Journal of Advanced Zoology

ISSN: 0253-7214
*Volume **45** Issue **S-4** **Year 2024** Page **91-95***

Alleviating Security Deficiencies In C++ Code

Prof. Shivendu Bhushan[1*], Prof. Ganesh Bhondve[2]

[1,2] Indira College of Commerce & Science.*
Email: shivendu@iccs.ac.in[1], ganesh.bhondve@iccs.ac.in[2]

***Corresponding Author:** *Prof. Shivendu Bhushan*
Email: shivendu@iccs.ac.in

Abstract

C++ is a broadly used programming language known for its versatility in handling diverse tech projects. However, the increasing prevalence of security deficiencies and threats poses significant challenges in the industry, hamper to work processes and technical damage to developers. In this paper we will be studying alleviating security deficiencies in C++ and avoiding deficiencies.

Keywords: Deficiencies, Dereferencing, Null Pointer, Buffer Overflow, Memory Outflow, Insecure Input, Referencing, Data Overflow, Data Underflow.

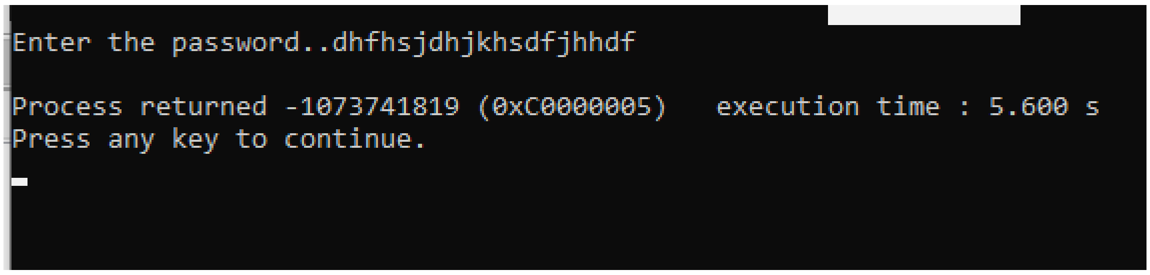
Introduction:

There are many scenarios where we have security deficiencies in C++ code. When we are programming in C++ we may face issues, bugs at runtime while memory allocation and memory deallocation.
In this paper listed deficiencies in C++ programming and also avoidance of mentioned deficiencies for better efficiency of code.

List of Deficiencies Programs in C++ Languages:

1. Buffer Overflow
Buffer overflow deficiencies can lead to unauthorized access, data manipulation, or even code execution

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
  // Allocate 6 bytes of buffer and also 0 plus the terminating NULL chara.
  //it  should allocate 8 bytes
  // To overflow, need more than 8 bytes...
  char buff[6]; // If more than 8 characters input
  // by user, there will be access
  // runtime segmentation fault
   char str[15];
   cout<<endl<<"Enter the password..";
   cin>>str;
  // coping user input to our, but its done bound checking
  // for this secure will be strcpy_s()
  strcpy(buff, str);
  printf("Tempory buffer data= %s\n", buff);
  // For safer handling we can use strcpy_s()
  printf("strcpy() has been succesfully executed...\n");
  return 0;
}
```



```
Enter the password..dhfhsjdhjkhsdfjhhdf

Process returned -1073741819 (0xC0000005)   execution time : 5.600 s
Press any key to continue.
```

Best practices to help you avoid buffer overflow deficiencies

1. Use safe standard library function that does bound checking std::vector or std::string instead of raw arrays.
2. Use safe library function strncpy, snprintf etc
3. Avoid raw arrays: Use latest arrays std:: array or std::vector for size and bound checking
4. Use of memory safety tool like valgrind or addresssanitizer
5. Restrict use of unsafe function like gets, strcpy, sprintf etc

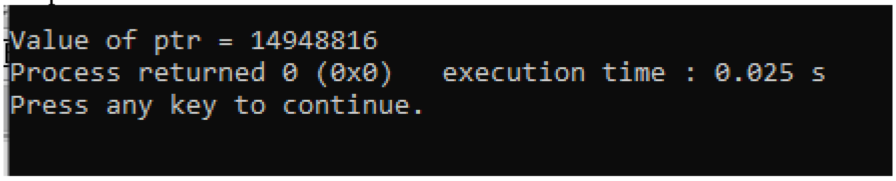
2. Memory Outflow

Memory leak generally occurs when we dynamically allocate memory but not freed

Code:

