Assignment 3: Shellshock attack, Buffer overflow attacks
Gunnar Yonker

1.

(1) For the shellshock vulnerability to be exploitable, two conditions need to be satisfied. The two conditions are:

The target process should run bash.

The target process should get untrusted user inputs via environment variables.

(2) When you run the given program, it will fork and create a child process. The parent process will print “parent” and the child process will print “child”. The environment variable foo contains a function definition with the echo command. Because of the shellshock vulnerability in bash, when the function is imported from an environment variable, it will not only define the function but also execute the command following the function definition. In the given program the /bin/sh which is symlinked to /bin/bash will trigger the Shellshock vulnerability when the child process is executed. The sequence of outputs will be:

parent (from the parent process)

child (from the child process)

world (because of the Shellshock vulnerability where echo world; is executed)

Additionally, the program will execute /bin/ls which will list the contents of the current directory. This will also be part of the output after the word “world.”

2.

(1) When a function is called, it gets a stack frame on the program’s call stack. This frame stores local variables, among other things. When “func1” is called the address of x and y are determined based on their relative locations in the stack frame. “y” is an argument to the function, so it will have an address higher on the stack than the local variable “x”. The exact address is determined by the runtime environment, and it can change from one execution of the program to another, especially if ASLR is in use. Typically the compiler and the calling convention used determine the exact layout of the stack frame, and thus where in the frame each variable resides.

(2)

var1 – As a global variable, it will be in the data segment.

char *ptr = malloc(sizeof(int)) – The pointer ptr is on the stack, but the memory it points to is in the heap because of the malloc.

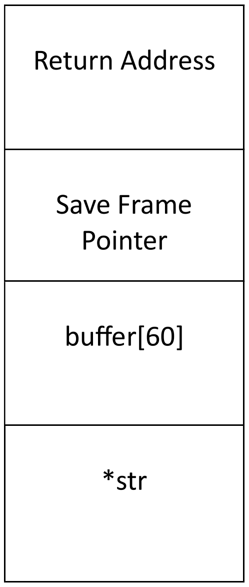
char buf[768] – This is a local variable, its located on the stack.

int var2 – A local variable, so it will be on the stack.

static int var3 – As a static local variable, it will be in the data segment, just like global variables.

Assignment 3: Shellshock attack, Buffer overflow attacks
Gunnar Yonker

(3)



(4)

It appears that the students are trying to overwrite the return address with the address where the shellcode (malicious code) resides in the buffer.

- case 1 and case 2: The return addresses 0xbffff250 and 0xbffff280 successfully point somewhere within the buffer (or NOP sled), leading to the shellcode. Hence, they get shell access. It's likely that these locations contain or lead to the attacker's shellcode.
- case 3: Address 0xbffff300 might be outside the bounds of the shellcode or NOP sled, or it could point to an invalid instruction or address within the buffer. Therefore, they cannot get shell access.
- case 4: Address 0xbffff310 is close to the address in case 3, but it might just be within the range of valid addresses that lead to the shellcode, hence shell access is achieved.
- case 5: Address 0xbffff400 is likely way outside the range of valid addresses, leading to no shell access. This could be pointing to a location well beyond the attacker's shellcode, or even beyond the current function's stack frame, leading to a crash or unexpected behavior instead of shell access.

The return addresses should lead to the shellcode, either directly or through a NOP sled, which slides the execution down to the shellcode. In conclusion, buffer overflow attacks rely on precise understanding and control of the memory layout of the target process. The observed differences between the cases likely result from the exact positioning of the shellcode, the layout of the stack, and the exact method used to overflow the buffer. Countermeasures like ASLR and non-executable stacks can help in understanding why some addresses might work while others don't.