Project 1

CS161 Computer Security, Fall 2008
Assigned 10/01/08
Due 10/15/08

If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle. - Sun Tzu

In this project, you will play the attacker's role. We will give you a program that is vulnerable, and you will create the exploit for it!

1 Problem Statement

Your goal is to create an exploit for a buffer overflow vulnerability. To get started, read over Aleph One's "Smashing the Stack for Fun and Profit".[1] (You don't have to read the section "Shell Code", since we provide you with shellcode, though you may find it interesting.)

The vulnerable program is `/home/ff/cs161/proj1-fa08/targets/target` on the instructional machines. Copy the directory `/home/ff/cs161/proj1-fa08/exploits` to your working space; it contains skeleton code and a Makefile for your exploit program. The source code for the vulnerable program, `/home/ff/cs161/proj1-fa08/targets/target.c`, is also provided for your reference.

Your task is to edit `exploit.c` so that it exploits the buffer overflow vulnerability in `target` to run a shell. We provide exploit code in `shellcode.h`; you just have to cause it to be executed in `target`. If you are successful, you should see a '$' shell prompt:

```
    bash-3.1$ ./exploit
    $
```

The only file you should edit is `exploit.c`. Build it with `gmake`. The path to `target` is hard-coded in `exploit.c`; please do not change it.

Because buffer overflow exploits are highly machine-dependent, you are restricted to working on **sphere.cs**, **rhombus.cs**, or **pentagon.cs**. Your exploit must work on one of those machines (they are Solaris x86 boxes).

To start with, we recommend that you use `gdb` to explore the stack and memory layout of `target`. It will be different when called via `execve()`, so here is the best way to get set up (after running `gdb ./exploit`):

```
(gdb) run
Starting program: ./exploit
```

[1] `http://reactor-core.org/stack-smashing.html`

```
Program received signal SIGTRAP, Trace/breakpoint trap.
0xce7cb7b6 in ?? ()
(gdb) symbol-file /home/ff/cs161/proj1-fa08/targets/target
Load new symbol table from "/home/ff/cs161/proj1-fa08/targets/target"? (y or n) y
Reading symbols from /home/ff/cs161/proj1-fa08/targets/target...done.
warning: rw_common (): unable to read at addr 0xce7a2060
warning: sol_thread_new_objfile: td_ta_new: Debugger service failed
(gdb) break main
Breakpoint 1 at 0x8050834: file target.c, line 21.
(gdb) continue
Continuing.

Breakpoint 1, main (argc=2, argv=0x8047f14) at target.c:21
21        if (argc != 2)

(gdb)
```

Running it this way makes it difficult to restart, however, so you may want to just run `gdb target` to explore initially and then switch to the `execve()` version when it's time to find the actual addresses for your exploit.

You will want to become familiar with the following `gdb` commands (use the 'help' command): `break`, `where`, `disassemble`, `next`, `nexti`, `x`, and `info`. Be sure to explore the display options for the `x` command.

You should not follow Aleph One's directions too closely. You may or may not want to execute the shellcode on the stack, and you can use `gdb` to figure out the exact address to jump to, so you don't have to use anything like `get_sp()` or NOP padding.

You must submit your code electronically. Go to the directory where `exploit.c` resides and type `submit proj1`. You should only submit exploit.c; you should not change the other files.

2 Grading

1. (1 point) *You must ensure that your code runs on one of the three servers listed above.* Please tell us which server you ran your code on (sphere, rhombus, or pentagon), as comment (`/* Server :  rhombus.cs/sphere.cs ...  */`) in the beginning of your submission file `exploit.c`. The grader will check out your submitted `exploit.c` and compile it. The grader will type `gmake` and then `./exploit` to run your exploit. Please also list names of your project group members as a comment in the beginning of the file `exploit.c`.

2. (7 points) When the exploit is executed by the grader, it must exploit the `target`, giving the grader a remote shell. Do not change the hard coded path for `target` in your solution.

3. (1 point) List the name of the function containing the vulnerable buffer. Draw the stack layout showing the absolute locations of the variables you had to be concerned about to make your exploit work.

4. (1 point) You should include in your homework writeup a brief description of how you arrived at your solution, including how you determined which address to jump to. The writeup for this question should be at most 4 sentences.