As my code contains pointers form the main function to a method, my code is vulnerable to memory subversion and injection attacks. In this type of attack, virtual tables (vtables) can be accessed, modified or completely corrupted (Elsabagh, 2017). Vtables are read only tables that contain pointers to function definitions in the binary. C++ is particularly vulnerable to injection attacks as all C++ compilers make use of vtables to support polymorphism. Elsabagh has proposed a Virtual Call Integrity, VCI, check to protect binaries from vtable attacks. The system works by constructing a strict control flow integrity policy that pairs function calls with precise classes (Elsabagh, 2017). According to Elsabagh this security threat in C++ is still prevalent.

Another vulnerability in my code is buffer overflow. There 2 areas where my code is vulnerable; first, my code takes input from the user and second my code copies one string into another. As user input size cannot be guaranteed, a solution to buffer overflow is to use string objects rather characters arrays to capture text. Another way to avoid buffer overflows is to restrict the size of the input. This can be implemented using the C++ standard library, which contains containers that check the input bounds and throw an out of bounds exception when necessary (std::vector::). Another way to avoid a buffer overflow is to avoid unbounded built in functions such as strcpy, strcat, sprint and printf (Buffer Overflow & Format String Attacks). Buffer overflows can also be prevented by implementing a non-executable stack. Stacks usually store function call arguments and local variables rather than executable code. By implementing a non-executable sack, we could prevent against malicious code infiltrating the stack in case of a buffer overflow (Buffer Overflow & Format String Attacks).

Another type of attack is man in the middle. This is possible in my code as one string is transferring its contents to another string. During this data transfer operation, an additional operation can be inserted to copy, manipulate or corrupt the data.

Reference:

Elsabagh, M., Fleck, D., & Stavrou, A. (2017). Strict Virtual Call Integrity Checking for C++ Binaries. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. New York, NY, USA, 140-154.

std::vector::at. (n.d.). Retrieved from <http://www.cplusplus.com/reference/vector/vector/at/>

“Buffer Overflow & Format String Attacks: The Basics.” *Infosec Resources*, 31 May 2017, resources.infosecinstitute.com/buffer-overflow-format-string-attacks-basics-part-1/#gref.

**Code and Code execution**

#include <iostream>

#include <string>

using namespace std;

string reverse(string name){

string removed;

string reversedName = name;

int index = name.length()-1;

for(int i = 0; i < name.length(); i++){

reversedName[i] = name[index - i];

}

return reversedName;

}

int main() {

string name;

int counter = 0;

string output;

while (counter < 3){

cout<<"Please enter a name"<<endl;

cin>>name;

output = reverse(name);

cout<<"the reversed name is "<<endl;

cout << output << endl;

counter++;

}

return 0;

}

A screenshot of a cell phone

Description automatically generated