Roll Number:

Thapar Institute of Engineering and Technology, Patiala

Department of Computer Science & Engineering

BE- CoE, CSE (VI Semester) MST 7 March 2023 **UCS638: Secure Coding**

Time: 02 Hours; MM: 30 Name of Faculty: Maninder Singh, Ashima Singh

Q1. Study given debug snippets and answer the following:

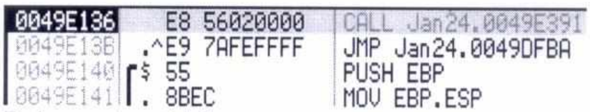
i) What does Debug snippet 1.1 showcase?

ii) In Debug snippet 1.2 first line is changed to jmp instruction, what does 0056D000 means? Correlate this address with your understanding of sections and explain .text, .rdata, .data, .rsrc, .MSingh sections in this context.

iii) What does Debug snippet 1.3 shows, why EIP is pointing to 0049E136?

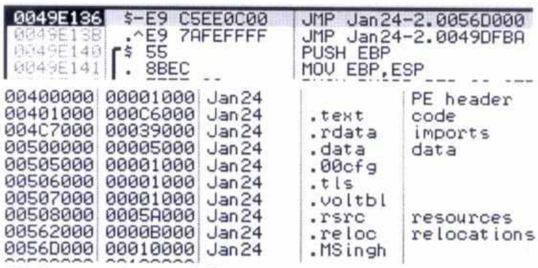
iv) What does Debug snippet 1.4 shows, why ModuleEntryPoint is listed at the bottom?

v) What does Debug snippet 1.5 shows, what do you understand by four push statements, What is meaning of 0056D02A address, what will be the status of stack if instruction at address 0056D014 is changed to NOP?



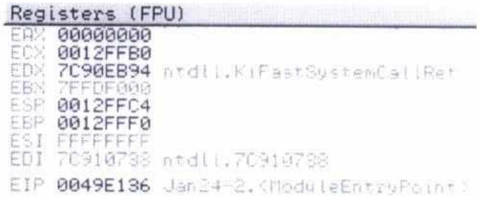
```
0049E136   E8 56020000   CALL Jan24.0049E391
0049E13B  .^E9 7AFEFFFF  JMP Jan24.0049DFBA
0049E140 ┌$ 55           PUSH EBP
0049E141 │. 8BEC         MOV EBP.ESP
```

Debug snippet 1.1



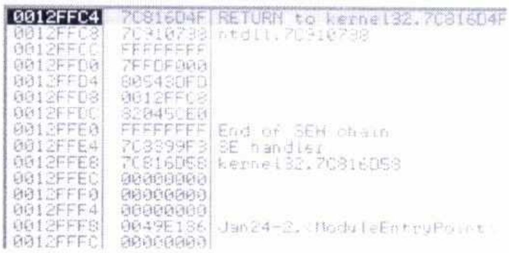
```
0049E136  $-E9 C5EE0C00  JMP Jan24-2.0056D000
0049E13B  .^E9 7AFEFFFF  JMP Jan24-2.0049DFBA
0049E140 ┌$ 55           PUSH EBP
0049E141 │. 8BEC         MOV EBP,ESP

00400000 00001000 Jan24             PE header
00401000 000C6000 Jan24   .text     code
004C7000 00039000 Jan24   .rdata    imports
00500000 00005000 Jan24   .data     data
00505000 00001000 Jan24   .00cfg
00506000 00001000 Jan24   .tls
00507000 00001000 Jan24   .voltbl
00508000 0005A000 Jan24   .rsrc     resources
00562000 0000B000 Jan24   .reloc    relocations
0056D000 00010000 Jan24   .MSingh
```

Debug snippet 1.2



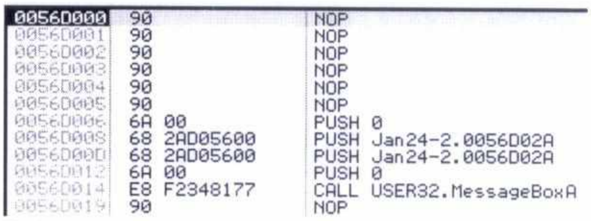
```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738

EIP 0049E136 Jan24-2.<ModuleEntryPoint>
```

Debug snippet 1.3



```
0012FFC4  7C816D4F  RETURN to kernel32.7C816D4F
0012FFC8  7C910738  ntdll.7C910738
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  805430FD
0012FFD8  0012FFC8
0012FFDC  820450E0
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C8399F3  SE handler
0012FFE8  7C816D58  kernel32.7C816D58
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0049E136  Jan24-2.<ModuleEntryPoint>
0012FFFC  00000000
```

Debug snippet 1.4

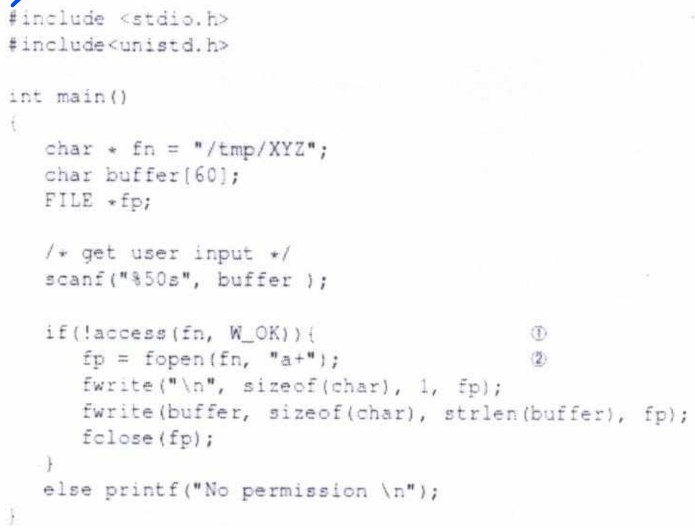


```
0056D000  90            NOP
0056D001  90            NOP
0056D002  90            NOP
0056D003  90            NOP
0056D004  90            NOP
0056D005  90            NOP
0056D006  6A 00         PUSH 0
0056D008  68 2AD05600   PUSH Jan24-2.0056D02A
0056D00D  68 2AD05600   PUSH Jan24-2.0056D02A
0056D012  6A 00         PUSH 0
0056D014  E8 F2348177   CALL USER32.MessageBoxA
0056D019  90            NOP
```

