Name: Sumithra.E

Project Title: SQL Injection Vulnerability Demonstration

Platform

The experiment was conducted on **Kali Linux**, a penetration-testing operating system. The vulnerable web application used was **DVWA**, which is intentionally designed to demonstrate common web security flaws. All testing was performed in a controlled lab environment. No real systems were targeted.

Tool Used

DVWA (Damn Vulnerable Web Application) is an open-source PHP/MySQL web application. It provides multiple security levels to practice vulnerabilities safely. The LOW security level was selected to clearly observe SQL Injection behavior.

Introduction

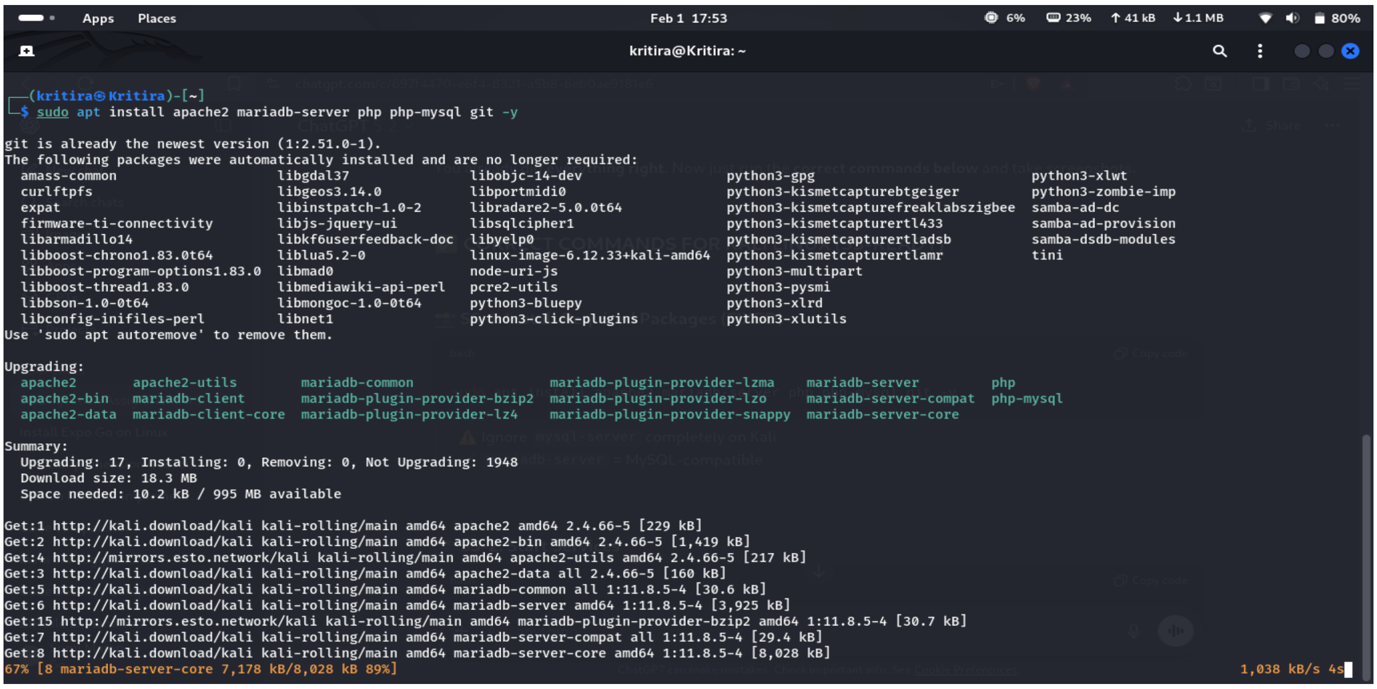
SQL Injection is a web security vulnerability that occurs when user input is directly included in SQL queries without validation. This allows attackers to manipulate database commands. SQL Injection can lead to data leakage, authentication bypass, and database compromise. It remains a major threat in modern web applications.

