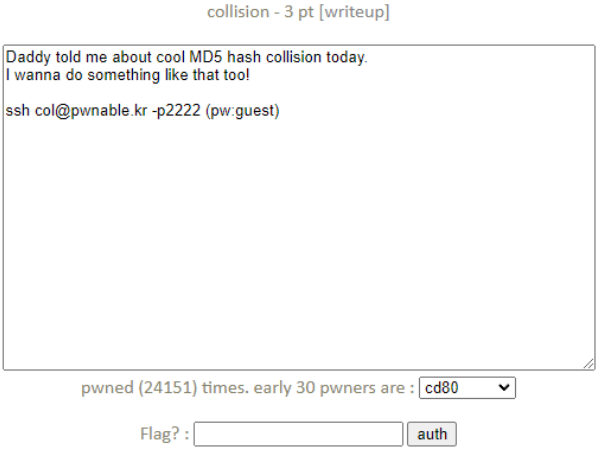
Pwnable.kr
- collision -

ssh col@pwnable.kr -p2222
pw : guest

0. 문제 살펴보기



문제에서 MD5 hash 충돌에 관한 얘기를 하는 것을 볼 수 있다.

1. SSH 접속 및 살펴보기

SSH를 이용해 상단에 표기해놓은 주소와 포트 번호로 접속한다.

디렉토리의 파일들을 살펴보자 C 코드 파일이 존재하여 확인해보니 위와 같은 코드를 알 수 있었다.
프로그램 실행 시, 글자 수가 20 인 인자를 하나 넘겨 그것이 hashcode 와 같을 경우 flag 파일을 볼 수 있는 것 같다.

※ MD5 hash collision
MD5는 함수에 의하여 512비트 데이터 블록에서 128비트 해시 값을 생성하여 데이터를 암호화한다.
하지만, 이 알고리즘의 약점이 발견되었다.
동일한 128비트 해시 값을 생성하는 두 개의 서로 다른 메시지가 존재하는 것이다.
이로 인해 해시 값을 유지하면서 파일이나, 메시지의 내용을 변경할 수 있게 된 것이다.

2. 취약점 파악

```
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;
}
```

현재 비밀번호를 확인하는 함수를 보면 포인터로 인자로 받고 int형 포인터 변수인 ip 또한 p를 강제 캐스팅하여 저장한 모습이다.
포인터는 4 bytes의 크기를 가지고 있다. 만약, int형 포인터 변수인 ip를 배열로 나타낼 때, p[0]의 크기를 확인해보면 4 bytes이다. p[1]의 크기를 확인해보아도 4 bytes일 것이다.
따라서, 우리는 ip에 우리가 입력한 비밀번호가 저장될 때, 4글자 즉, 4 bytes씩 끊어서 저장된다는 것을 알 수 있다.

```
#include <stdio.h>

int main() {
        char* p = "\x01\x00\x00\x00\x02\x00\x00\x00\x03\x00\x00\x00\x04\x00\x00\x00\x05\x00\x00\x00";
        int* ip = (int *)p;
        for(int i = 0; i < 5; i++) {
                printf("value: %d, size: %d\n", ip[i], sizeof(ip[i]));
        }
        return 0;
}
```

이해를 위해 다음과 같은 코드를 작성해보았다.
p에 16진수로 된 값을 저장하였다. 프로그램은 리틀 엔디안 방식을 사용하므로 우리의 예상대로라면 ip 배열에는 4글자씩 나눠서 들어가 출력 값으로 1, 2, 3, 4, 5가 순서대로 나올 것이다.

```
┌──(kali㉿kali)-[~]
└─$ ./col
value: 1, size: 4
value: 2, size: 4
value: 3, size: 4
value: 4, size: 4
value: 5, size: 4
```

우리의 예상은 맞았다. 각 값은 1, 2, 3, 4, 5이며 그 크기는 모두 4 bytes씩이다.
그렇다면 우리는 이제 이러한 정보를 이용하여 공격을 할 수 있을 것이다.

3. 공격

현재 hashcode의 값은 0x21DD09EC이다. 현재 check_password() 함수에서 각 ip 배열의 값을 res 변수에 더하여 res 변수를 return 하고 있다. 이 값을 hashcode의 값과 비교하는데, 그렇다면 우리는 위에서 검증한 방식으로 값을 입력해야 한다.
hashcode = ip[0] + ip[1] + ip[2] + ip[3] + ip[4]의 값을 가지면 된다.
0x21DD09EC를 다섯 등분 하면 되는데 이 값은 0x06C5CEC8 * 4 + 0x06C5CECC이다.
이를 우리는 리틀 엔디안 방식으로 프로그램 실행 시, 인자로 넘겨주면 된다.

```
col@pwnable:~$ ./col `python -c 'print "\xc8\xce\xc5\x06" * 4 + "\xcc\xce\xc5\x06"'`
```

파이썬의 출력을 이용하여 위의 값을 리틀 엔디안 방식으로 전달하였다.
그러자, flag 값을 확인할 수 있었다.