Timings

Section	Timing
Intro / Attacklab intro	5 minutes
Basic stack review	8 minutes
Activity 1: Setup and intro	5 minutes
Activity 1: Stack diagrams (students)	6 minutes
Activity 1: Review stack diagrams	4 minutes
Activity 1: Exploitation (students)	5 minutes
Procedure calling review	8 minutes
Activity 2: Exploitation (students)	7 minutes
Wrap-up	2 minutes

Learning goals

What do we want to teach specifically?

- Review push, pop, call, ret instructions
    - ret always looks at the top of the stack, just like pop
- Review stack layout, what you find on the stack
    - What is $rsp pointing to?
    - What is a stack frame?
    - What things are stored in the stack? (callee-saved registers, return addresses, saved arguments >6, local variables)
    - frames are not really "freed", stack frames w/ multiple function calls
- Practice drawing stack diagrams based on assembly
- Activities help:
    - Introduce concepts related to buffer overflow
    - Teach how to use gdb to inspect stack layout (correlate this with a stack diagram)

Most important thing for students to learn:
How to draw a stack diagram based on the C and assembly code incl. return address
How to examine the stack in GDB

Guidance

Stacks Review

- "Manipulating the stack": maybe address any confusion regarding what push and pop instructions do. (push will first decrement %rsp, then write to (%rsp). pop will read from (%rsp), then increment %rsp.) The best way to think of this: everything at the current value of %rsp and above is "in use", whereas everything at a lower address is "freed".
- For open-ended questions (which instructions can we use to change %rsp?), ask students for suggestions, and write their answers on the board. You can go over the slides afterwards to make sure you didn't miss anything.

Activity 1

- Clearly, you should make a joke about one of the variables being stored "long before" the buffer in memory.
- Try drawing the stack diagram on the board; that way, everyone can follow along. The slides can be used afterwards to review.
- Stop on the "Exploitation" slide to give students time to work.

Activity 2

- Go around to individually help students, especially if any are having trouble understanding how to convert hex codes into binary.

Activity Solutions

Find them on GitHub:
https://github.com/cmu15213s20/recitation-activities/tree/attacklab-s20/rec5/inputs

Note that the solution to activity 1 doesn't require hex2raw, simply the string 'aaaaaaaabbbbbbbbcccccccc15213' will suffice.

Activities

Learning objectives of the first activity:

- learn about the stack layout: local variables, return address
- learn that buffer overflows can overwrite different parts of the stack
- learn that buffers may have extra space allocated on the stack for padding
- learn how to correlate GDB output with stack diagrams

First, briefly review the C code in act1.c with everyone. Make sure they understand what the Gets() function does. Also make sure they understand that in a program with no buffer overflows, neither of the if statements should be able to succeed. (If they ask, "volatile" tells the compiler that the value can change at any time, which prevents it from assuming that before is always 0.)

Now, give students some time to draw a stack diagram in the solve() function, based on the assembly code (they'll need to open gdb and use "disas solve"). How large is the stack frame? Where are before, buf, and after located within the stack frame?

Reconvene to draw the stack frame on the board, to make sure everyone has it right. Once this is done, ask everyone to give a show of hands, and predict which variable will be overwritten when buf is overflowed: before or after?

Now, you can give them some time to try performing the exploit on their own by typing in a string, so that win(0x15213) is called. Also let them try to get the second cookie, if they can.

Learning objectives of the second activity:

- Learn how to use hex2raw to input binary data
- Learn that retq transfers control to the return address saved on the stack
- Learn that it is possible to overwrite the saved return address to return to any address within a program, not just to the start of a function.

Reconvene and walk through the first and second activities with the students, going over any parts that they missed or didn't understand. You can write solutions to the activities on the board if that's useful.

Learning objectives of the third activity:

- Learn that ROP gadgets can help load values from the stack into registers
- Reason through the changes to the stack pointer made as a ROP gadget executes
- Learn that the "retq" instruction at the end of a gadget transfers control to the next item on the stack

Probably we need to tell them about the existence of the "gadget1" function.

Also at some point help students understand defenses against buffer overflow attacks:

	Stack ASLR	Code ASLR (PIE)	Non-executable stack	Stack canary	Control flow integrity
Stack buffer overflow	no	no	no	yes	no
Return-oriented programming	no	yes	no	n/a	yes
Code injection / execution	yes	no	yes	n/a	yes