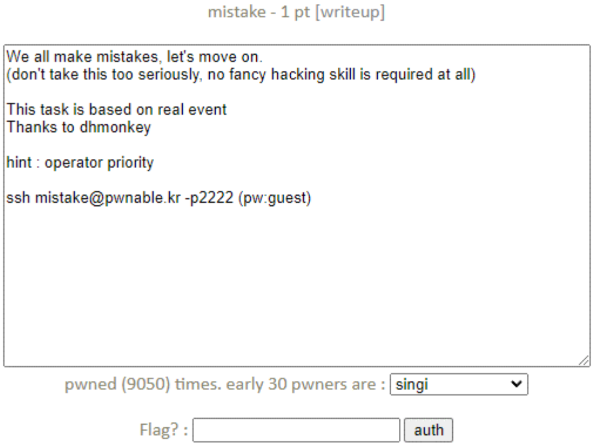
Pwnable.kr
- mistake -

ssh mistake@pwnable.kr –p2222
pw : guest

0. 문제 살펴보기



실제 일어날 수 있는 일을 바탕으로 한 문제라고 한다. 그러면서 연산자 우선순위에 대하여 힌트를 주고 있다.

1. SSH 접속 및 살펴보기


```
mistake@pwnable:~$ ls
flag  mistake  mistake.c  password
mistake@pwnable:~$ 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;
}
```

SSH를 이용해 상단에 표기해 놓은 주소와 포트 번호로 접속하였다.
디렉토리의 파일들을 살펴보자 C 코드 파일이 존재하여 확인해보니 위와 같은 코드를 알 수 있었다.
password 파일을 읽어 배열에 저장한 후, 입력을 받아 이 입력을 xor 연산한 후에 비교하는 것 같다.

2. 취약점 파악

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

첫 번째 if 문부터 봐보자.
힌트로 연산자 우선순위에 대하여 얘기를 하고 있기 때문에 이 점을 유심히 볼 것이다.
현재 언뜻보기에는 open 함수의 return 값을 fd 에 저장하여 0 보다 작은지에 대해 비교하는 것으로 보인다.
하지만 연산자 우선순위에서 '<'는 '='보다 우선순위를 가진다.
따라서, 위의 if 문에서는 open 함수의 return 값을 0 보다 작은지에 대한 결과 값을 fd 에 저장한다는 뜻이다.

open 함수의 return 값은 파일이 정상적으로 존재하고 열리므로 양의 정수를 가질 것이다.
이 양의 정수는 0 보다 크기 때문에 그에 대한 결과 값은 0 (false)이다.
따라서, fd = 0 으로 저장된다.

```
if(!(len=read(fd,pw_buf,PW_LEN) > 0)){
        printf("read error\n");
        close(fd);
        return 0;
}
```

두 번째 if 문을 봐보자.
여기서는 괄호로 인해 len 에 read 함수의 return 값이 len 에 정상적으로 저장될 것이다.
하지만, 지금 read 함수의 fd 값을 보면 0 으로 초기화 되어 있었다.
read 함수에서 fd 값이 0 이면 표준 입력을 가지게 되므로, 현재 pw_buf 배열에 PW_LEN 의 길이만큼 입력을 받아 저장한다는 의미가 된다.
즉, password 를 우리가 조작할 수 있다는 뜻과 같다.

```
char pw_buf2[PW_LEN+1];
printf("input password : ");
scanf("%10s", pw_buf2);
```

우리는 pw_buf 의 값을 지정할 수 있기 때문에, 이후 scanf 를 통해 받는 입력에 대한 결과인 pw_buf2 배열의 값과 같게 만들면 될 것이다.

```
#define XORKEY 1
xor(pw_buf2, 10);
void xor(char* s, int len){
        int i;
        for(i=0; i<len; i++){
                s[i] ^= XORKEY;
        }
}
```

이 xor 함수를 통해 각 배열의 원소를 xor 연산으로 처리하기 때문에 pw_buf2 의 각 원소의 값이 0 과 1 만 있을 때를 가정하여 1 과 같으면 0, 1 과 다르면 1 이 배열에 다시 저장될 것이다.

따라서, 우리는 pw_buf 에는 pw_buf2 를 xor 연산을 한 값이 들어가게 하면 된다.

※ 연산자 우선순위

기호 [1]	연산 유형	associativity
`[ ]` `( )` `.` `->` `++` `--` (후위)	식	왼쪽에서 오른쪽
`sizeof` `&` `*` `+` `-` `~` `!` `++` `--` (전위)	단항	오른쪽에서 왼쪽
형식 캐스팅	단항	오른쪽에서 왼쪽
`*` `/` `%`	곱하기	왼쪽에서 오른쪽
`+` `-`	더하기	왼쪽에서 오른쪽
`<<` `>>`	비트 시프트	왼쪽에서 오른쪽
`<` `>` `<=` `>=`	관계	왼쪽에서 오른쪽
`==` `!=`	같음	왼쪽에서 오른쪽
`&`	비트 AND	왼쪽에서 오른쪽
`^`	비트 제외 OR	왼쪽에서 오른쪽
`|`	비트 포함 OR	왼쪽에서 오른쪽
`&&`	논리 AND	왼쪽에서 오른쪽
`||`	논리 OR	왼쪽에서 오른쪽
`? :`	조건식	오른쪽에서 왼쪽
`=` `*=` `/=` `%=` `+=` `-=` `<<=` `>>=` `&=` `^=` `|=`	단순 및 복합 할당 [2]	오른쪽에서 왼쪽
`,`	순차적 계산	왼쪽에서 오른쪽

(출처: miscrosoft)

3. 공격

```
mistake@pwnable:~$ ./mistake
do not bruteforce...
1111111111
input password : 0000000000
Password OK
```

만약, 우리가 scanf로 입력할 문장이 '0000000000'이라면 1과의 xor 연산으로 인해 '1111111111'로 바뀔 것이다. 따라서, 처음 read 함수를 통해 받는 문장을 '1111111111'로 입력하고 scanf에서 '0000000000'으로 입력하면 flag를 얻을 수 있다.

```
mistake@pwnable:~$ ./mistake
do not bruteforce...
0000000000
1111111111
input password : Password OK
```

반대로 우리가 scanf로 입력할 문장이 '1111111111'이라면 xor 연산으로 인해 '0000000000'이 되므로 처음 read 함수를 통해 입력하는 문장을 '0000000000'으로 하면 된다.