Assignment 5

25 pts

This is a written assignment made up of one problem. Your solution is due on October 31, 10:00 P.M. PDT. You may work with *one* other person in the class on this assignment. See Section 4 for additional information on submitting your solution. You and your partner may *not* discuss your solution with other students until seven days after the assignment deadline. You may consult any online references you wish. If you use any text in your answer that you did not write yourself, you *must* document that fact. Failure to do so will be considered a violation of the academic integrity policy.

For this assignment, you will construct return-oriented programming attacks using a fixed set of instruction sequences listed in Section 1. This is not a programming assignment; you will describe your attack on paper (i.e., in a text file) only. You must use the instruction sequences provided in Section 1 only.

1 Instruction Sequences

The following is the set of instruction sequences you may use for this assignment. The instructions are given using AT&T syntax, where the destination is the second argument.[1] You will refer to each sequence using the addresses `A1`,...,`A6`, shown for each sequence.

```
A1:  pop  %ecx
     pop  %ebx
     ret

A2:  xor  %eax, %eax
     ret

A3:  mov  %ecx, %eax
     pop  %edx
     pop  %ebp
     ret

A4:  mov  %eax, 12(%ebx)
     ret

A5:  neg  %eax
     ret

A6:  int  0x80
```

2 Solution Format

Your solution to each problem will be a listing of the stack contents after overflowing a stack buffer in a vulnerable function of a 32-bit Linux program, similar to the in-class return-oriented programming worksheet. Assume that the buffer overflow involves `strcpy`, so your return-oriented shellcode *must not contain zero bytes*. (You can assume the addresses `A1`,...,`A6` do not contain zero bytes.) List the 4-byte values, one per line, as they would appear on the stack after the buffer overflow, immediately before the `ret` instruction is executed in the vulnerable function. Assume that the stack pointer has the value `0xBFFF8000` and points to the return address (saved `%eip`) that you have just overwritten. Your listing should be in bottom-to-top order, with the last element being the top of the stack, which will always be at address `%esp` = `0xBFFF8000`. For example, the following stack contents will result in the CPU storing the value `0x0` at address `0x12345678`.

```
0xBFFF8010:  A4
0xBFFF800C:  0x1234566C
0xBFFF8008:  0xFFFFFFFF
0xBFFF8004:  A1
0xBFFF8000:  A2
```

The grey addresses are shown for convenience only; do not include them in your solution. Note that `0x1234566C` + `12` = `0x12345678`. The `0xFFFFFFFF` can be any value not containing zero bytes.

[1] See https://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax for additional details.

3 Problem 1: Root Shell

The goal of this attack is to launch a root shell in the same manner as AlephOne. Your return-oriented "program" should be equivalent to the code:

```
char *name[2];
name[0] = "/bin/sh";
name[1] = NULL;
execve(name[0], name, NULL);
```

3.1 Linux System Calls

Your hypothetical target is a 32-bit Linux system. You will need to make the `execve` system call by transfering control to instruction sequence `A6`, the instruction sequence for a system call. Note that for system calls on Linux, arguments are passed in registers, rather than on the stack. Register `%eax` contains the system call number (`0xDD` for `execve`), register `%ebx` the second argument, `%ecx` the third, and `%edx` the fourth. Your solution will need to arrange to have the right values in these registers.

4 Submitting Your Solution

4.1 Solution Format

Your solution to this assignment must be a plain text file named "`{PID}-hw5.txt`" (where `{PID}` is your PID) containing your answers. The contents of the file must follow a specific format:

1. The first line of the file must be your name, last name first, with a comma between last name and first name.
2. The second line of the file must be your student id number.
3. The third line must be "`Assignment 5`".
4. If you worked with another student, the fourth line must be "`Worked with`" followed by a space and the name of your partner, last name first, with a comma between last name and first name. If you worked alone, the fourth line must be "`Worked alone`".
5. The fifth line must be blank.
6. Your solution to each problem must start with "`Problem` X" (where X is the problem number), with a blank line before and after "`Problem` X".

Figure 1 below shows an example of this format.

4.2 Encryption and Signature

Your solution must by submitted via email to `cs127f1@ieng6.ucsd.edu` by October 31, 2017, 10:00 P.M. PDT. Both members in a group must submit a solution; however, your answers to each problem may be the same as your partner's. The text file (if submit text only) or archive file (if including figures) must be signed with your PGP key and encrypted to the `cs127f1@ieng6.ucsd.edu` PGP key, which is provided on the CSE 127 Web page and has key fingerprint:

```
E1BF 1E04 1104 28DA 4F89 6543 B033 B3DC 10D3 7DBD.
```

You must send a plain email with an encrypted and signed file as an attachment. The email must have the subject "Homework 5 Submission" and the attachment must be named "`{PID}-hw5.txt.asc`" (where `{PID}` is your PID).

To sign and encrypt your submission with GPG, you can use the following command:

```
gpg --encrypt --sign --armor -r cs127f1@ieng6.ucsd.edu {PID}-hw5.txt
```

```
Foster, Ian
A00000000
Assignment 5
Worked with Maskiewicz, Jacob

Problem 1

A2
0x12345666
0xFFFFFFFF
A1
A4

...
```

Figure 1: Example solution file format. Note blank lines before and after `Problem` X.

This will produce a file named "{PID}-hw5.txt.asc" in the same directory. You will need to have imported the cs127f1@ieng6.ucsd.edu public key into your GPG keyring first.