CPSC 213: Assignment 5

Due: Monday, February 27 at 6pm.

Late assignments are accepted until Wednesday, February 29 at 6pm with a 25% penalty per day (or fraction of a day) past the due date. This rule is strictly applied and there are no exceptions.

Goal

The goal of this assignment is to examine how programs use the runtime stack to store local variables, arguments and the return address. You will begin by examining three snippets. Then you will write an SM213 assembly program based on C code, with a vulnerability from a bug. Then you will mount a buffer-overflow (stack smash) attack on that SM213 program, by creating an input string that will take over execution. Finally, you will test your understanding and assembly-reading skill by examining two SM213 assembly-language programs in the simulator to determine what they do, and explain it in English and by producing equivalent C code.

Code Snippets Used this Week

As explained in detail below, you will use the following code snippet this week. There are C, Java and SM213 Assembly versions for each snippet.

- `S7-static-call-{reg,stack}`
- `S8-locals`
- `S9-args-{regs,stack}`

Requirements

Here are the requirements for this week's assignment.

1. Carefully examine the execution of `S7-static-call-stack.s` in the simulator and compare it to `S7-static-call-regs.s`. Document what you see. Describe the difference between the two approaches. List one benefit for each approach.

2. Carefully examine the execution of `S8` in the simulator. Document what you see.

3. Carefully examine both versions of `S9` in the simulator. Document what you see. Describe the difference between the two approaches. List one benefit for each approach.

4. Write a simple SM213 assembly-language program that copies a 0-terminated array of integers (using Snippets `S8` or `S9` as a guide).

   The source array should be stored in a global variable. The destination array should be a local variable (i.e., stored on the stack). You need two procedures: one that copies the array, one that initializes the stack pointer and calls the copy procedure. Ensure that the array-copy procedure saves `r6` (the return address) on the stack in its prologue and restores it from the stack in its epilogue.

Here is a C template for the program.

```
int src[2] = {1,0};
void copy() {
  int dst[2];
  int i = 0;
  while (src[i] != 0) {
        dst[i] = src[i];
        i++;
  }
}
void main () {
  copy ();
}
```

5. Mount a buffer overflow attack on this program to get it to set the value of every register to -1 and to then halt, by changing **only** the value of `src` and the size of `src`.

   Recall that what you are doing here is what most virus writers do to exploit buffer-overflow bugs to gain control of programs (i.e., to get those programs to execute their virus code) and that a real virus writer will have the virus do something more sinister than just changing the values of registers.

   You may **NOT** directly change the program, or its stack, or any program data except for the `src` input when mounting this attack. When thinking about what machine language commands to include as part of your `src` data, you might find it easier to first create an assembly language program and then convert it to machine language, either by hand or by using the simulator to convert assembly to machine code.

6. Execute the provided SM213 assembly programs `A5-a.s` and `A5-b.s` to determine what they do. Explain their behaviour by both giving an equivalent C program and by explaining in plain English what simple computation they each perform.

Material Provided

The snippets and mystery programs are provided in the file `code.zip`.

What to Hand In

Use the **handin** program. The assignment directory is **a5**. Please hand in exactly the following files with the specified names. Do not hand in class files, or your entire Eclipse project, or a README in formats like `.doc` or `.rtf`.

1. `buffer-overflow-attack.s` that implements the copy procedure and mounts the buffer-overflow attack as specified by Requirements 4 and 5 above.
2. `A5-a.c` that gives equivalent C code for `A5-a.s` for Requirement 6.
3. `A5-b.c` that gives equivalent C code for `A5-b.s` for Requirement 6.
4. `README.txt` that contains:
   - header with your name, student number, four-digit cs-department undergraduate id (e.g., the one that's something like a0b1)
   - statement that "I have read and complied with the collaboration policies" at http://www.ugrad.cs.ubc.ca/~cs213/winter11t2/policies.html
   - Description of the difference between and benefit for the two versions of `S7` assembly snippets, as specified by Requirement 1.
   - Description of your observation during the execution of `S8`, as specified by Requirement 2.
   - Description of the difference between and benefit for the two versions of `S9` assembly snippets, as specified by Requirement 3.
   - Description of each of `A5-a.s` and `A5-b.s` (i.e. what they do) in plain English language, as specified by Requirement 6.