CPSC 213: Assignment 5

Due: Monday, October 17, 2011 at 7am.

Late assignments are accepted until Thursday, October 20 at 7am with a 20% 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 mount a buffer-overflow (stack smash) attack on an SM213 program. 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.

Code Snippets Used this Week

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

- 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 in the simulator and compare it to S7-static-call-regs. Document what you see. Describe the difference between the two approaches. List one benefit of 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 of benefit of each approach.
4. Write a simple SM213 assembly-language program that copies an 0-terminated array of integers (use Snippets 8 or 9 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. By changing only the value of `src` and the size of `src`, mount a buffer overflow attack on this program to get it to set the value of every register to -1 and to then halt. You may not change the program or its stack directly when mounting this attack. 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 value of registers.
6. Execute the SM213 programs A5-a and A5-b to determine what they do. Explain their behaviour by giving an equivalent C program and by explaining in plain English what simple computation they perform.

Material Provided

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

What to Hand In

Use the `handin` program. The assignment directory is **a5**.

1. A single file called "README.txt" that includes your name, student number, four-digit cs-department undergraduate id (e.g., the one that's something like a0b1), and all written material required by the assignment as listed below.
2. The written material requested in requirements 1-3
3. Your buffer-overflow program and the value `src` you used to mount the attack.
4. A plain-English description of your virus and how it works.
5. A written description of the key things you noted about the machine execution while A description of the mystery programs A5-a and A5-b that includes **(a) an equivalent C program** and **(b) a plain-English description**.