CERT-IS Seminar

designed By. 스마일게이트

INDEX

0x04

SQL Injection

SQL 사전지식 조금

DBMS(DB) 는 데이터를 관리하는 시스템임
-> 데이터를 조작하는 구문
-> 데이터를 정의하는 구문
크게 2개로 나뉨 (교수님이 좋아함)

SQL 사전지식 조금

DBMS(DB) 는 데이터를 관리하는 시스템임
-> 데이터를 조작하는 구문 (데이터를 넣거나 읽거나)
-> 데이터를 정의하는 구문 (통째로 만들거나 없앨때)

공격자 입장에서 자주 마주하게 될 구문은
- SELECT
- INSERT

1. SELECT 구문

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT 는 출력시키는 구문이라 보면됨

print(1) == SELECT 1

1. SELECT 구문

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT 는 출력시키는 구문이라 보면됨

FROM은

- 존재하는 테이블을 선언
- 임시 테이블 만들고 선언

SELECT name,pw FROM userdata

1. SELECT 구문

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT 는 출력시키는 구문이라 보면됨

FROM은 테이블을 지정

WHERE은 검색 조건

SELECT name FROM userdata WHERE pw='1234'

1. SELECT 구문

유저의 입력이 주로 일어나는 부분은 WHERE 부분
->SELECT name FROM userdata WHERE id='입력1' and pw='입력2'
ex) id='certis' and pw='certis1004'

single quote(') 나 double quote(") 안에 들어가 있는것은 모두 문자열 취급한다.

single/double quote 안에 들어있지 않은건 모두 구문(syntax) 취급한다.

1. SELECT 구문

SELECT name FROM userdata WHERE id='입력1' and pw='입력2';

문자열에 "'입력" 이 들어갔다고 가정해보자.

SELECT name FROM userdata WHERE id=' '입력1' and pw='입력2';

escape

1. SELECT 구문

SELECT name FROM userdata WHERE id='입력1' and pw='입력2';

문자열에 "'입력" 이 들어갔다고 가정해보자.

SELECT name FROM userdata WHERE id=' '입력1' and pw='입력2';

구문을 맞춰주면 동작한다

SELECT name FROM userdata WHERE id=' '입력1-- ' and pw='입력2';

1. SELECT 구문

이 상태에서 입력에 비교구문을 넣으면 동작한다.

SELECT name FROM userdata WHERE id=' ' or id='admin'-- ' and pw='입력2';

SQL 구문을 좀더 잘 안다면 union을 통해 다른 정보를 얻어 올 수도 있다.

union은 결과를 덧붙여서 내어준다.

1. SELECT 구문

```
mysql> SELECT 'admin';
+-------+
| admin |
+-------+
| admin |
+-------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 'admin' union SELECT 'no-admin';
+----------+
| admin    |
+----------+
| admin    |
| no-admin |
+----------+
2 rows in set (0.00 sec)
```

1. SELECT 구문

union의 이후 구문은 SELECT 구문과 동일하며,

이전의 SELECT 문과 동일한 Column 갯수를 반환해야 한다.

ex)SELECT ‘admin’,’password’ UNION SELECT ‘guest’ //error
SELECT ‘admin’,’password’ UNION SELECT ‘guest’,’guest’ // OK

1. SELECT 구문

여러가지 함수를 사용해서 n글자씩 정보를 유출하는 방법도 있다.

pw=certis1004! 라는 값을 유출한다고 가정해보자.

if(substr(pw,1,1)='a',True,False)

1. SELECT 구문

if(substr(pw,1,1)='a',True,False)

가장 먼저 실행되는것은 if함수 안에 있는 substr 구문이다.

pw라는 컬럼값에서 1번째 index(둘째인자)에서 1글자(셋째인자)를 잘라온다

이 값이 a와 같은지 비교한다

본예제에서는 certis1004!의 첫글자인 c와 a가 같지 않다 라고 판단한다.

1. SELECT 구문

if(substr(pw,1,1)='a',True,False)

같지 않았기 때문에 False의 값으로 건너뛰고 이를 출력하거나
SQL 구문내에서 사용한다.

True/False에 따라 웹페이지에서 출력되는 결과가 다르므로 이를 통해
데이터를 유추할수 있다.

2. INSERT 구문

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

INSERT INTO 테이블(컬럼1,컬럼2..) VALUES(값1, 값2..)

보통 입력은 값1, 값2.. 등에 들어가므로 크게 다른 부분은 없다.

SELECT와 같이 바로 출력받아 볼수 있는게 아니므로
별도의 확인이 필요하다

2. INSERT 구문

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

mysql은 여러개의 values를 한 줄에 입력 가능하다

INSERT INTO userdata(name,pw) VALUES("1","2"),("3","4");

이것을 응용하여 원하는 값을 Injection 할 수 있다.

INSERT INTO userdata(name,pw) VALUES("1"," "),("test","zxcv")

3. 번외

DBMS에는 모든 Table 이름과 Column이름을 담고 있는 특수한 DB가 있다.

mysql : information_schema (table_name, column_name)

sqlite : sqlite_master(tbl_name, sql)

mssql : sys.sysobject,sys.syscolumns (name)

이 DB는 항상 고정된 이름의 Table과 Column을 가진다.

3. 번외

만약 여기 내용을 대부분 이해하고 따라왔다면

https://los.rubiya.kr에 도전하고 All clear를 따내는 것을 마지막 과제로

무기한 과제로 내주고 끝

bypass 방법과 같은 것들은 그때그때 찾아서 임기응변으로..

SQLi

End Of Document

