Jared Ricks, Jack Lin, Kolbe Williams-Wimmer

Part 1: Setting Up the Environment in Ubuntu:

Update the Ubuntu System:

The system was updated, but it restarted so we could not screenshot.

Install Apache (webserver), MySQL (or MariaDB), and PHP:

Installing Apache:

```
labuser1@ML-RefVm-535928:~$ sudo apt install apache2 -y
[sudo] password for labuser1:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Installing MySQL:

```
labuser1@ML-RefVm-535928:~$ sudo apt install mysql-server -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Installing PHP:

```
labuser1@ML-RefVm-535928:~$ sudo apt install php libapache2-mod-php php-mysql -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Download DVWA: Clone DVWA from GitHub:

```
labuser1@ML-RefVm-535928:/var/www/html$ sudo git clone https://github.com/digininja/DVWA.git
Cloning into 'DVWA'...
remote: Enumerating objects: 4857, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 4857 (delta 5), reused 9 (delta 3), pack-reused 4840 (from 1)
Receiving objects: 100% (4857/4857), 2.43 MiB | 7.31 MiB/s, done.
Resolving deltas: 100% (2342/2342), done.
labuser1@ML-RefVm-535928:/var/www/html$
```

Set Permissions: Set appropriate permissions for the DVWA folder:

```
labuser1@ML-RefVm-535928:/var/www/html$ sudo chown -R www-data:www-data /var/www/html/DVWA
labuser1@ML-RefVm-535928:/var/www/html$
```

This gives the Apache webserver permission to access the DVWA files.

Copy config.inc.php.dist into config.inc.php:

