Buffer Overflow Attack - Exercise

Total points 18/22

Email address *

aftab.hussain46@gmail.com

Please enter your Student ID Number *

0000000

Please enter your UCINetID *

test

Please enter your Full Name *

test

Questions:

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

- ○ From lower addresses to higher addresses.
- ◉ From higher addresses to lower addresses. ✓

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

- ○ Yes
- ◉ No ✓

Feedback

It is the return address region of the current stack frame, that points to this location, not $ebp.

✓ 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? * 1/1

- ○ Yes
- ◉ No ✓

Feedback

The return address points to the location referred to in the previous question.

✓ The "oldebp" region of a stack frame points to: * 1/1

- (●) 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

✓ 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): * 1/1

- [x] Return address region ✓
- [x] Old ebp region ✓
- [x] Local variables region ✓
- [ ] None of the above

Feedback

While our main goal in a buffer overflow attack is to write on the return address region, the other regions would also be overwritten.

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

- (●) 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.

Feedback

Say our character array contains "hello world". "h" would be stored in the lowest stack address as compared to the other letters, "e" would be in the next higher position, and so on. Thus, in the stack, the array contents will start from the lower stack addresses, to the higher ones.

✓ 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? * 1/1

- [x] 0x400e13ab ✓
- [ ] 0xb7e42da0
- [ ] 0xb7e4036d
- [x] 0x5ea00d00 ✓
- [ ] All would work.
- [ ] None would work.

Feedback

The strcpy() function would stop the copy process as soon as it encounters a null byte. In the hexadecimal addresses, each character (after 0x) represents 4 bits (a nibble). Thus, when we have two consecutive 0's in the hex address, these two characters would represent a null byte and thus foil our attack.

✓ 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? * 2/2

```
#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
```

- ( ) 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. ✓

Feedback

The binary may have null bytes. There may be other reasons too, e.g. incorporating dynamically linked libraries. For these reason, we use shell code to directly inject the opcode of our exploit program.

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

- ○ memory
- ◉ registers ✓
- ○ cache
- ○ None of the above.

Feedback

The three arguments are taken from the following registers respectively: $ebx,$ecx,$edx

✓ 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) * 2/2

```
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
```

- ( ) Yes
- (•) No ✓

Feedback

We should add the instruction that invokes the system call. This is "int $0x80" in assembly, i.e, "\xcd\x80" in opcode.

✓ 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). * 2/2

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

✓ 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). * 2/2

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

Feedback

Only choose the push instructions. Note the need to push "0" after pushing "/bin/sh". As all strings need to be null terminated.

✗ 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). * 0/2

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

Correct answer

- [x] 7

Feedback

After pushing "/bin//sh" into the stack, the contents of the name array are at the top of the stack, which is pointed to by %esp, which is essentially the address of name array at this stage. Line 5 then saves this address into the %ebx register. The %ebx register value is then pushed into the stack in line 7.

✗ 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). * 0/2

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

Correct answer

- [x] 9

✓ 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? * 2/2

- ○ 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. ✓

Feedback

WIth NOPs preceding the exploit code, we only need to ensure we can jump to any one of the NOP instructions, in order to execute our code.