PWNABLE.KR - random

Lets connect to the server

```
ra@moni~/P/p/random> ssh random@128.61.240.205 -p 2222
random@128.61.240.205's password:
 ____  __    __  ____    ____  ____   _        ___      __  _  ____
|    \|  |__|  ||    \  /    ||    \ | |      /  _]    |  |/ ]|    \
|  o  )  |  |  ||  _  ||  o  ||  o  )| |     /  [_     |  ' / |  D  )
|   _/|  |  |  ||  |  ||     ||     || |___ |    _]    |    \ |    /
|  |  |  `  '  ||  |  ||  _  ||  O  ||     ||   [_  __ |     \|    \
|  |   \      / |  |  ||  |  ||     ||     ||     ||  ||  .  ||  .  \
|__|    \_/\_/  |__|__||__|__||_____||_____||_____||__||__|\_||__|\_|

- Site admin : daehee87@gatech.edu
- IRC : irc.netgarage.org:6667 / #pwnable.kr
- Simply type "irssi" command to join IRC now
- files under /tmp can be erased anytime. make your directory under /tmp
- to use peda, issue `source /usr/share/peda/peda.py` in gdb terminal
You have new mail.
Last login: Fri Jun  4 17:29:35 2021 from 82.196.111.219
random@pwnable:~$
```

Lets list the files using `ls -la`,

```
random@pwnable:~$ ls -la
total 40
drwxr-x---   5 root       random 4096 Oct 23  2016 .
drwxr-xr-x 115 root       root   4096 Dec 22 08:10 ..
d---------   2 root       root   4096 Jun 30  2014 .bash_history
-r--r-----   1 random_pwn root     49 Jun 30  2014 flag
dr-xr-xr-x   2 root       root   4096 Aug 20  2014 .irssi
drwxr-xr-x   2 root       root   4096 Oct 23  2016 .pwntools-cache
-r-sr-x---   1 random_pwn random 8538 Jun 30  2014 random
-rw-r--r--   1 root       root    301 Jun 30  2014 random.c
```

As usual there are some privilege restrictions to access the `flag`,we have to read it through our binary

Lets analyze the file type of our binary using `file` command,

```
random@pwnable:~$ file random
random: setuid ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/l, for GNU/Linux 2.6.24,
BuildID[sha1]=f4eac0a1434a84aef72dfabfc1f889e6f6f73023, not stripped
```

So it is a `not stripped` binary

Lets view the source code of the binary,

```
random@pwnable:~$ 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;
}
```

Lets try running our binary,

```
random@pwnable:~$ ./random
12
Wrong, maybe you should try 2^32 cases.
random@pwnable:~$ ./random
1234
Wrong, maybe you should try 2^32 cases.
random@pwnable:~$ ./random
3735928559
Wrong, maybe you should try 2^32 cases.
```

So its expecting a different input

To bypass this we need to find that `rand()` is producing random output or fixed one

```
ra@moni~/P/p/random> cat test.c
#include <stdio.h>

int main(){
    unsigned int random;
    random = rand();
    printf("Random is %d",random);
    return 0;
}
ra@moni~/P/p/random> gcc -o test test.c
test.c: In function ‘main’:
test.c:5:11: warning: implicit declaration of function ‘rand’ [-Wimplicit-
function-declaration]
    5 |  random = rand();
      |           ^~~~
ra@moni~/P/p/random> ./test
Random is 1804289383
ra@moni~/P/p/random> ./test
Random is 1804289383
ra@moni~/P/p/random> ./test
Random is 1804289383
ra@moni~/P/p/random> ./test
Random is 1804289383
ra@moni~/P/p/random> ./test
Random is 1804289383
```

NOTE

```
rand() is not a secure function to implement in programs

So this rand() produces same output because the seed in it is not
initialized,if there is different seed each time it would create a
different random number

Here rand() is called with default seed (1),thus it produces the same
output for random number

To prevent this type of security failure by rand() for pseudo random number
generation, use srand()
```

We know that the condition to pass the program is,

`if( (key ^ random) == 0xdeadbeef )`

Now `random` gives `1804289383`,

`key` is our input,

If `random` performs `xor` operation with `key` it should give `0xdeadbeef` as result,

On converting `int` to `hex`,

```
>>> hex(1804289383)
'0x6b8b4567'
```

So `0x6b8b4567 ^ key == 0xdeadbeef`,

Its time for mathematical calculation (Converting into binary format),

```
0x6b8b4567 = 1101011100010110100010101100111

0xdeadbeef = 11011110101011011011111011101111
```

To find the key just XOR these,

I used Online XOR Calculator

So our `key` value should be ,

In `hex` ---> `0xb526fb88`

In `int` ---> `3039230856`

Now if we pass this `int` type `key` in the input,we get our flag

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

Done! We got our flag

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