Objective

The objective of this experiment is to identify and demonstrate SQL Injection in a controlled environment. The experiment shows how improper input handling affects SQL query execution. It also helps understand real-world risks and secure coding practices.

Lab Setup

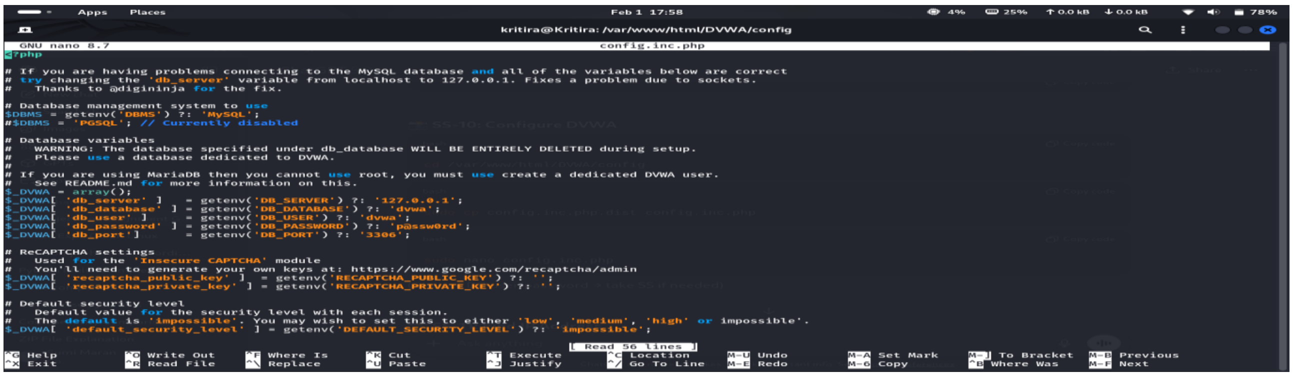
Apache2, MariaDB, and PHP were installed on Kali Linux. DVWA was cloned from GitHub and configured successfully. A dedicated database and database user were created. The security level of DVWA was set to LOW for demonstration purposes.



```
┌──(kritira㉿Kritira)-[~]
└─$ sudo apt install apache2 mariadb-server php php-mysql git -y

git is already the newest version (1:2.51.0-1).
The following packages were automatically installed and are no longer required:
  amass-common                   libgdal37                 libobjc-14-dev                python3-gpg                           python3-xlwt
  curlftpfs                      libgeos3.14.0             libportmidi0                  python3-kismetcapturebtgeiger         python3-zombie-imp
  expat                          libinstpatch-1.0-2        libradare2-5.0.0t64           python3-kismetcapturefreaklabszigbee  samba-ad-dc
  firmware-ti-connectivity       libjs-jquery-ui           libsqlcipher1                 python3-kismetcapturertl433           samba-ad-provision
  libarmadillo14                 libkf6userfeedback-doc    libyelp0                      python3-kismetcapturertladsb          samba-dsdb-modules
  libboost-chrono1.83.0t64       liblua5.2-0               linux-image-6.12.33+kali-amd64 python3-kismetcapturertlamr         tini
  libboost-program-options1.83.0 libmad0                   node-uri-js                   python3-multipart
  libboost-thread1.83.0          libmediawiki-api-perl     pcre2-utils                   python3-pysmi
  libbson-1.0-0t64               libmongoc-1.0-0t64        python3-bluepy                python3-xlrd
  libconfig-inifiles-perl        libnet1                   python3-click-plugins         python3-xlutils
Use 'sudo apt autoremove' to remove them.

Upgrading:
  apache2       apache2-utils      mariadb-common                 mariadb-plugin-provider-lzma    mariadb-server         php
  apache2-bin   mariadb-client     mariadb-plugin-provider-bzip2  mariadb-plugin-provider-lzo     mariadb-server-compat  php-mysql
  apache2-data  mariadb-client-core mariadb-plugin-provider-lz4   mariadb-plugin-provider-snappy  mariadb-server-core

Summary:
  Upgrading: 17, Installing: 0, Removing: 0, Not Upgrading: 1948
  Download size: 18.3 MB
  Space needed: 10.2 kB / 995 MB available

Get:1 http://kali.download/kali kali-rolling/main amd64 apache2 amd64 2.4.66-5 [229 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 apache2-bin amd64 2.4.66-5 [1,419 kB]
Get:4 http://mirrors.esto.network/kali kali-rolling/main amd64 apache2-utils amd64 2.4.66-5 [217 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 apache2-data all 2.4.66-5 [160 kB]
Get:5 http://kali.download/kali kali-rolling/main amd64 mariadb-common all 1:11.8.5-4 [30.6 kB]
Get:6 http://kali.download/kali kali-rolling/main amd64 mariadb-server amd64 1:11.8.5-4 [3,925 kB]
Get:15 http://mirrors.esto.network/kali kali-rolling/main amd64 mariadb-plugin-provider-bzip2 amd64 1:11.8.5-4 [30.7 kB]
Get:7 http://kali.download/kali kali-rolling/main amd64 mariadb-server-compat all 1:11.8.5-4 [29.4 kB]
Get:8 http://kali.download/kali kali-rolling/main amd64 mariadb-server-core amd64 1:11.8.5-4 [8,028 kB]
67% [8 mariadb-server-core 7,178 kB/8,028 kB 89%]                                                                   1,038 kB/s 4s
```

