By their nature, some programming language are limited in handling the buffer and can lead to risks. C programming has functions that do not effectively check for boundaries.

The length of the username we gave was long enough to cover all the space reserved for both username and password and overwrite the content of the stored values of $fp and $ra. As I said before the stored value of $ra contains the address of the next instruction to execute once the running function will complete the job (in our case main). When the current function will finish executing, the content of the actual register $ra will be restored with the old value stored in the stack that has been overwritten by the (way too big) given username. The register $ra is a 32 bits register, so it will contain 4 times the length of a character, in our case it will contain: AAAA or in hex values 0x41414141. Now, the program will try to execute whatever instruction is at the address 0x41414141, this can lead to unpredictable results, most of the time to a memory access violation like in our case.

Introduc

A stack b uld allow an attacker t code. A nice, simple demo of this process can be found here at Wikipedia.

For this assignment you will begin by first naively overwriting stack values to crash a program. Next, you will more intelligently overwrite the stack to break a vulnerable authentication mechanism. Finally you will trace through a susceptible MIPS function and craft a real input that overflows a buffer and returns control to a "malicious" function and then patch the vulnerability.

Assignment

Step 1: Run the Code

Begin by familiarizing yourself with the C code in overflow.c (found on Blackboard). Once you have an idea of what the code is doing I recommend using REPL or OnlineGDB to compile and run your C code. Practice running the code and make sure you can demonstrate both a failed and a successful authentication attempt by providing a correct and an incorrect password.

Step 2: Crashing the Application **A x 100**

For the first part of this project I'd like for you to provide an input (password) to overflow.c that causes a segmentation fault/core dump, essentially crashing the program. Please use OnlineGDB as it has the necessary security measures to detect your malicious attempt. **Write a detailed 1-2 paragraph explanation describing why you chose your input and how it caused a seg-fault to occur. Please also provide a screenshot showing the input you chose and evidence that a segmentation fault occurred.**

Step 3: Authentication Trick **A x 10**

For this part you'll use REPL and will be a little bit more thoughtful about the input we provide to overflow.c so we don't crash the application, but rather gain access despite using a password different than the actual secret password. Remember, we're inside of the main() function, so think about how the program is keeping track of the buffers on the stack. Derive an input, other than that true secret password, that will trick the program into authenticating a user. Note: there are many such passwords, so don't focus quite so much on the content (but do not ignore it) and think about where the input data is being stored in memory. **Again, write a detailed 1-2 paragraph explanation describing why you chose your input and how it tricked the program into authenticating the user. Please also provide a screenshot showing the input you chose and evidence that you were able to gain unauthorized access successfully.**

Step 4: Stack Buffer Overflow Attack

The final part of the assignment will require you to download overflow.s (found on Blackboard) which contains a buffer overflow vulnerability. Paste the code into MARS and make sure you can run it successfully with the input: Hello, world!

Next, please comment each line of code in the .text section. Please do this FIRST! It will make tracing through and understanding the code much easier.

Once you understand what the code is doing, you'll notice there is a 'print_a' function that is not reachable through the execution path of the code as it's written. Your job is to devise an input that overflows the stack buffer and overwrites the $ra register causing the program to execute the 'print_a' function. **Please provide the successful input that triggers the overflow, a screenshot of the successful execution of your attack that prints the A+ message, and a detailed description of how you figured out how to exploit the buffer overflow and how you devised the proper input.**

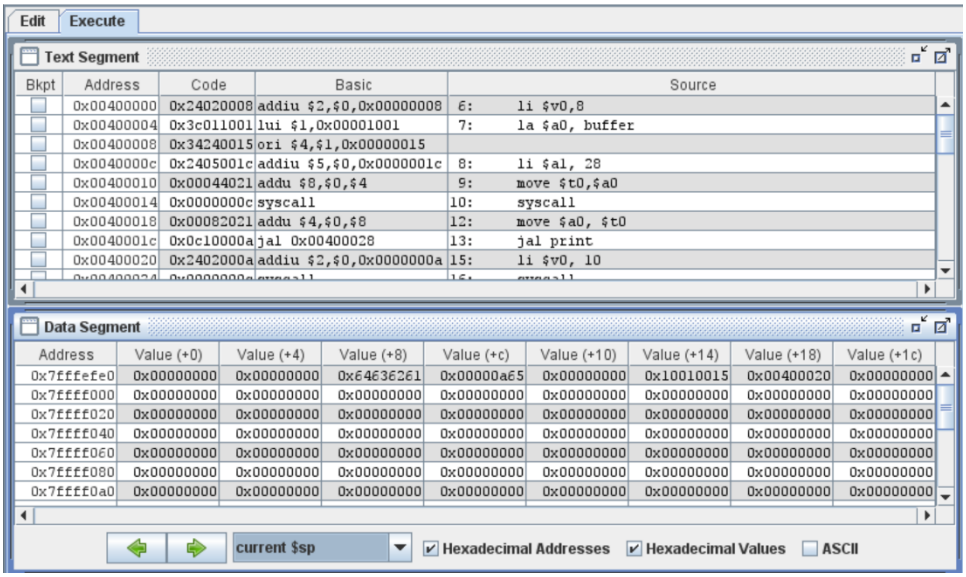
Finally, you will write a small amount of MIPS code to patch the vulnerability. Using the existing code from overflow.s, implement logic to defeat the exploit you wrote above. To keep you on track, your patch should only require around ~10 lines of code. **Please submit your patched code in a file called overflow_patch.s along with a screenshot demonstrating that your patched code successfully prevents the malicious input devised above from working.**

Deliverables

To recap what should be submitted:

1. Written explanation and screenshot from Step 2
2. Written explanation and screenshot from Step 3
3. Written explanation, including the specific malicious input, and a screenshot from Step 4
4. Patched MIPS code in overflow_patch.s

Hints:

1. Know thy stack!
2. Become familiar with asciitable.com. You will need it for a portion of your input to get the right values into the $ra register.
3. Remember your endianness.
4. Be sure to know how to examine the stack using MARS. Here is a sample for an input of abcde:
   a. 

5. Be sure to know how to examine your .text section in memory:
   a.