IC220: Computer Architecture Name:_____

Partner(s):_____

Buffer Overflows and the Stack

A. Introduction

In IC221, you recently learned about some attacks on system programs. In this activity, we focus on one of those attacks:

> *Overflow Attacks: Where the attack overflows a buffer to alter program state.*

Previously, you saw the harmful effects when the contents of one buffer can overflow into another, *different* buffer. What happens, though, if a buffer overflow actually causes changes to the *program stack?* Since you have already studied the MIPS program stack in great detail, you are prepared to investigate!

Scenario: You are an attacker and have managed to steal the source code for an ATM control program. You can't modify the code, but by studying the code can you find a stack-related weakness that allows you to take control of the ATM?

B. Review of the Program Stack

Recall that, when a function begins executing, it typically allocates new space on the stack using a command like this:

```
addi $sp, $sp, -12
```

That command would save space for three items on the stack (each of size 4 bytes). Recall also that the stack grows downward (e.g. towards lower addresses) – thus the shift of -12 instead of +12 in the stack pointer.

The stack is potentially used to "save" four types of information:

1. Argument registers (e.g., $a0, $a1, etc.)
2. The return address ($ra)
3. "Saved" registers ($s0, $s1, etc.) – if the function wants to modify them.
4. Local arrays or structures – note this might include a buffer!

Consider #4: suppose a user manages to overflow a buffer of characters that is stored on the stack. This might possibly overwrite the contents of any of those 4 things in the list above. Any such change could cause problems, but changing which kind of thing (from the list above) is most likely to cause major problems for the executing program? → Answer the question by circling 1, 2, 3, or 4 above.

C. Getting Started with the Target Program

1. Ask your instructor if you have any "group number".

   → Enter group number:_____

2. Follow directions from your instructor to download the source code to the ATM program.
   Be sure to get the right source code for your group!
   Remember, you may not modify this program.
3. Open qtSpim.
4. Select File->Reinitialize and Load File, then open the file you saved above.
5. Run the program (use F5) and enter "secret43" when prompted for a password. Did that work?
6. Run the program a few more times (you must use "Reinitialize and Load File" each time). Try a few different passwords.
7. You probably can't guess the password, but you should still be able to crash the ATM/program without too much difficulty – keep trying until you can!
8. → What is a password that crashes the program?_____

D. Studying the Target Program

1. Open the target program in a text editor (or refer to a printed copy if available).
2. Look first at main().
   a. What function does main() first call? →_____
   b. What function does main() ultimately call if the password is correct?

      →_____ (not just a label, but a proper function)

      Remember this function – your goal is to trick the program into executing it!
3. Look back at 2(b) – double-check that what you wrote down is definitely a function and not something else – this will be important later!
4. Now look at the functions below main(). What function actually reads the password from the user? →_____

   (Hint: the password is read as a string. What syscall reads a string into an array/buffer?)
   (Hint: this is NOT necessarily the same function that prompts the user – we want to know where a character string is actually read, via a syscall, into the program)
5. Examine the function that you just found (in Question 3). If you could crash the program by simply entering a password, then quite likely there is a bug or flaw in that function!
   This may take a little time, but review the function with your partner to find the flaw.
   Talk to your instructor if you need help.

   What is the bug/flaw? →_____

E. In-depth Recon on the Target Program

The flaw that you discovered allows an attacker to possibly overwrite the contents of a return address that was stored on the stack. Big problem! When the program later reloads the return address and then uses it (`jr $ra`), then the program "returns" to the wrong return address! Let's watch how this works:

1. In qtSpim, select "Data Segment" from the menu up top. Remove the check from "User Data", but ensure that the check is present for "User Stack."
2. Restart the program (Reinitialize and Load File), but **do NOT yet run**.
3. Find address `[0040009c]` in the program – this is inside the readAndCheckPassword() function, just before actually reading the password string from the user.
4. Verify that the instruction at this address is "syscall" – if not, see instructor.
5. Right click on that address, and select "Set breakpoint." A small "hand" icon should appear on that line. [NOTE: if you reset the program, you'll have to add this breakpoint again]
6. Run the program (F5). You may see (in the console) that it has printed the password "prompt" string, but do NOT enter a password yet.
7. When the "Breakpoint" dialog window appears, **select "Abort".**
8. In qtSpim, click on the "Data" tab (next to the "Text" tab, near top of screen, just below the menu bar). You should see something resembling this (numbers will be different):

```
User Stack [7ffff4d4]..[80000000]
[7ffff4d4]    00422740  00000000  00000000             @ ' B . . . . . . . . .
[7ffff4e0]    00000000  00400038  00000001  7ffff5af    . . . . 8 . @ . . . . . . . .
[7ffff4f0]    00000000  7fffffe1  7fffffaf  7fffff78    . . . . . . . . . . . . x . . .
[7ffff500]    7fffff3c  7fffff0b  7ffffeee  7ffffeca    < . . . . . . . . . . . . . . .
[7ffff510]    7ffffe98  7ffffe84  7ffffe75  7ffffe68    . . . . . . . . u . . . h . . .
[7ffff520]    7ffffe5d  7ffffe39  7ffffe04  7ffffde9    ] . . . 9 . . . . . . . . . .
```

   Make sure it says "User Stack." Note the first row shown is the "top" of the stack (e.g. where the most recent data goes).