```
labuser1@ML-RefVm-535928:/var/www/html$ cd /var/www/html/DVWA/config/
labuser1@ML-RefVm-535928:/var/www/html/DVWA/config$ sudo cp config.inc.php.dist config.inc.php
labuser1@ML-RefVm-535928:/var/www/html/DVWA/config$
```

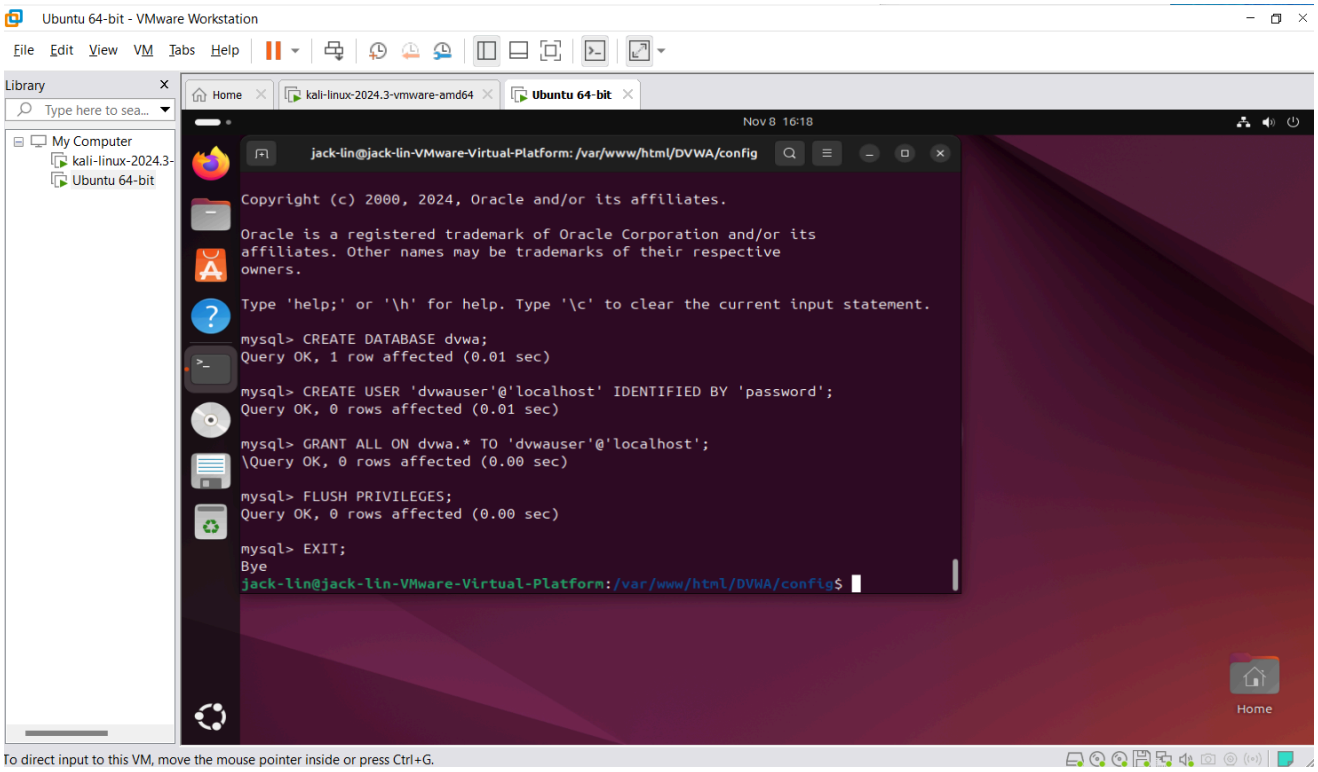
The source code is copied to the active configuration file.

Open the DVWA configuration file to set up database credentials:

Change these lines to match your MySQL/MariaDB setup:

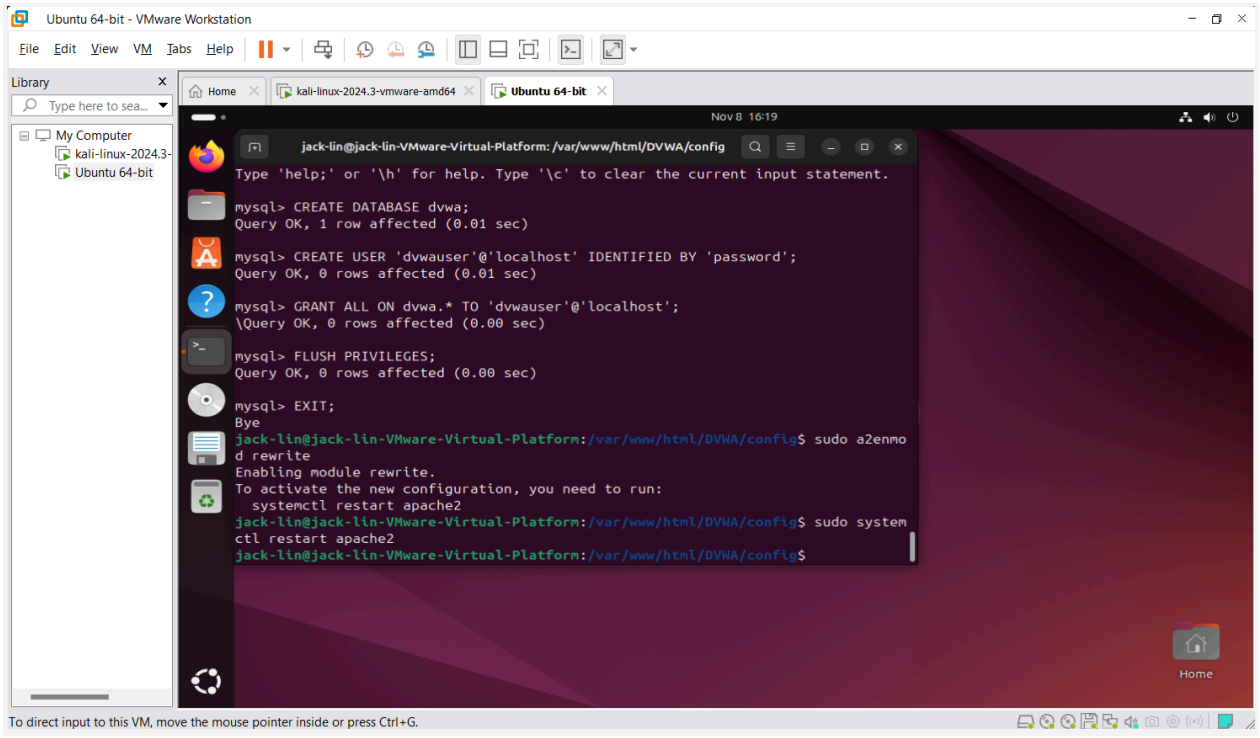
```
$_DVWA = array();
