CS 5435 – Security and Privacy Concepts in the Wild

Homework 4

Name: Samhitha Tarra st786, Martin Eckardt me424 *Fall 2022*

Question 3.1 Briefly explain the vulnerabilites in target1.c, target2.c, target3.c, and target4.c.

The code in target1.c is vulnerable to a buffer overflow attack. The function "foo" takes in a 12 character string from the command line argument, but the name array it is copied into is only 4 characters long, leaving the 8 extra characters to overflow into other parts of the program's memory.

The code in target2.c is vulnerable to a numerical overflow. We are essentially passing in the number 69535 in our sploit to cause an overflow. The variable input-size is expecting a short that ranges from -32768 to 32767 which is a total of 65536 allowed values. When we send in an int due to the type mismatch we are sending 65536 + 399 (the overflow) so the input size becomes 399. This causes the check on input-size to pass so that we can then overflow the buffer.

The code in target3.c also contains a buffer overflow vulnerability. The memcpy function is copying 5 ints into an array of size 4, which can potentially overwrite other data in memory. Additionally, the code does not check the size of the argument passed in, meaning it can be larger than the intended size and cause an overflow.

The code in target4.c has the potential for a buffer overflow as well. In this code, the function foo is copying the data from the command line argument passed to main (argv[1]) into a character array of size 4 (name[4]). Since the size of the character array is fixed, if the command line argument is longer than 4 characters, then there will be a buffer overflow as the extra characters will not fit in the array.

Question 3.2 Does ASLR help prevent return-into-libc attacks? If so, how? If not, why not?

Yes, ASLR (Address Space Layout Randomization) can help prevent return-into-libc attacks. It does so by randomizing the address space layout so that an attacker cannot accurately predict where the return address for a function will be. This means that an attacker cannot simply craft a malicious code injection to return into a specific location in the libc library, since that location will be different each time the application is loaded. This makes it much harder for an attacker to successfully execute a return-into-libc attack.

Question 3.3 Explain the stack protector countermeasure, which we turned off via the "-fno-stack-protector" option to gcc. Which of your exploits would be prevented

by turning on this stack protection?

The stack protector countermeasure is a type of security feature that prevents buffer overflow attacks by protecting the stack from being overrun with malicious data. It works by placing a guard variable on the stack and checking it before return from a function. If the variable has been modified, the program will terminate and alert the user that an attack has occurred.

This countermeasure would prevent our return-to-libc exploit, as the guard variable would be checked before returning to libc, and the attack would be detected and stopped. It would also prevent our buffer overflow/stack smashing exploits. By turning on the -fno-stack-protector option, the compiler will not generate extra code to detect buffer overflows. This prevents the stack smashing exploit by removing the code that would usually detect and prevent the buffer overflow. Without this extra code, the buffer overflow can occur, but it will not be detected, and thus the exploit cannot take advantage of it.