Vulnerability Identification

The application directly uses user input in SQL queries without sanitization. No input validation or prepared statements are implemented. This allows modification of SQL query logic. As a result, attackers can inject malicious SQL code.



```
  GNU nano 8.7                                   config.inc.php
<?php

# If you are having problems connecting to the MySQL database and all of the variables below are correct
# try changing the 'db_server' variable from localhost to 127.0.0.1. Fixes a problem due to sockets.
#   Thanks to @digininja for the fix.

# Database management system to use
$DBMS = getenv('DBMS') ?: 'MySQL';
#$DBMS = 'PGSQL'; // Currently disabled

# Database variables
#   WARNING: The database specified under db_database WILL BE ENTIRELY DELETED during setup.
#   Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedicated DVWA user.
#   See README.md for more information on this.
$_DVWA = array();
$_DVWA[ 'db_server' ]   = getenv('DB_SERVER') ?: '127.0.0.1';
$_DVWA[ 'db_database' ] = getenv('DB_DATABASE') ?: 'dvwa';
$_DVWA[ 'db_user' ]     = getenv('DB_USER') ?: 'dvwa';
$_DVWA[ 'db_password' ] = getenv('DB_PASSWORD') ?: 'p@ssw0rd';
$_DVWA[ 'db_port']      = getenv('DB_PORT') ?: '3306';

# ReCAPTCHA settings
#   Used for the 'Insecure CAPTCHA' module
#   You'll need to generate your own keys at: https://www.google.com/recaptcha/admin
$_DVWA[ 'recaptcha_public_key' ]  = getenv('RECAPTCHA_PUBLIC_KEY') ?: '';
$_DVWA[ 'recaptcha_private_key' ] = getenv('RECAPTCHA_PRIVATE_KEY') ?: '';

# Default security level
#   Default value for the security level with each session.
#   The default is 'impossible'. You may wish to set this to either 'low', 'medium', 'high' or impossible'.
$_DVWA[ 'default_security_level' ] = getenv('DEFAULT_SECURITY_LEVEL') ?: 'impossible';

                                          [ Read 56 lines ]
^G Help      ^O Write Out   ^F Where Is   ^K Cut     ^T Execute   ^C Location     M-U Undo   M-A Set Mark   M-] To Bracket   M-B Previous
^X Exit      ^R Read File   ^\ Replace    ^U Paste   ^J Justify   ^/ Go To Line   M-E Redo   M-6 Copy       ^B Where Was     M-F Next
```

Exploitation Demonstration

Normal input returns a specific user record from the database. When modified input is supplied, the SQL condition changes. This results in unintended data being displayed. The behavior confirms the presence of an SQL Injection vulnerability.

