Andrew Niklas
CS 256

Exercise 8 Report

Problem 1: Launching a Shell

After compiling and then running shellcode1, the terminal shows only $ and had removed all the other user info from the previous prompt.

Shellcode2 contains a copy of the compiled code from the previous file (shellcode1). It then puts this code into an array that is copied to the stack and then executed as a program. After running shelcode2 we obtain the same results as we did from running shellcode1, which is opening a new terminal session inside of the current terminal.

Problem 2: A Vulnerable Program

The program, *stack.c*, causes a buffer overflow by copying the character array *str* which has a size of 517 bytes into the character array *buffer* which only has a size of 24 bytes.
The line strcpy(buffer, str) has an overflow vulnerability because it copies until the terminating null character. This causes a vulnerability if the source C-string is larger than the destination C-string.

Problem 3: The Exploit

- *memset* is used to fill the *buffer* array with NOP 517 times. NOP is basically the same as having a blank line in a program, it does nothing except tell the program to continue onto the next instruction.
- *strcpy* is used the same a previously to copy the contents of shellcode into buffer.
- *addr* is a variable used to set the new return address .
- offset is another variable is the number of bytes offset from the start of the buffer we overflow where the return address of the function is stored.
- *ptr =* is setting the location that ptr will be stored in memory(the locatioin of the return address)
- **ptr =* is setting the contents of ptr to the new return address(our program)

- The next part opens a new file called ‘badfile’ and writes the contents of *buffer* into the file and then closes the file.

Problem 4: Using the Debugger

#1 addr = 0x080484ff, offset = 10

0x080484ff because gdb listed it as in main, and when it showed up in memory near buffer I assumed it was the return address back into main. Offset because it was the 10th word listed when the memory was printed. The result was a segmentation fault.

#2 addr = 0x080484ff, offset = 5

0x080484ff because it looks similar to the address listed as in main. 5 because it was the 5th word listed. The result was a segmentation fault.

#3 addr = 0xbffff118 offset = 36

address is near the beginning of the buffer. Offset because I counted wrong before. Resulted in Trace/breakpoint trap (core dumped).

#4 addr = 0xbffff1a8 offset = 36

address is in buffer but past the original return address. Offset because the address is 36 bytes away from the start of buffer. SUCCESS!!!