15213 - Recitation 5 - Attack Lab

September 23, 2019

Introduction

In these activities you will review the stack, function frames, and assembly. This serves as practice for Attack Lab. A simplified version of these activity instructions can be found in the recitations slides, but we recommend you go through all of the instructions in this handout for better understanding. Before each activity, you should take a few moments to look at each activities .c file to understand the input to clobber (the meaning of the input will be explained during the activity). The assembly for clobber is also available in support.h.

To get started, login to a shark machine and execute:

```
$ wget http://www.cs.cmu.edu/~213/activities/attacklab_activity.tar
$ tar xf attacklab_activity.tar
$ cd attacklab_activity
$ make
```

Things to Remember

Little Endianness

Remember that when bytes are stored using "Little Endian" ordering, the least significant byte is stored at the lowest address, and the most significant byte is stored at the largest address. This will be particularly important when injecting assembly code onto the stack in Activity 2 and Attack Lab.
Refer to today's recitation slides or the "Bits, Bytes, & Integers II" lecture slides for more review.

The Stack

Remember that the stack grows down (so the "top" of the stack is at the lowest address) and that assembly instructions are read by increasing address. Also remember that when a function is called, the caller pushes the return address onto the stack. However, the called function often immediately decreases the stack pointer.

Activity 1

This activity focuses on how the stack can be modified to replace the return address on the previous function frame to an arbitrary return address.

```
$ gdb act1
(gdb) break clobber
(gdb) run
(gdb) x $rsp    //When calling a function, the return address is on the top of the stack
(gdb) backtrace
```

Q. Does the value at the top of the stack match any frame?

```
(gdb) x /2gx $rdi    //This treats $rdi as an array containing 2, 8-byte words
(gdb) stepi    //Repeat until you see:
                clobber () at support.s:16
                16              ret
(gdb) x $rsp
```

Q. Does this return address look familiar?

```
(gdb) disas return_address
```

Q. When clobber returns, where will it return to?
Q. Note that there is no *callq* or *retq* here. When will *retq* next be called?

```
(gdb) finish
```

Activity 2

This activity focuses on how assembly instructions can be written onto the stack and executed.

```
$ gdb act2
(gdb) break clobber
(gdb) run
```

(gdb) x $rsp

Q. What is the address of the stack and the return address?

(gdb) x /4gx $rdi

Q. What will the new return address be?

(gdb) disas *return_address*

Q. Why didn't this work? (Hint: does this address look similar to something you've seen?)

(gdb) x /5i $rdi + 8 //This will print out the next 5 instructions starting at $rdi + 8

Q. Why $rdi + 8?
Q. What are the three addresses?

(gdb) break puts
(gdb) break exit

Q. Do these addresses look familiar?
Q. What is $rdi? How do you know?

(gdb) break *0x401378 //This is *retq* in clobber
(gdb) continue
(gdb) stepi //You should see "0x00007fffffffdd40 in ?? ()"
(gdb) disas //You are not in a defined function...
(gdb) x /5i $rip //...but your instructions are here
(gdb) stepi

Q. Do you know what $rdi is? (Remember that $rsi is a pointer to *puts*)

(gdb) disas $rdi //Is it a pointer to instructions?
(gdb) x /x $rdi //Is it a pointer to a hex value?

(gdb) x /d $rdi //Is it a pointer to an integer?
(gdb) x /s $rdi //Is it a pointer to a string?
(gdb) c
(gdb) c
(gdb) c

Activity 3

This activity focuses on return-oriented programming, by which you can use piece together existing assembly instructions to do something new.

$ gdb act3
(gdb) break clobber
(gdb) run
(gdb) x /5gx $rdi

Q. What is the new return address?

(gdb) x /2i *return_address* //This is what will execute after clobber returns

Q. What will be *pop*ped into $rdi?
Q. Where will the program *ret*urn to next?

(gdb) x /2i *return_address*

Q. What will be *pop*ped into $rax?
Q. Where will the program *ret*urn to next?

(gdb) _____ *return_address* //Choose the right command to figure out what happens next

Q. After we return to the last return address, what function is called?
Q. What is the value of $rdi? Where is it finally used?

(gdb) **finish**