```
#include<iostream>
using namespace std;
int main()
{
   // Memory outflow code
   int *ptr = new int[6];

   // forgotten to free allocated memory
   // need to free dynamic array using delete operator
   cout<<endl<<"Value of ptr = "<<*ptr;
   return 0;
}
```

Output:



```
Value of ptr = 14948816
Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

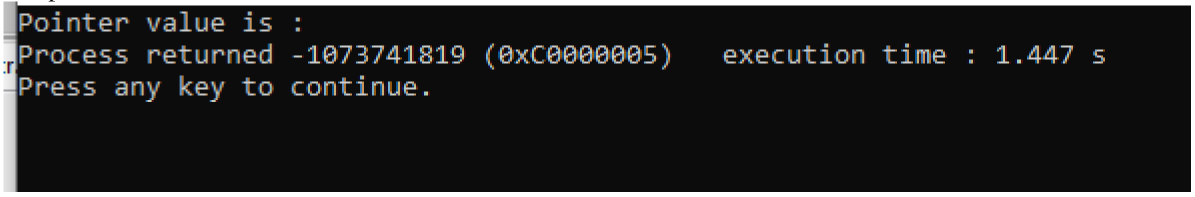
In the above program allocated the memory using new operator but forgotten to freed memory using delete. To avoid memory outflow we should use delete to free memory i.e. [] delete operator.

3. Null Pointer Value

Null pointer value access occurs due to program is accessing or modifying the value where pointer which is set to NULL. Because of this program unpredictable behavior, crashes, or some deficiencies issues occur

```
#include<iostream>
using namespace std;
int main()
{
  int  *ptr = NULL;
  cout << "Pointer value is : " << *ptr ;
  return 0;
}
```

Output:



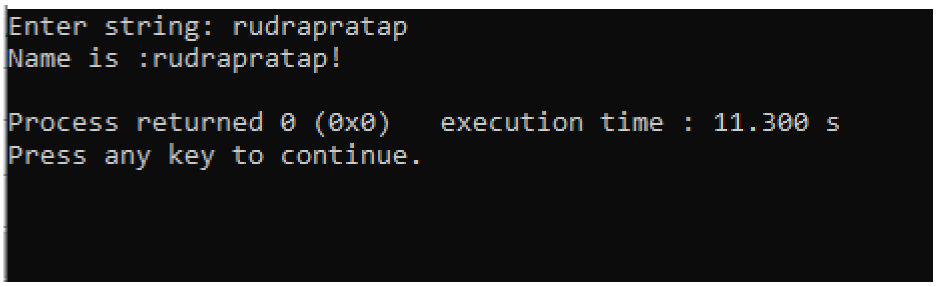
```
Pointer value is :
Process returned -1073741819 (0xC0000005)   execution time : 1.447 s
Press any key to continue.
```

One solution to avoid Null pointer value is to check in the condition whether pointer is NULL or not and then print the value. We can also use smart pointers for this or safe memory management techniques for this.

4. Insecure Input Handling

If we handle input insecurely then there will be chance of injection attack or buffer overflow

```
#include<iostream>
using namespace std;
int main()
{
    char tname[4];
    // Insecure input handling - deficiencies to buffer overflow
    cout << "Enter string: ";
    cin >> tname;
    // Insecure use input
    cout << "Name is :" << tname << "!" <<endl;
    return 0;
}
```



```
Enter string: rudrapratap
Name is :rudrapratap!

Process returned 0 (0x0)   execution time : 11.300 s
Press any key to continue.
```

In the above code we have accepted the input without checking the length of the input. Because of this there can be buffer overflow if the users enter more characters there might be chance of deficiencies.
To solve above issue we should use safer input functions in code such as getline() function or fail() function to avoid buffer overflow.