Debug snippet 1.5

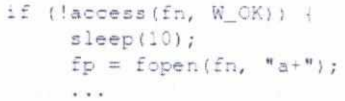
(2, 2, 2, 2, 2)

Q2.



```
#include <stdio.h>
#include<unistd.h>

int main()
{
   char * fn = "/tmp/XYZ";
   char buffer[60];
   FILE *fp;

   /* get user input */
   scanf("%50s", buffer );

   if(!access(fn, W_OK)){                          ①
      fp = fopen(fn, "a+");                        ②
      fwrite("\n", sizeof(char), 1, fp);
      fwrite(buffer, sizeof(char), strlen(buffer), fp);
      fclose(fp);
   }
   else printf("No permission \n");
}
```

Code snippet 2.1



```
if (!access(fn, W_OK)) {
     sleep(10);
     fp = fopen(fn, "a+");
     ...
```

Code snippet: 2.2

a) Study the given code snippet 2.1 and write technical comments about highlighted part marked as 1 and 2. What will happen if this code is changed to the variant as shown in code snippet 2.2. Using both the variants, spell out how the root account can be added to the system.

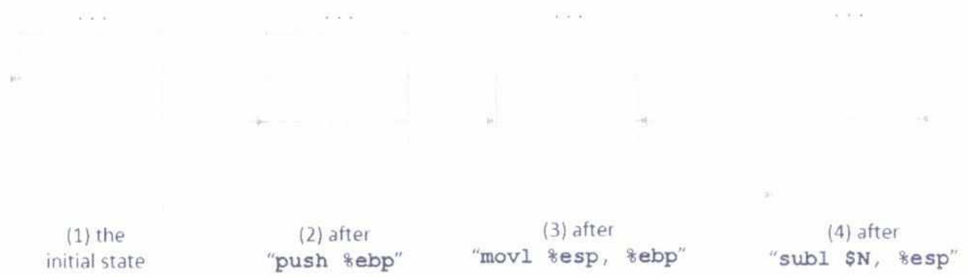
b) Assume you are using Ubuntu 12.04. According to the documentation, "symlinks in world-writable sticky directories (e.g. /tmp) cannot be followed if the follower and directory owner do not match the symlink owner." How this Countermeasure can be Turned Off?

c) There are three kinds of Honeypots used in security implementation: elaborate each of one of these with appropriate use case.

(5, 2, 3)

Q3.

a) Fill the blank spaces and arrow positions for the function prologue execution:


(1) the initial state

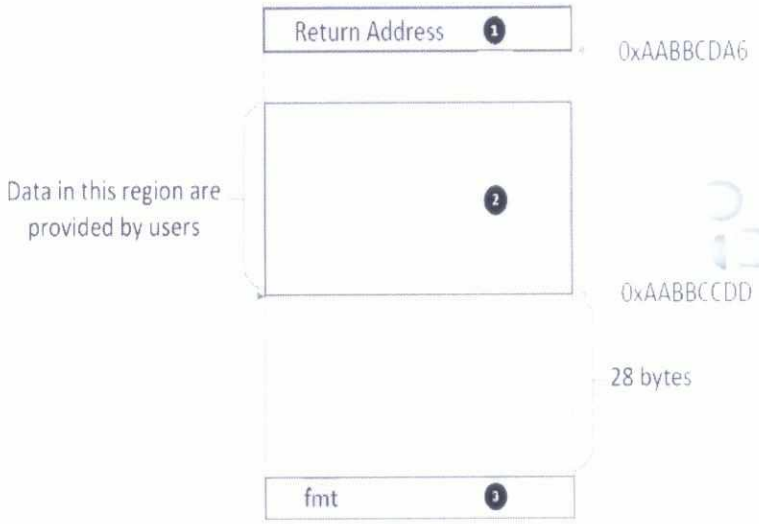
(2) after "`push %ebp`"

(3) after "`movl %esp, %ebp`"

(4) after "`subl $N, %esp`"

b) When printf(fmt) is executed, the stack (from low address to high address) contains the following values (4 bytes each), where the first number is the content of the variable fmt, which is a pointer pointing to a format string. If you can decide the content of the format string, what is the smallest number of format specifiers that you can use to crash the program with a 100 percent probability?

0xAABBCCDD, 0xAABBDDFF, 0x22334455, 0x99663322, 0x00000000



c) A server program takes an input from a remote user, saves the input in a buffer allocated on the stack. The address of this buffer is then stored in the local variable `fmt`, which is used in the following statement in the server program:

```
printf(fmt);
```

When the above statement is executed, the current stack layout is shown. If you are a malicious attacker, can you construct the input, so when the input is fed into the server program, you can get the server program to execute your code? Please write down the actual content of the input (you do not need to provide the exact content of the code; just put "malicious code" in your answer, but you need to put it in the correct location). Mere mentioning the format string will not fetch any credit, you need to explain: how these calculations are done step by step.

Note: Assume malicious code is at 0XAABBCCEE

If your answer causes the server to print out more than a billion characters, it may take a while for your attack to succeed. Please revise your answer, so the total number of characters printed out is less than 60,000.

(3, 2, 5)