Exploit Lab

10% of lab grade, 5% of overall grade
out: 11/7 in class; in: 11/16 at midnight

In this lab, you will write two buffer overflow exploits. The purpose of this is to give you a good, low-level feel for how procedure calls and stack discipline work. However, the skills you will learn here can also be put to use for evil purposes. Please note that actually attacking or breaking into a computer system is a criminal offense punishable under federal law. You may work in groups of up to two people.

Getting the code

On a TLAB machine, copy the file /home1/pdinda/HANDOUT/exploitlab.tar to your working directory and then untar it with tar xvf exploitlab.tar. You will find the following files:

- Makefile
- attackee.orig
- attacker.c
- attackee.c
- exploitlab.pdf

Running "make" will build four things: attacker, attackee, attacker.s, and attackee.s.

Description

Attackee contains the code that you will be attacking. It reads up to 256 bytes from standard in into a buffer which is only 125 bytes long. Unlike the book's example and the example shown in class, it does not use gets(), but rather it calls read() directly. This is intended to make it easier for you to create your exploit code. Unlike gets(), read() will not stop reading on a newline character, and thus you don't have to worry about avoiding newlines in your exploit code.

Attacker takes two arguments – an offset (how much filler space to write before your exploit), and the number (0 or 1) of the exploit code that should be written. An attack will look like this:

```
unix> attacker 35 0 | attackee
```

That means "write 35 zero bytes, then write my exploit 0".

Exploit 0

In exploit zero, your goal is to get attackee to run its "cracked()" function, which will print the string "Drat! I'm cracked." Notice that all the code that you need to run is already in attackee – you just have to figure out how to get attackee to jump to it.

Exploit 1

In exploit one, your goal is to get attackee to print "Hi!" In this case, you will need to send code to attackee and then get attackee to jump to the code that you sent. It is possible to do exploit 1 without writing any assembly code yourself, but you may do so if you wish.

What you can do

You may write attacker in any way you choose. You may modify attackee in any way you choose, but your code will ultimately be graded using attackee.orig. You may build and run attacker and attackee in any way you choose and anywhere you choose, but grading will ultimately happen on a tlab machine using /usr/local/bin/gcc.

Advice

We strongly suggest that you start by working on exploit zero. Exploit one will build on what you learn there. You should also reread the description of buffer overflow exploits from the book.

For exploit zero, you basically want core() to return to cracked() instead of to main(). Your attacker should generate a write that goes beyond the end of the buf array and overwrites the return address on attackee's stack with the address of cracked().

For exploit one, you want to place your exploit code on attackee's stack and then overwrite the return address so that core() jumps into that code. This means you need to know the address at which your exploit code will begin in attackee's stack. The exploit code itself can be a function in attacker. You can read the machine code of the function in C:

```
  void exploit1_exploit() {
    // code
  }
  void exploit1_code(…) {
    memcpy(output_buffer,exploit1_exploit,EXPLOIT1_LEN);
  }
```

To figure out EXPLOIT_LEN, you will need to examine attacker.s.

You can read registers in gdb using syntax like "p/x $esp" and "p/x $ebp". You can read words on the stack using syntax like "p/x *((int*)($ebp+4))". You can have a continuously updating display of variables using syntax like "disp /x *((int*)($ebp+4))".

You may find it helpful to save your exploits in a file and then use that file to driver attackee running in gdb:

```
unix> ./attacker 34 0 > exploit.out
unix> gdb attackee
(gdb) break core
(gdb) run < exploit.out
```

Hand in

To hand in the lab, rename your attacker.c to attacker-uid1-uid2.c, chmod +r it, and copy it to /home1/pdinda/HANDIN before the due date/time. In addition, hand in a file named attacker-uid1-uid2.README which describes the appropriate arguments to your attacker to create a successful exploit and gives other details you think we should know.