5. Data Overflows and Underflows

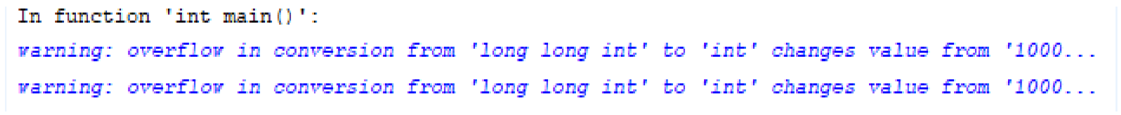
Data overflows comes if the result of arithmetic operation goes maximum range value of data type and Data underflow occurs when data is smaller than the data type can hold

In c++

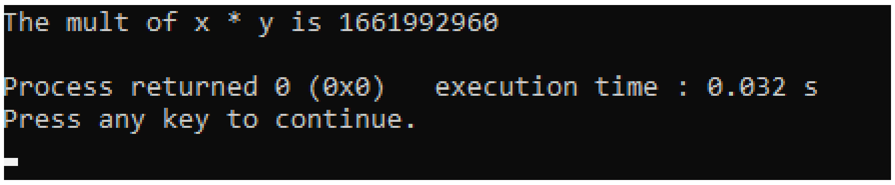
1. signed int: The signed int data type ranges between -2,147,483,648 to 2,147,483,647 (-109 to 109).
2. unsigned int: The unsigned int data type ranges between 0 to 4,294,967,295.

3. long long: The long long data type ranges between -9, 223, 372, 036, 854, 775, 808 to 9, 223, 372, 036, 854, 775, 807 (-1018 to 1018).

```
#include <iostream>
using namespace std;
int main()
{
   int x = 10000000000;
   int y = 10000000000;
   int z = x * y;
   cout << "The mult of x * y is " <<
   z << endl;
   return 0;
}
```



```
In function 'int main()':
warning: overflow in conversion from 'long long int' to 'int' changes value from '1000...
warning: overflow in conversion from 'long long int' to 'int' changes value from '1000...
```

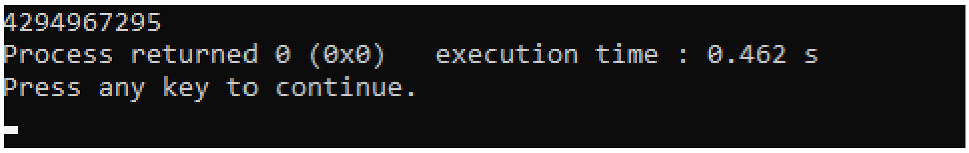


```
The mult of x * y is 1661992960

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

If we can solve above issue using proper data type as long long or LL extension also we can initialize all variables to long long
long long z = x * 1LL * y;

Data Underflow

```
#include <iostream>
using namespace std;

// underflow
int main()
{
     unsigned int x = 3, y = 4;
     unsigned int z = x - y;
     cout << z;
     return 0;
}
```



```
4294967295
Process returned 0 (0x0)   execution time : 0.462 s
Press any key to continue.
```

We think that value will be -1 but the output is 4294967295. This is because unsigned int c does not store negative store.
Above problem can be solve by using signed data type where negative number can be store. to handle underflow error.

Conclusion:

While development in C++ programming language there are many cases where we use pointers, memory allocation, deallocation. When we want to use these functionality at that time deficiency can be there in our code.
So to avoid these deficiency and potential crash our C++ program we need to take care about Memory outflow, Data overflow, Data underflow, Insecure input handling, Null pointer value and Buffer overflow deficiencies and while harm to our development. so while dealing with the above situations we need to considered solutions as discussed below.

References:

1. Programming Language Pragmatics(Michael L. Scott)
2. C++ Primer by Stanley B. Lippman, Jossee Lajoie, and Barbara E. Moo
3. Jumping into C++ Kindle Edition By Alex Allain
4. Accelerated C++: Practical Programming(By Andrew Koenig, Barbara E. Moo)
5. E Balagurusamy object oriented programming with C++
6. Effective Modern C++ : 42 Specific ways to Improve Your Use of C++11 and C++14(1st Edition) by Scott Meyers