Introduction to Malware Analysis

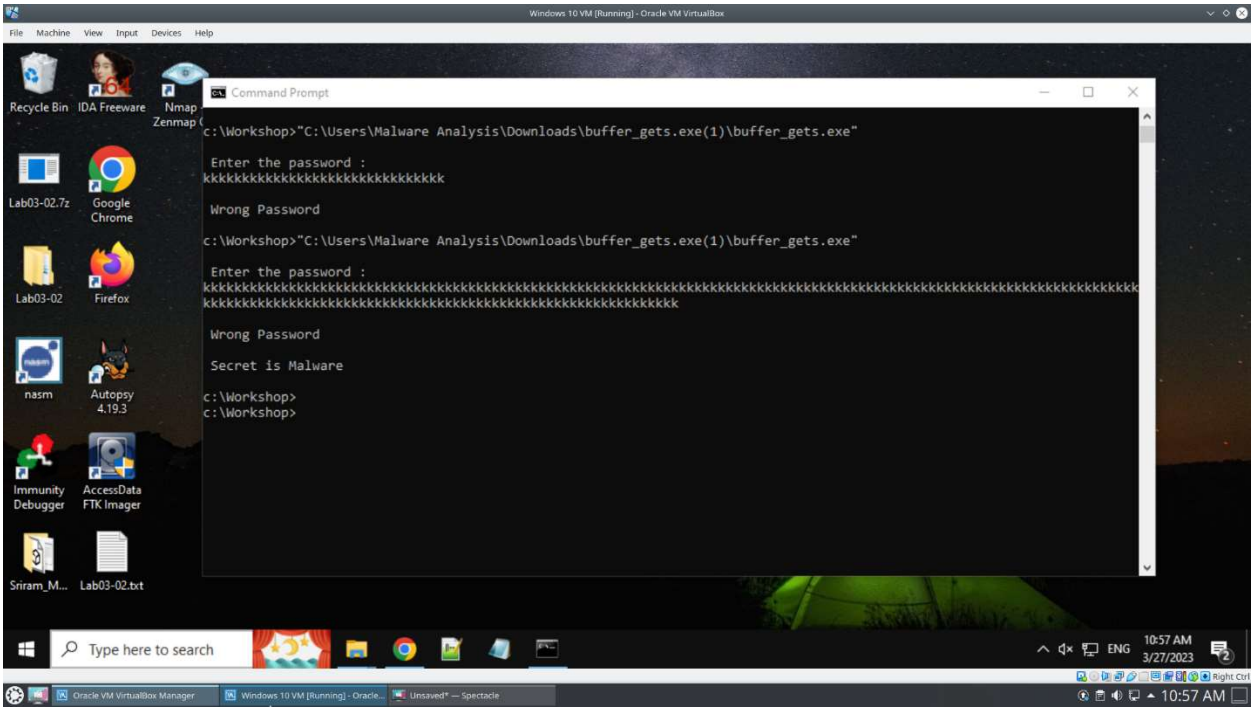
Assignment 6 – Immunity Debugger/OllyDbg

10 points

LAB – IMMUNITY DEBUGGER

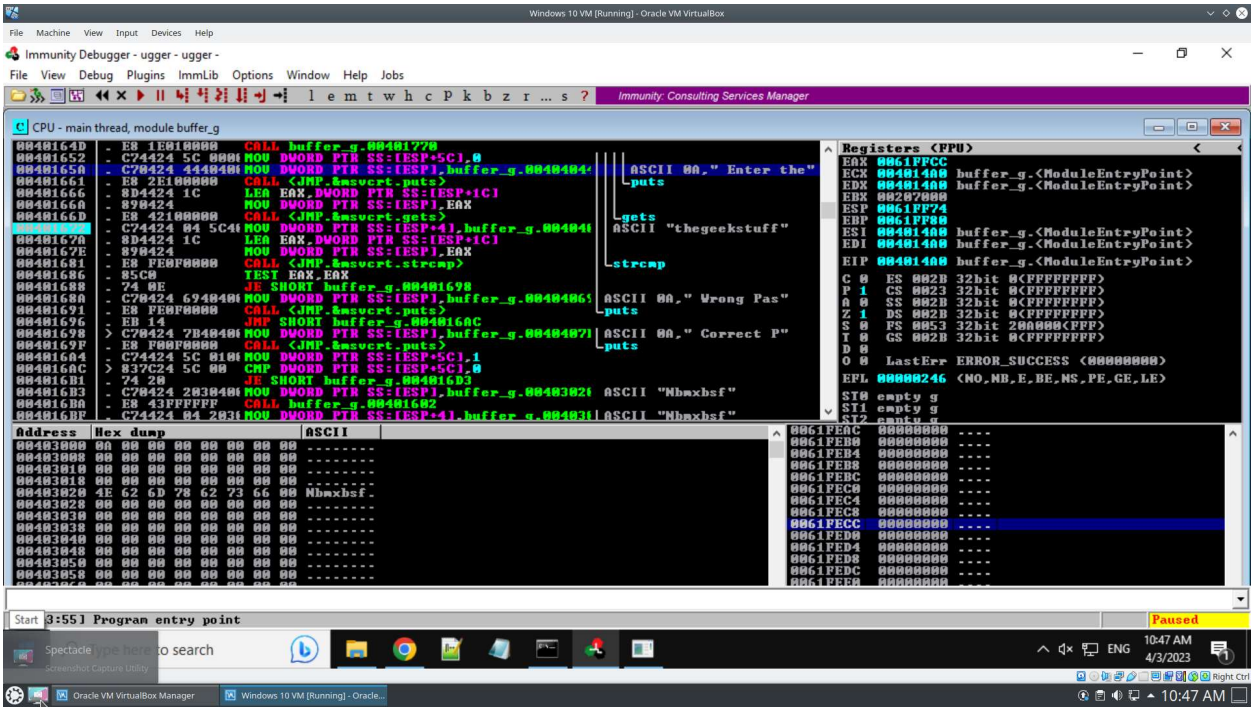
The attached **program *buffer_gets.exe*** requires a password to show a secret message. Design a long password to the program to deploy a buffer overflow attack so that even if the password is wrong, the program shows the secret message. The source code of the binary *buffer_gets.c* is also provided in Figure 1.

