IERG4130 Assignment 1

LAU Long Ching
SID: 1155127347

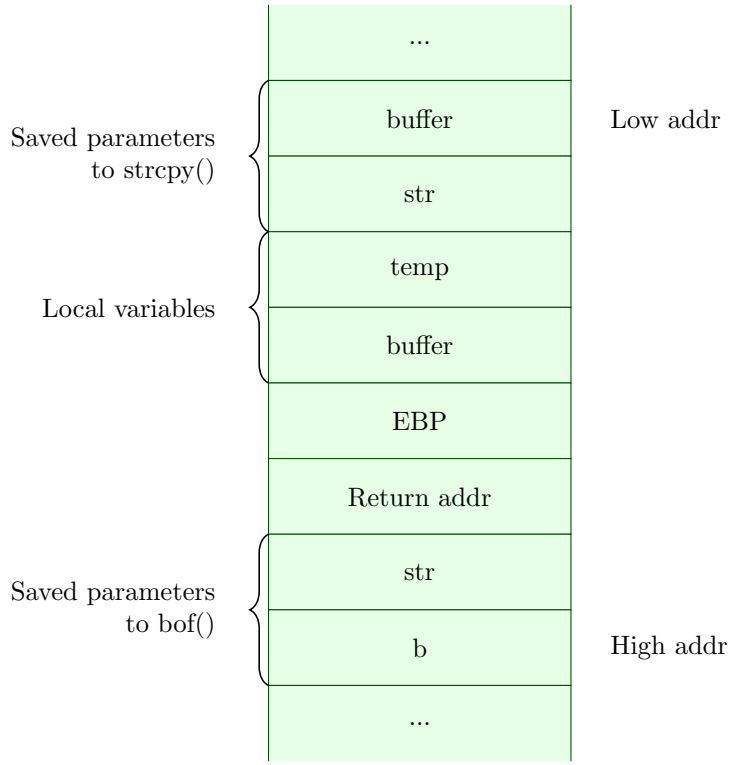
06/02/2021

Question 1

`a` will be allocated on the Data segment since it is an initialized global variable.

`k, m, buf` and `ptr` will be allocated on the stack since there are local variables, however the memory `ptr` points to will be allocated on the heap since it is dynamic memory.

Question 2



Question 3

Assume that the adversary has `data` and `len` under control, the function is unsafe. The `memcpy()` function indeed has length control but note that the definition of the data type `size_t` is an unsigned integer. Attackers can provide a negative integer for `len` (which is obviously less than 64) and it would be implicitly casted to an unsigned integer and becomes a very large positive integer. `memcpy()` then copies the potentially huge amount of memory into `buf`, and buffer overflow vulnerabiliy shows up.

Question 4

(1)

Stack buffer overflow and its variants including replacement stack frame, return-to-libc attack, heap overflow and global data overflow. There are format string overflow and interger overflow as well.

(2)

All of them are conditions at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system.

Classic stack buffer overflow occurs when the targeted buffer is located on the stack, usually as a local variable in a function's stack frame. The attacker overwrites the saved frame pointer and return address.

A replacement stack frame attack overwrites the buffer and saved frame pointer address, than change the saved frame pointer value to refer to a location near the top of the overwritten buffer and create a dummy stack frame with a return address pointing to the shellcode lower in the buffer.

During a return-to-libc attack, the return address is changed to jump to existing code on the system. The attacker replaces the return address with the address of a desired library function, and writes a fake return address for the library function. The function assume it is called, so it makes use of the fake return address in the stack.

Heap overflow works similarly as stack overflow, but the target is now heaps that contain data object created with `malloc()` or `new()`. It has the same property as buffer objects on a stack. There will not be return address here to easily cause a transfer of control, but through function pointers that are subsequently called, the attacker can modify the pointer address to a shellcode.

Global data overflow involves buffers located in the static data area. like heap overflow, if unsafe buffer operations are used, data may overflow a global buffer and change the memory locations.

Integer overflow targets at the limited range of integer objects.It occurs when the attacker attempts to cast a value that is outside of the range that can be represented with the data type either higher than the maximum or lower than the minimum representable value. Examples include casting a signed integer to an unsigned one.

Format string overflow occurs when the attacker submits an input string that is evaluated as a command by the application. In this way, the attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

(3)

No-Execution-Bit (NX) prevents the execution of injected code by implementing a default "if write then no execution" policy. However attackers can inject arguments in stack but execute existing code without NX in TEXT segment. An example is to inject a fake return address so that the system executes something in libc. This is also called the return-to-libc attack.

Question 5

`0x66AA0050-0x66AA0020=0x30=48`

The target address of my code should be in `str[48-51]`. The scope of the new value would then be `0x66AA0054`∼`0x66AA0020`+300-`sizeof(shellcode)`.

Let's assume the return address is 0x66AA00E1, the new scope is then 0x66AA00E1∼0x66AA014C-`sizeof(shellcode)`.

`str[0-47]` would be NOP sleds or random values (except `\x00`), and `str[48-51]` contains the fake return address (0x66AA00E1), and `str[52-299]` would then be NOP sleds and the malicious code.

Question 6

Changing how the stack grows indeed protects the return address of `bar()`. Normally if you passed `"overflow"` into `bar()`, it could override the return address of `bar()`. This does not happen anymore when the stack grows from low address to high address. However, a new vulnerability emerges as this time the attacker can override the return address of `strcpy()`. Buffer overflow still occurs, just that the behaviour changed a bit.