Real-World Scenario

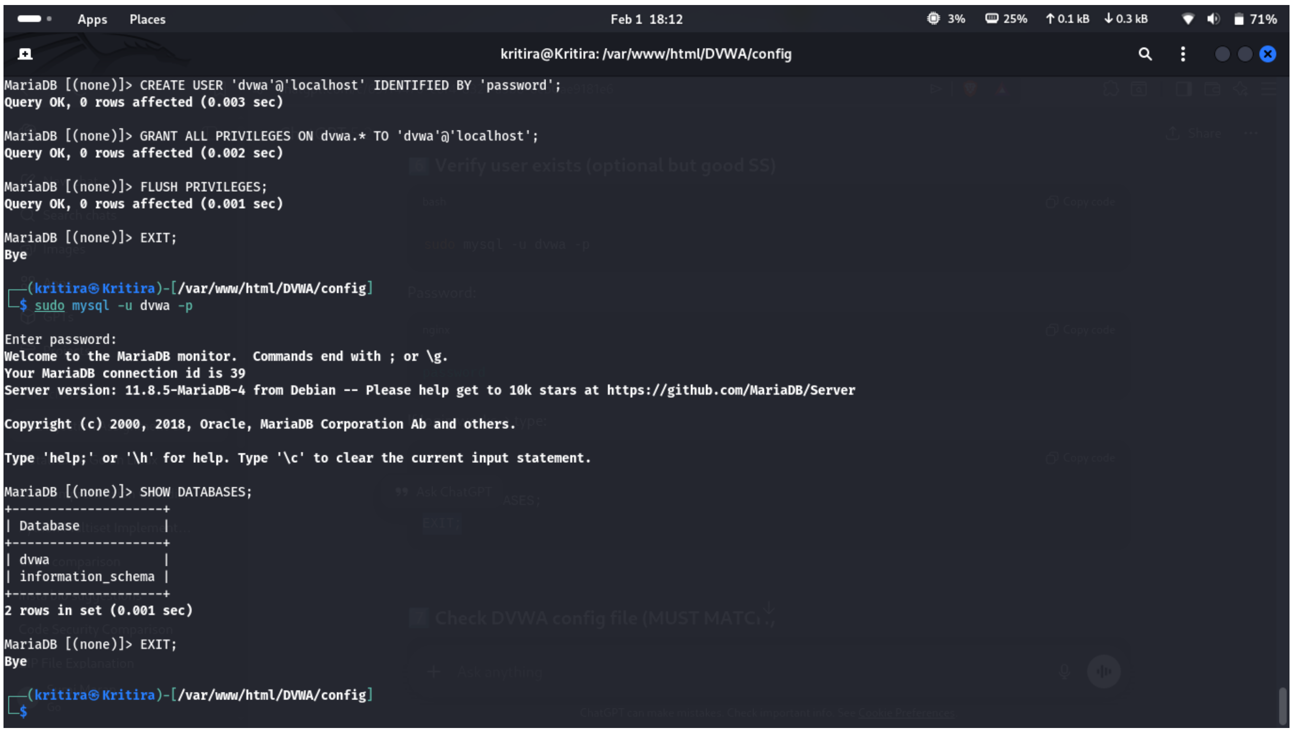
Several organizations have suffered SQL Injection attacks due to insecure coding. Companies such as **Yahoo** experienced large-scale data breaches. E-commerce and government websites are common targets. These incidents highlight the seriousness of SQL Injection vulnerabilities.

Impact

SQL Injection can lead to unauthorized access to sensitive information. Attackers may steal, modify, or delete database records. In severe cases, full database compromise is possible. This can cause financial loss and reputational damage.