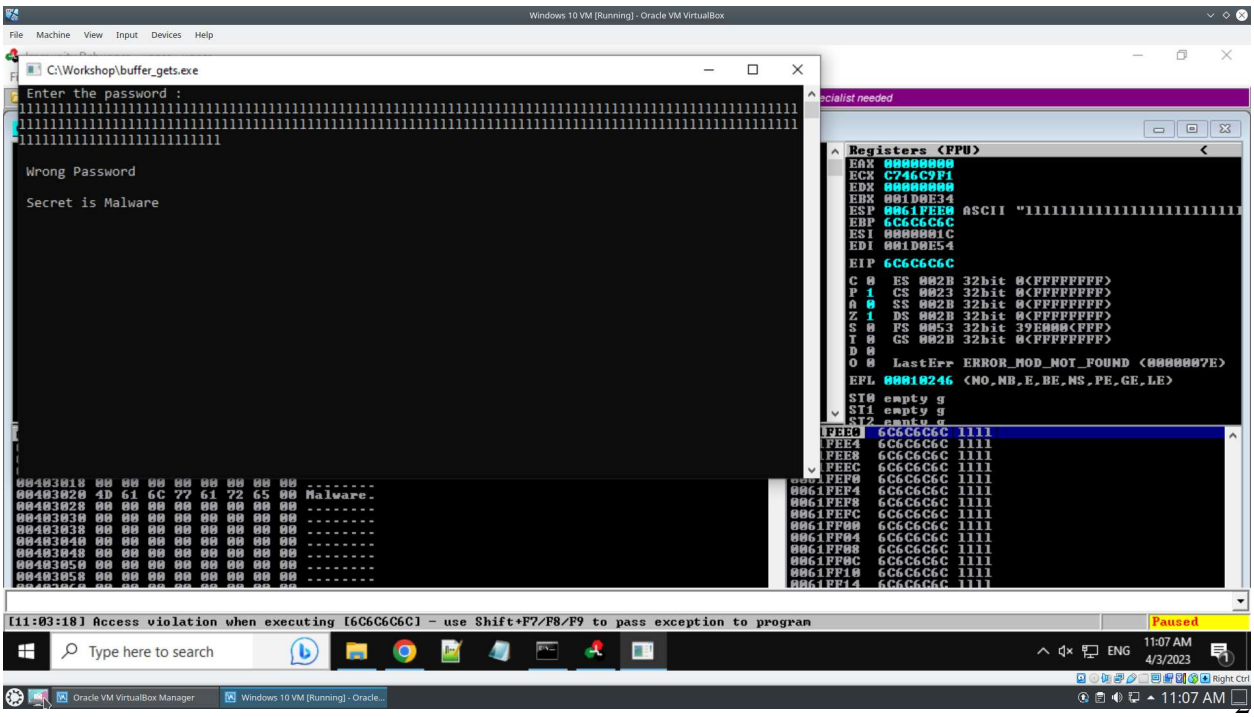
1. Provide a screenshot to show that the buffer overflow works and the secret message. (4 points)



2. Provide at least one screenshot of Immunity Debugger showing the buffer overflow successfully changes the stack and the change makes the attack work. Highlight the changed value, i.e., *pass* in *buffer_gets.c*, on the screenshot. (3 points)

Hint 1: The purpose of this buffer overflow attack is not to change the return address. In Figure 1, we can see the variable *pass* controls if the code shows the secret or not. Actually *pass* is allocated on the stack above *buff* if you look at the assembly and the stack in IDA Pro or immunity debugger. Can overflowing *buff* overwrite *pass*?
Hint 2: To supply the arguments to a running program within Immunity Debugger, *Debug -> Arguments*. After the arguments are provided, Immunity Debugger will notify you the program shall be restarted: *Debug -> Restart*





3. Explain how this buffer overflow attack works and why the secret can be obtained with the buffer overflow attack. (3 points)

The buffer overflow works by inputting too many/a very long argument into the input. Which means too much data was put into he buffer and the program than outputs whatever is contains. The buffer was a size of 64, so I inputted over 64 l's to do the buffer overflow attack.

Figure 1 buffer_gets.c

```c
/* buffer_gets.c */

#include <stdio.h>
#include <string.h>

char notSecret[128] = "Nbmxbsf";

void secretEncoder(unsigned char *fustr) {
    for (int i = 0; i < strlen(fustr); i++) {
        fustr[i] = fustr[i]+1;
    }
}

void secretDecoder(unsigned char *fustr) {
    for (int i = 0; i < strlen(fustr); i++) {
        fustr[i] = fustr[i]-1;
    }
}

int main(void)
{
    char buff[64];
    int pass = 0;


    printf("\n Enter the password : \n");
    gets(buff);

    if (strcmp(buff, "thegeekstuff"))
    {
        printf("\n Wrong Password \n");
    }
    else
    {
        printf("\n Correct Password \n");
        pass = 1;
    }


    if (pass)
    {
        // Now Give root or admin rights to user
        secretDecoder(notSecret);
        printf("\n Secret is %s\n", notSecret);
    }
    return 0;
}
```