   Each row shows some contents from the stack. Examine the first row:

   - At far left (e.g., `[7ffff4d4]`) is the starting address for that row of data.
   - Next is the hex value for each word of data. For instance, `00422740` is one word (4 bytes). Key point: we have here a "Little Endian" machine. That means that for data value `00422740`, the bytes are actually stored in memory in reverse order: `40 27 42 00`.
   - Finally, on the right hand side is the ASCII representation of each byte. So the bytes `40 27 42 00` are shown as the equivalent characters **`@ ' B .`**
     (a period is used for any "non-printing" value, like 0).
   - Find an ASCII chart (hardcopy or online). Verify that ASCII hex `42` is the letter **B.**

     → What is the ASCII hex value for the letter g (small G)? _____

9. Look again at your "User Stack" in qtSpim. Within the first 2-3 rows of data you should see two return addresses (hex values starting with 004….).
   What are the two return addresses? (should both start with 004…)

   → _____ and _____ (are stored return addresses, point to code)

   At what address on the stack are they stored? (should start with 7ff...)

   → _____ and _____ (are addresses on the stack, point to storage)

   (Note: if both partners do this, the addresses may be different)
10. Before continuing, look again at where those two return addresses are on the stack. Things may change soon, so make sure you know where they are to begin with!
11. Now let's see what goes wrong when you enter a password that is too long for the buffer:
    a. Click on the "Text" tab of qtSpim (instead of "Data")
    b. Verify that you are still at PC=`[0040009c]` (a syscall instruction)
    c. Press "F10" to "step" one instruction.
       (If a window pops up about a breakpoint, select "Single Step")
    d. If the input "console" does not immediately pop up, find it in your Windows taskbar, or via selecting Window->Console (possibly twice) from the qtSpim menu at top.
    e. In the console, **slowly** enter this password: **`abcdABCD1234567`**
       (**type slowly** because the Backspace/Delete key won't work to correct errors!!)
    f. Click back on the main window for qtSpim.
    g. Click on the "Data" pane to see the "User stack" again.
    h. Can you see the password that you just entered, on the stack?
       (if not, try again or ask the instructor!)
    i. Look at Question #9 again (that found where two return addresses were being stored on the stack). One of them is not there anymore!

       → Which return address was clobbered? 004_____ (complete the address)

       → What four bytes (in hex) are there now instead? _____
       (write down in same order as shown on your screen)

       → What four ASCII characters does that correspond to? ___ ___ ___ ___
       (write down in same order as the raw bytes above)

       → Which part of the password did those characters come from? _____

F. Exploiting the Target Program

1. Look back at what you just did. You overwrote a return address with a password – in other words, with some characters that you can control!
    a. Look back at Part D, Question 2(b) – this identified your goal function. Look carefully in qtSpim. → What is the address of the start of that function? _____
    b. → What four characters does that correspond to? ___ ___ ___ ___
    c. So what complete password would cause the program to "return" to that location?
    → Enter a "candidate" working password here: _____
    (This is a key step, so please **write especially clearly!**)
    (Think carefully about the order and placement of your characters!)
2. Let's see if it works!
    a. File->Reinitialize and Load File
    b. Set breakpoint at PC=[0040009c]
    c. F5 to run, then click "Abort".
    d. F10 to step, then click "Single Step."
    e. Switch to console, enter your "candidate" password and hit return.
    f. Cick on the "Data" tab and examine the stack. Does it have the address you want (see question 1(a) above)? Is it in the right place (replacing where a return address used to be)?
    g. If not, revise yor password and go back to step 2(a).
    h. If it does, press F5 to continue.
        i. Do you get an error? If so, something is wrong.
        ii. Did you reach the goal? If so, celebrate (!) and then….
        → Enter account #:_____

G. Wrapping up

1. This program was exploitable because
    a. There were return addresses on the stack.
    b. There was a buffer on the stack.
    c. It was possible to overflow the buffer.
2. We probably can't avoid (a), but we could put the buffer somewhere else (where?). And we could try to prevent the buffer from overflowing (how?). Be prepared to discuss this.
3. How else could we subvert this program?
    (Bonus) Can you find a "valid" password – e.g. that grants access to the program, but does NOT overflow the buffer? → **_____**
    (Hint: Look at the hash function!)
    (continued on next page)

H. Assessment

(answers will not affect your grade)

After completing this activity….

Do you think this activity was useful for your learning?	YES	NO	SOMEWHAT
Do you have a better understanding of buffer overflow attacks?	YES	NO	SOMEWHAT
Do you have a better understanding of return addresses?	YES	NO	SOMEWHAT
Do you have a better understanding of how data goes on the stack?	YES	NO	SOMEWHAT

What was the most interesting part of this activity?

Any suggestions for improvement?

Any other comments?