Prevention

Prepared statements and parameterized queries should be used. Input validation must be strictly enforced. Database users should have limited privileges. Secure coding practices significantly



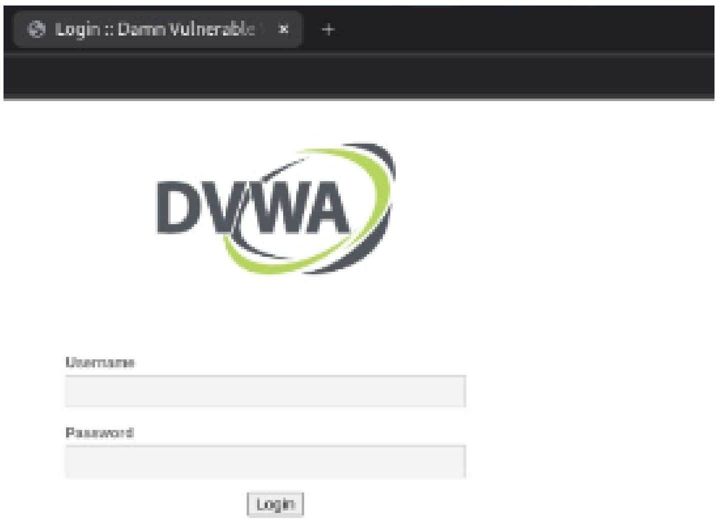
```
MariaDB [(none)]> CREATE USER 'dvwa'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.003 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwa'@'localhost';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> EXIT;
Bye

┌──(kritira㉿Kritira)-[/var/www/html/DVWA/config]
└─$ sudo mysql -u dvwa -p

Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 39
Server version: 11.8.5-MariaDB-4 from Debian -- Please help get to 10k stars at https://github.com/MariaDB/Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| dvwa               |
| information_schema |
+--------------------+
2 rows in set (0.001 sec)

MariaDB [(none)]> EXIT;
Bye

┌──(kritira㉿Kritira)-[/var/www/html/DVWA/config]
└─$
```

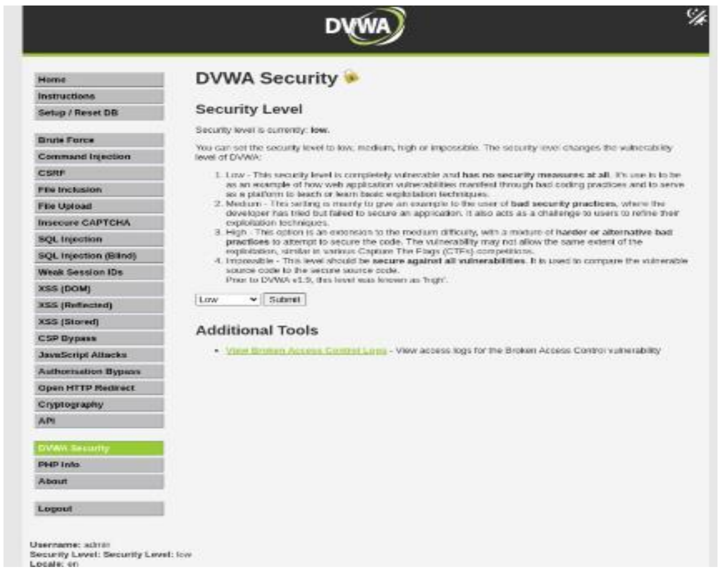
reduce SQL Injection risks.

Conclusion

This experiment demonstrates how SQL Injection occurs due to improper input handling. DVWA provides a safe platform to understand the vulnerability. Real-world incidents show that SQL Injection is still a major threat. Secure development practices are essential to prevent such attacks.

