in another variable. There are several advantages to utilizing pointers such as efficient program code, faster execution and memory saving. Pointers can also be used to dynamically allocate memory. Despite its advantages, using pointers creates vulnerabilities and room for exploitation in code. Pointers can become threat to code integrity when they become dangling, void or null. As my code utilizes pointers that point to three different user-input variables, it may be vulnerable to attacks. (Backman K., 2012) The following paper will discuss the vulnerability of dangling pointers.

A dangling pointer points to memory location that has been cleared either by the programmer or the compiler. Dereferencing a dangling pointer could cause a program to crash or exhibit unusual behavior. Dangling pointers become at risk of exploitation, however, when a memory location is deallocated twice. Typically, free chunks of memory are stored in a linked list in order of size. If a chunk with the same size as chunk 2, it is inserted in between chunk 1 and chunk 2 by reassigning the forward and backward pointers. If the same memory chunk it freed twice, its forward and backward pointers will point to itself, rather than a new freed memory chunk. When the system requests memory space, It will assume it had unlinked that memory chunk but in reality it had not as its pointers are pointing to itself. Attackers can now have the forward pointer point 12 bytes before the return address and the backward pointer point to code that will jump over the bytes to be overwritten. If the compromised memory chunk is allocated by a program, it will remain linked and overwrite the return address with the code from the chunk’s backwards pointer. (Younan Y., 2004)

To counter measure this vulnerability, some languages have removed explicit control over memory allocation and introduced garbage collection instead. Some compilers also provide bound checking to protect against dangling points by storing the state of pointers. If a pointer is deallocated it is marked so, and any attempts of further dereferencing will result in an error. Additionally, sandboxing, encryption, randomization and non-executable memory can protect against dangling pointer vulnerabilities and injected code. (Younan Y., 2004)

References:

Backman, K. (2012). Structured programming with C++. Bookboon.com. Retrieved from

http://bookboon.com/en/structured-programming-with-c-plus-plus-ebook

Younan, Y., Joosen, W., & Piessens, F. (2004). *Code Injection in C and C : A Survey of*

*Vulnerabilities and Countermeasures*. Belgium: Department of Computer Science, K.U.Leuven.

Code and Execution

#include <iostream>

**using** **namespace** std;

**int** main(**int** argc, **const** **char** \* argv[]) {

**int**\* ptr1;

**int**\* ptr2;

**int**\* ptr3;

**int** v1;

**int** v2;

**int** v3;

cout<<"Enter a number" << endl;

cin>>v1;

ptr1 = **new** **int**(v1);

cout<<"The address of v1 is "<<ptr1 << endl;

cout<<"The value of v1 is "<< \*ptr1 << endl;

cout<<"Enter a number" << endl;

cin>>v2;

ptr2 = **new** **int**(v2);

cout<<"The address of v2 is "<<ptr2 << endl;

cout<<"The value of v2 is "<< \*ptr2 << endl;

cout<<"Enter a number" << endl;

cin>>v3;

ptr3 = **new** **int**(v3);

cout<<"The address of v3 is "<<ptr3 << endl;

cout<<"The value of v3 is "<< \*ptr3 << endl;

**delete** ptr1;

**delete** ptr2;

**delete** ptr3;

**return** 0;

}

A screenshot of a cell phone

Description automatically generated