Buffer Overflow Attack - Exercise

* Required

1. Email address *

2. Please enter your Student ID Number *

3. Please enter your UCINetID *

4. Please enter your Full Name *

Questions:

5. The stack is populated with functions as they are invoked. How does the stack grow in the memory? * 1 point

   Mark only one oval.

   - ◯ From lower addresses to higher addresses.
   - ◯ From higher addresses to lower addresses.

6. $ebp points to the location of the instruction that follows the call instruction to the current function in the stack frame. Is this true? * 1 point

Mark only one oval.

- Yes
- No

7. The return address region in the stack frame of a function points to the base address of the stack frame of the function that previously called the current function. Is this true? * 1 point

Mark only one oval.

- Yes
- No

8. The "oldebp" region of a stack frame points to: * 1 point

Mark only one oval.

- The base address of the stack frame of the function that previously called the current function.
- The location of the instruction that follows the call instruction to the current function in the stack frame.
- Neither

9. A buffer overflow attack aiming to execute malicious code involves writing over one or more of the following regions of the stack frame from where the attack is initiated. Choose the correct region(s): * 1 point

Check all that apply.

- ☐ Return address region
- ☐ Old ebp region
- ☐ Local variables region
- ☐ None of the above

10. During a buffer overflow attack using strcpy(), you copy a character array. How is the stack populated with the array elements? * 1 point

Mark only one oval.

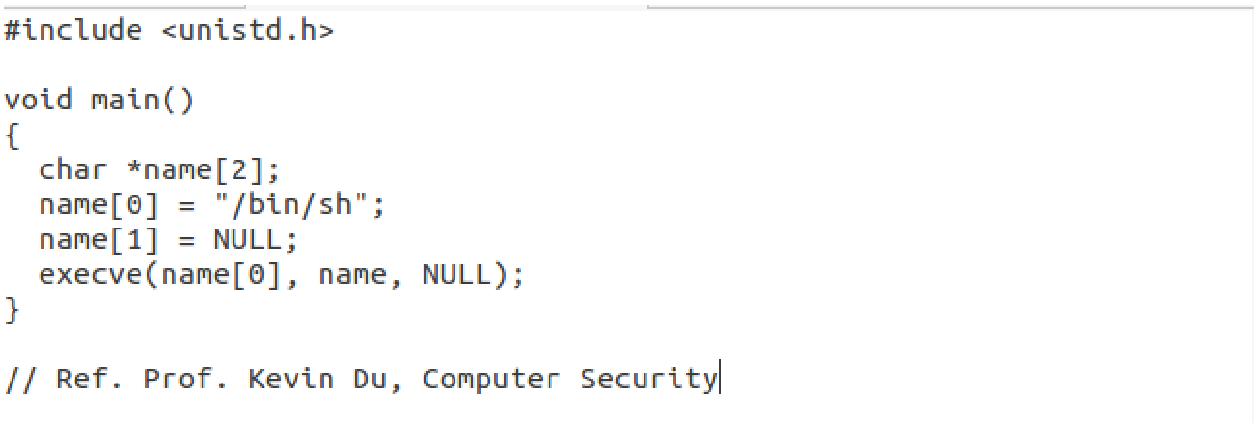
- ◯ The array content is placed in the stack starting from the lower stack addresses to the higher ones.
- ◯ The array content is placed in the stack starting from the higher stack addresses to the lower ones.

11. During a buffer overflow attack using strcpy(), we provide an address in hex to which we want the execution to jump to. By only seeing them and not knowing what they contain, which of the following addresses could we tell would not work in the attack? * 1 point

Check all that apply.

- ☐ 0x400e13ab
- ☐ 0xb7e42da0
- ☐ 0xb7e4036d
- ☐ 0x5ea00d00
- ☐ All would work.
- ☐ None would work.

12. Say we want to execute shell in the target machine using the following C code. A strategy to trigger this in a buffer overflow attack with strcpy() may be to pass the executable binary of the code in a character array (say buffer) in the vulnerable program. What may be the reason(s) why this might not be a good strategy? * 2 points



```
#include <unistd.h>

void main()
{
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}

// Ref. Prof. Kevin Du, Computer Security
```

Mark only one oval.