Screenshots:





DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

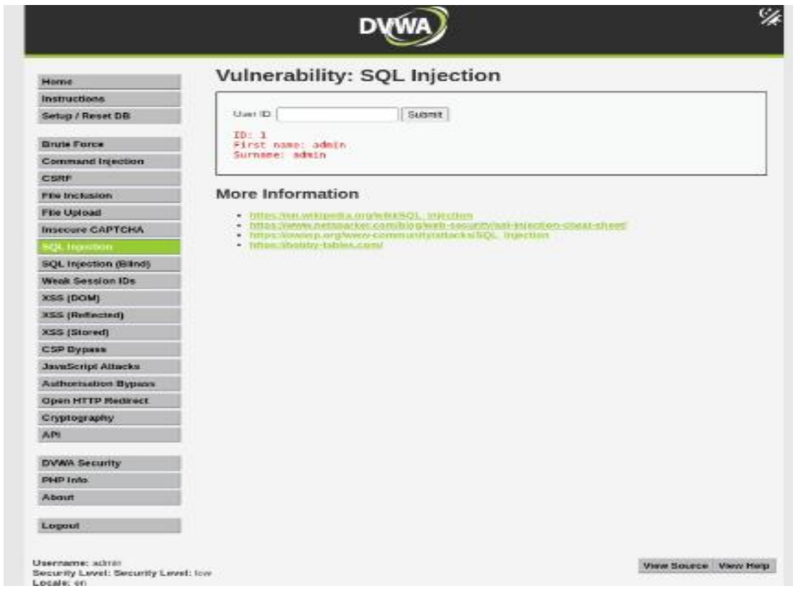
1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Low | Submit

Additional Tools

- View Broken Access Control Logs - View access logs for the Broken Access Control vulnerability

Username: admin
Security Level: Security Level: low
Locale: en



Vulnerability: SQL Injection

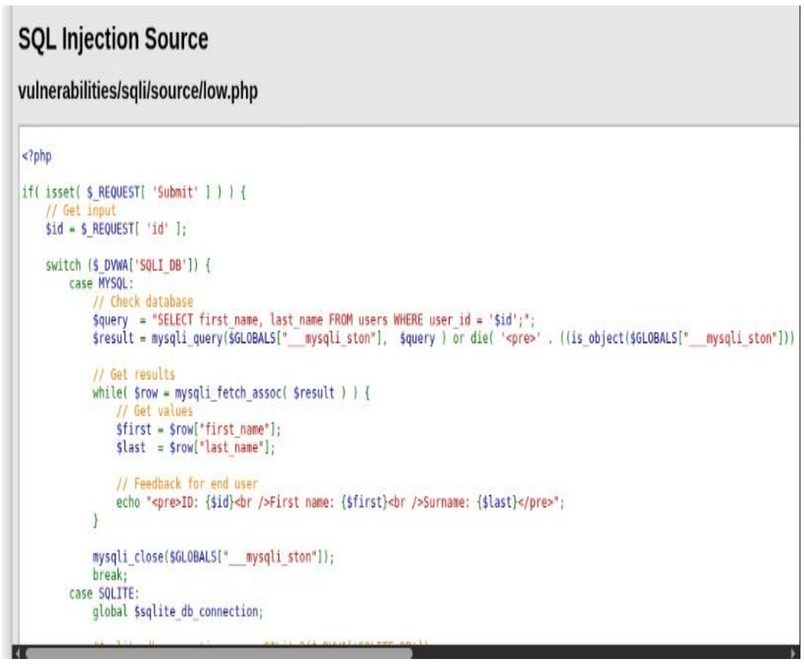
User ID: Submit

```
ID: 1
First name: admin
Surname: admin
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Username: admin
Security Level: Security Level: low
Locale: en



SQL Injection Source

vulnerabilities/sqli/source/low.php

```php
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"]))

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Get values
                $first = $row["first_name"];
                $last  = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
            }

            mysqli_close($GLOBALS["___mysqli_ston"]);
            break;
        case SQLITE:
            global $sqlite_db_connection;
```

References

1. OWASP – SQL Injection
2. OWASP Top 10 Web Application Security Risks
3. DVWA GitHub Repository
4. NIST Secure Coding Guideline