8 Cybersecurity (fms27)

For the purpose of this question, assume that you are working on a 32-bit x86 system, that `strcpy()` is the C standard library function and that all the other named functions were invented for this question.

(*a*) You discovered a buffer overflow vulnerability in a program. You wish to execute some shellcode you prepared, which is 48 bytes long. During the execution of the vulnerable function, the `ebp` is at `0xbffff3e4` and the 32-byte vulnerable buffer starts at `0xbffff3c0`. The `badstuff` you supply gets copied into the vulnerable buffer via `strcpy()`, provided it does not exceed 512 bytes in length.

You prepare the following `badstuff` input...

16 copies of `0xbffff400` ; 32 bytes of nop sled ; 48 bytes of shellcode

...but your shellcode does not get executed.

(*i*) Assuming `badstuff` has just been copied into the buffer, give hex addresses for: the start of the nop sled; the start of the shellcode; the location of the return address; the first location not overwritten by `badstuff`. [4 marks]

(*ii*) Clearly explain why your shellcode did not run. [2 marks]

(*iii*) Fix the attack by changing some of the numbers, with clear justification, but without altering the structure or length of `badstuff`. [4 marks]

(*iv*) Construct, with explanation, the shortest possible `badstuff` that still executes your shellcode payload. Give the lengths of your new `badstuff` and of the original one. [4 marks]

(*b*) The host system has been strengthened by making the stack not executable. Nonetheless, you wish to exploit a buffer overflow vulnerability you discovered in another program. Your input gets copied via `strcpy()` into a vulnerable buffer in the stack frame of `print()`, at absolute memory position `&buf`. The return address, during the execution of `print()`, is at `&ret`. With your attack, you wish to invoke `unlock(0)`, whose entry point is at `&unlock`. You may use library function `clear(p)` (entry point `&clear`), which writes 0 into the word at address `p`. You may not skip the functions' prologues. You may crash the program after exiting `unlock(0)`.

Clearly explain how to construct the input, describing the evolution of the stack throughout the attack. Your answer must provide the exact content of your input, byte by byte, as in the example provided in Part (*a*), parametric in the symbolic addresses above. [6 marks]