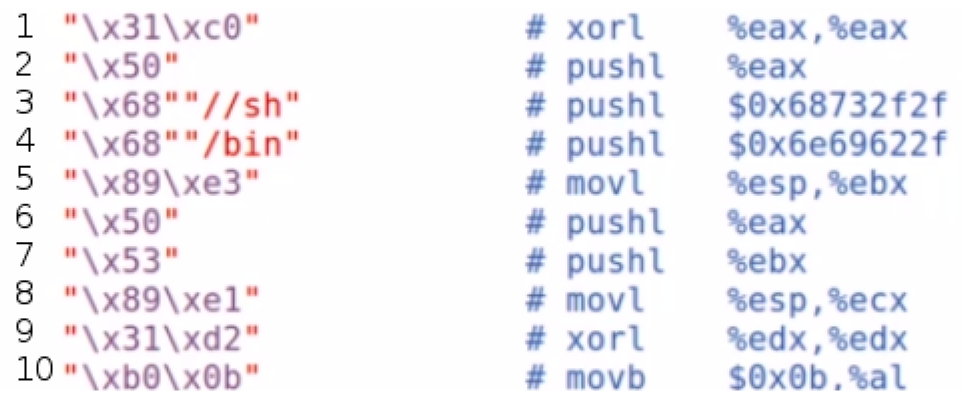
- The binary of this file may be very large.
- The opcode of this binary may contain that based on which you chose your answer in the previous question.
- Both of the above.

13. execve() is a system call taking in 3 arguments. Where does it get those arguments from? * 1 point

Mark only one oval.

- memory
- registers
- cache
- None of the above.

14. Given that injecting the binary of the C code above in the stack is not a good attack strategy, we decide to inject its machine opcode, as given below. Is this complete? (The commented code is the assembly instruction for each opcode instruction, Ref. Prof. Du, Computer Security) * 2 points



```
1  "\x31\xc0"                    # xorl    %eax,%eax
2  "\x50"                        # pushl   %eax
3  "\x68""//sh"                  # pushl   $0x68732f2f
4  "\x68""/bin"                  # pushl   $0x6e69622f
5  "\x89\xe3"                    # movl    %esp,%ebx
6  "\x50"                        # pushl   %eax
7  "\x53"                        # pushl   %ebx
8  "\x89\xe1"                    # movl    %esp,%ecx
9  "\x31\xd2"                    # xorl    %edx,%edx
10 "\xb0\x0b"                    # movb    $0x0b.%al
```

Mark only one oval.

- Yes
- No

15. The above opcode has 10 lines of code, let's say the line no. of the first instruction is 1. Identify all line(s) that generate a "0". (Select the relevant line numbers in the opcode listing above). * 2 points

Check all that apply.

- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7
- [ ] 8
- [ ] 9
- [ ] 10

16. Identify all lines that push "name[0]", the first argument of execve(), into the stack (refer to the corresponding C code above). (Select the relevant line number(s) in the opcode listing above). * 2 points

Check all that apply.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ 8
- ☐ 9
- ☐ 10

17. Identify all line(s) that push "name", the 2nd argument of execve(), into the stack (Hint: "name" is the address of the name array). (Select the relevant line number(s) in the opcode listing above). * 2 points

Check all that apply.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ 8
- ☐ 9
- ☐ 10

18. Identify all line(s) that handle the 3rd argument of execve(), NULL, into the stack. (Select the relevant line number(s) in the opcode listing above). * 2 points

Check all that apply.

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7
- ☐ 8
- ☐ 9
- ☐ 10

19. Since it is difficult to guess the exact starting address of the exploit code, which we want to execute, which of the following changes do we make to our shell code? * 2 points

Mark only one oval.

- ◯ Put a bunch of Null Characters, at the start of the opcode of our exploit code, while feeding the opcode to a target array.
- ◯ Put a bunch of No Operation instructions, at the start of the opcode of our exploit code, while feeding the opcode to a target array.

This content is neither created nor endorsed by Google.

Google Forms