EECS 700 – Security and Performance
Assignment 1

The intention of this exercise is to allow you hand-on experience with viewing and deciphering architecture reference manuals, ABI documents, and small assembly files (when you have the source code as well). The assignment will further require understanding and decoding the stack layout and procedure calling conventions for a popular machine architecture.

This assignment employs the popular ARM architecture, and ARM-Linux ABI.

From Wikipedia: The ARM is a 32-bit reduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Limited. The ARM architecture is the most widely used 32-bit ISA in terms of numbers produced. The relative simplicity of ARM processors have made them made them suitable for low power applications, and led to their dominance in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers. As of 2007, about 98 percent of the more than a billion mobile phones sold each year use at least one ARM processor. Furthermore, as of 2009, ARM processors account for approximately 90% of all embedded 32-bit RISC processors.

For this assignment, you are required to perform two tasks. For the example program from the code listing *example3.c* in the paper *Smashing The Stack For Fun And Profit* (reproduced below):

```
1.   void function(int a, int b, int c) {
2.       char buffer1[5];
3.       char buffer2[10];
4.       int *ret;
5.
6.       ret = (int *)buffer1 + 12;
7.       (*ret) += 8;
8.    }
9.
10.   void main() {
11.     int x;
12.
13.     x = 0;
14.     function(1,2,3);
15.     x = 1;
16.     printf("x = %d\n",x);
17.   }
```

1. Use either *arm-linux-gcc* or the provided *vpo.exe* to compile the above program into a valid executable. Draw the stack layout when the program execution is at statement number 6. Assume that the stack grows down, as studied in class, and number all memory locations appropriately, as done in the lecture slides.

2. By looking at the stack and code layout in your executable program, appropriately modify code statements 6 and 7 to accomplish the attack mentioned in the paper. Specifically, modify lines 6 and 7 such that the program execution skips execution of line number 15 "`x = 1`", and statement 16 prints out the result "`x = 0`".

Tools made available:

arm-linux-gcc : This is a version of the standard GNU C Compiler for the ARM-Linux architecture. *arm-linux-gcc* and other binary tools for the ARM-Linux platform are installed on the server `cycle4.eecs.ku.edu`, in the directory: /usr/local/arm-linux/bin.

You may want to add the path `/usr/local/arm-linux/bin` to your PATH environment variable in ∼/.bashrc, to avoid specifying the entire path to the binary tools every time.

vpo.exe : This is a version of the VPO compiler backend for ARM-Linux platform that is used by our research group. This compiler generates some more debugging information regarding the stack layout in the assembly (.s) file, as compared to GCC, which might help you in your attempt to decipher the stack layout.

As mentioned, VPO is a compiler backend. Therefore, you will also need the provided *frontend.exe* to use with VPO, and the header files in the *ARM_headers* directory. The correct usage is:

a. To generate an intermediate-code `print.cex` file

```
./frontend.exe --c -IARM_headers print.c
```

b. Read the `print.cex` file to produce am assembly `print.s` file

```
./vpo.exe -A print.cex > print.s
```

c. You can then use GCC to produce either an .o file

```
./arm-linux-gcc -c print.s
```

OR

d. Use GCC to produce an executable file

```
./arm-linux-gcc -o print.exe print.s
```

arm-linux-objdump : A version of the standard GNU tools objdump. Usage is provided below, and also listed in the class notes.

```
arm-linux-objdump -D print.o > print.dump
```

sim-uop : This is an executable copy of the SimpleScalar simulator for the ARM. Note that it is not possible to directly execute ARM executables on the available x86 machines in the EECS computer labs. Thus, since native execution is not possible, we will simulate the execution using SimpleScalar:

```
./sim-uop -redir:sim print.sim print.exe
```

The output will be printed to the screen.

The assignment will be due before class on: September 25, 2009.

Additional Notes:

1. Each instruction on the ARM is 4 bytes long.
2. All executable files are generated for the x86-32-linux platform.
3. It is not compulsory to use the VPO compiler. You may just use arm-linux-gcc for all your compilation needs. VPO is mainly provided to further assist your debugging efforts.