Homework 4
CS 3410 Fall 2008 **Due Friday, November 14, 2008 at 11:59 p.m.**

Overview

The goal of this homework is to get intimately familiar with the layout and use of the MIPS call stack, as well as MIPS machine language, disassembly, debugging, and reverse engineering. We will do this by writing a buffer overrun exploit.

For this homework you will "0wn" a binary program called `server` that we will provide to you. We will not be providing the sources for this program. All that we know about this program is that it reads some input and prints its input back out with a prefix of "`The input was:`". So if the input is `hello`, the output is "`The input was: hello`". (Note the `$` below represents the shell prompt.)

```
$ echo hello | hw4simulator server
NetID: kb97
The input was: hello
```

The rumor among elite hackers (31337 h4x0rs) is that `server` suffers from a buffer overrun vulnerability. Since the program takes only one input, it's not difficult to guess where the problem lies.

0wning `server`

The object of the homework is for you to craft some input to the server that will make the server print out "`0wnt! h4x0r3d by <netid>`" (substitute your own NetID for `<netid>`) on the terminal, without the "`The input was:`" prefix. The fact that the prefix is missing will constitute proof that you have completely taken over the server and have gotten it to do something that it could not do before:

```
$ cat exploit | hw4simulator server
NetID: kb97
0wnt! h4x0r3d by kb97
```

To do this, you will need to inject new code into the `server` binary. You are not allowed to regenerate, replace or modify the `server` binary on disk. The only way you get to interact with the server is to feed it some carefully crafted inputs. You can run `server` by invoking the command line "`hw4simulator server`". (Note that all files referenced here reside in `/courses/cs3410/hw4`, or in the `.tar.gz` file which you can download from CMS.)

To figure out how to attack server, you'll need to step through its code as it is executing and reverse engineer the parts that matter, namely, where (i.e., at which memory location) the buffer is, what the values are that lie around the buffer, and what precise instruction sequence is vulnerable to a buffer overflow attack. You can examine the step-by-step operation of the server program by passing the argument "`-d`" to the simulator, which will cause every instruction to be dissassembled prior to execution. You can examine the contents of the top N words of the stack by passing the argument "`-s` N" to the simulator, which will dump the contents of the stack at each instruction.

Note that the simulator initializes the stack to an address that is based on your NetID. This introduces a bit of variability from one student to the next. If you are using the CSUG machines, you don't need to do anything special. If you're using another machine, however, and your username does not match your NetID, then it is *very* important that you set the environment variable `NETID` before running the simulator. (See "Setting Up Your Linux Environment" below.)

Once you have figured out the stack layout, you need to come up with the carefully crafted input that will take over the server. This input will likely contain some binary (the vector that you inject) that corresponds to MIPS instructions. You need to figure out how to translate MIPS assembler into such binary code (Hint: there are at least three ways; the simplest one involves a book or PDF and some manual labor).

Once your exploit has printed "`0wnt! h4x0r3d by <netid>`" to the terminal, it can just crash, loop forever, or exit gracefully. We will not penalize any of these three approaches, though looping forever is trivial to do, and you can figure out how to exit gracefully by disassembling another program that does so.

Setting Up Your Linux Environment

Setting your `PATH`

- If you are using `tcsh`:
  `setenv PATH ${PATH}:/courses/cs3410/mipsel-linux/bin`

  To avoid having to re-set your `PATH` every time you log in:

  ```
  cd
  chmod u+w .cshrc
  echo 'setenv PATH ${PATH}:/courses/cs3410/mipsel-linux/bin' >> .cshrc
  ```

- If you are using `bash`:
  `export PATH=${PATH}:/courses/cs3410/mipsel-linux/bin`

  To avoid having to reset your `PATH` every time you log in:

  ```
  cd
  chmod u+w .bashrc
  echo 'export PATH=${PATH}:/courses/cs3410/mipsel-linux/bin' >> .bashrc
  ```

Setting your `NETID`

This is only needed if you are using a non-CSUG machine.

- If you are using `tcsh`:
  `setenv NETID <netid>`
- If you are using `bash`:
  `export NETID=<netid>`

Copying the HW4 files

You may find it useful to copy the HW4 files into your home directory:

```
$ cp -R /courses/cs3410/hw4 ~
$ cd ~/hw4
```

Tools

To help make your life as a h4x0r easier, we have provided you with some tools:

- You can obtain a listing of the assembly code of the server by running:

  ```
  mipsel-linux-objdump -xdl server
  ```

(The `mipsel-linux-objdump` program is found in `/courses/cs3410/mipsel-linux/bin`, which should be on your `PATH` if you followed the previous section on "Setting Up Your Linux Environment".)

- Since it is not possible to generate every input combination on the keyboard, we have provided you with a program called `helper`. `helper`'s sole function is to convert a set of binary values you specify, in a separate file you pass as an argument to `helper`, into a string and print it out. If you invoke "`helper test-input.txt`", it prints out "`Far above Cayuga's waters...`":

```
$ helper test-input.txt
Far above Cayuga's waters...
```

  Recall that in Linux (and Windows), the output of a program (say `helper`) can be connected to the input of another program (say `server`), using the "`|`" (pipe) operator. So if you execute the command line "`helper test-input.txt | hw4simulator server`", you see "`The input was: Far above Cayuga's waters...`":

```
$ helper test-input.txt | hw4simulator server
NetID: kb97
The input was: Far above Cayuga's waters...
```

What to Submit

Submit your exploit (the text file you provide as an input to "`helper`", containing the specially crafted input). We'll run it against our server and check if you can 0wn it. Remember to comment your exploit so that you'll receive partial credit if it doesn't work.

Epilogue

We're here to help. Take advantage of our office hours if you are stuck.

For an entertaining read on buffer overflow attacks, check out:

> Aleph One. Smashing the Stack for Fun and Profit. *Phrack Magazine,* 7(49), November 1996. `http://www.phrack.org/issues.html?issue=49&id=14`

Hacking skills (skillz) are priceless, as they reflect a deep understanding of the operation of a computer system. But use them wisely. Taking over someone else's machine for real, or worse, 0wning the Internet not only carries stiff penalties, interferes with other people's lives and is a waste of your talent, but is also plain wrong.