Lab 12 - Secure Programming

(and Command Line Arguments)

Due: end of class

OVERVIEW

For this lab, you will learn some of the basics in secure and bug free programming, though this will only scratch the surface. You will also learn how to pass command line arguments to C/C++ programs.

Task 0:

Consider the following program:

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

const int MAXSIZE = 16;
static int values[MAXSIZE];
static int i;  // bad idea to call a global variable i!
void initArray(int[], int);
void printArray(int[]);

// This initializes element arr[i] to val for each array element
void initArray(int arr[], int val)
{
  for (i=MAXSIZE; i >= 0; i--)
    arr[i] = val;
  return;
}

// This prints the contents of the argument array, with each element printed as
"index: value" on its own line
// For example, a 4-element array containing {10,11,12,13} would print as:
//   0: 10
//   1: 11
//   2: 12
//   3: 13
void printArray(int arr[])
{
  ...
};

int main()
{
  int dummy;
  initArray(values,5);
  int *arr2 = values;
  values[0]=9;
  arr2[1]=8;
  cout << "values is:" << endl;
  printArray(values);
  cout << endl << "arr2 is:" << endl;
  printArray(arr2);
```

```
  return 0;
};
```

As you can see, the above program should initialize each element of the array `values` to 5, update the 0th element to 9 and then print the entire array out. The `arr2` variable should be another array with the same contents, except without the update of the 0th element and an additional update of the 1st element.

After understanding the above code, finish writing printArray. Make sure to align columns - see here for a good short description, or here for more details (part of this lab is to learn how to read documentation!).

Run the program and you will get unexpected results. Understand why you get these results and explain in comments (be precise - your grade is based largely on your explanation). If you don't understand what is happening, use ddd to trace through the program while displaying the appropriate variables.

Then, fix the code (with a minimal number of changes) to do as described above and run the program.

More information (read this after you finish the above, as it won't make sense otherwise): One problem above resulted from a buffer overflow error. This is a common security error since it allows malicious code to access memory locations that they should not have access to. That memory location might have a function parameter, or even machine code for the program that can now be overwritten with malicious code (if the OS allows)!

Task 1:

It is often useful to pass command line arguments to your C/C++ program. For example, you could then type "lab12 17" to pass the value 17 to call main with an argument of 17. You've already used this in program 2 where you could provide a seed argument to the supplied code. Now you will learn how to do this in this task.

The main function actually has the following [overloaded] prototype:

```
int main(int argc, char * argv[]);
int main();
```

The argument argc represents the number of command line arguments plus 1 (the 0'th argument is the name of the program). The argument argv is a pointer to an array of arguments[1], where each element of the array is a C-string[2]. Then, main can be written as before using argc and argv as pass-by-value formals.

Modify the program from the previous task to accept 2 (and no more) arguments, corresponding to the initialized values (5 above) and the index of the updated element (0 above). For example, you would be typing

```
lab12 5 0
```

to get the same effect as above. There is one complication: the arguments have type C-string, while you desire ints for this program. Fortunately, there is a library function in cstdlib called atoi (for ascii to int) that does the conversion. For example, atoi(argv[1]) would evaluate to an int having the value of the appropriate command line argument[3].

Footnotes:

1. Since an array is itself a pointer, its a pointer to a pointer (i.e., a char **), though you typically don't need to worry about that.
2. This is the first use of C-strings in this course, as we have always been using C++ strings. The two can NOT be mixed. However, several builtin operators such as cout are overloaded to support both C and C++ strings.
3. Although this is the simplest way to do the conversion, it does not do any input validation to ensure that the user did indeed supply an integer. Better (but slightly more complicated) functions to use are the C function strtol or some functions in the C++

library sstream, though you don't need to use them in this lab.

HAND IN

Your 136 instructor will tell you what to hand in and how.

GENERAL COMMENTS FOR ALL PROGRAMS THIS SEMESTER

You should have the following header on all programs:

```
/*
  Author: <name>
  Course: {135,136}
  Instructor: <name>
  Assignment: <title, e.g., "Lab 1">

  This program does ...
*/
```

GRADING

All 135 and 136 programs this semester will be graded on:

- Correctness: Does your program work?
- Testing: Have you generated sufficient and good test data to give reasonable confidence that your program works?
- Structure: Have you structured your code to follow proper software engineering guidelines? This includes readability and maintainability.
- Documentation: How well documented is your code? Good documentation does not repeat the code in English, but explains the point of each code block, highlighting any design decisions and/or tricky implementation details.