$_DVWA[ 'db_server' ]   = getenv('DB_SERVER') ?: '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'dvwauser';
$_DVWA[ 'db_password' ] = 'password';
$_DVWA[ 'db_port']      = '3306';
```

Create DVWA Database:
Open MySQL and create a database for DVWA:

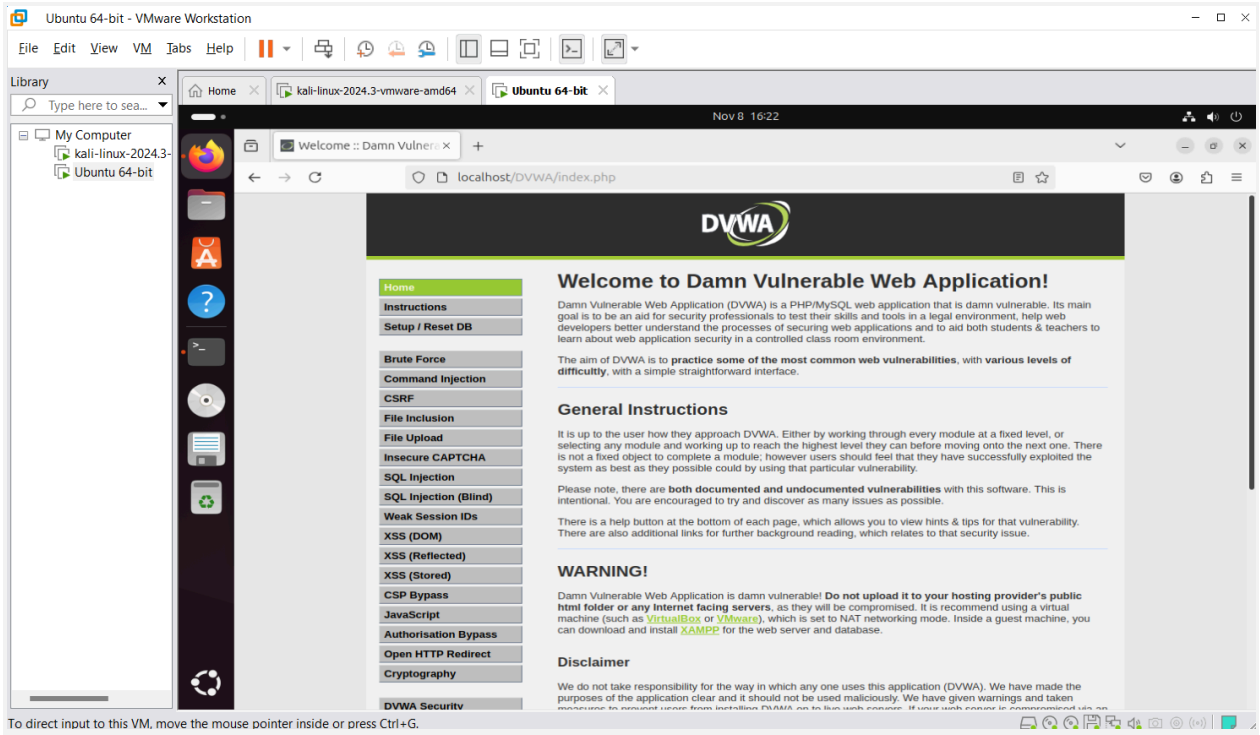


A database and a database user are created. All permissions are then granted to the database and the user and the changes are applied.

Enable mod_rewrite and Restart Apache:
This helps allow Apache to access DVWA.

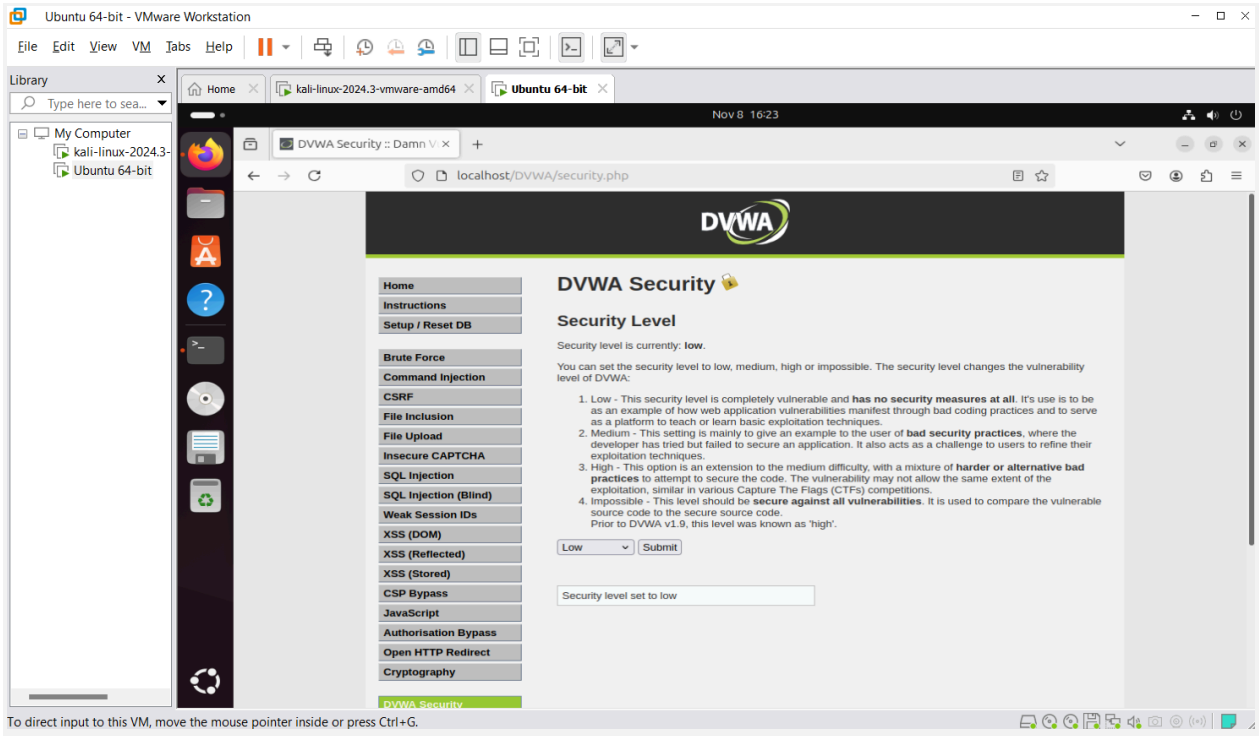


Set Up DVWA:
Access DVWA using Firefox:



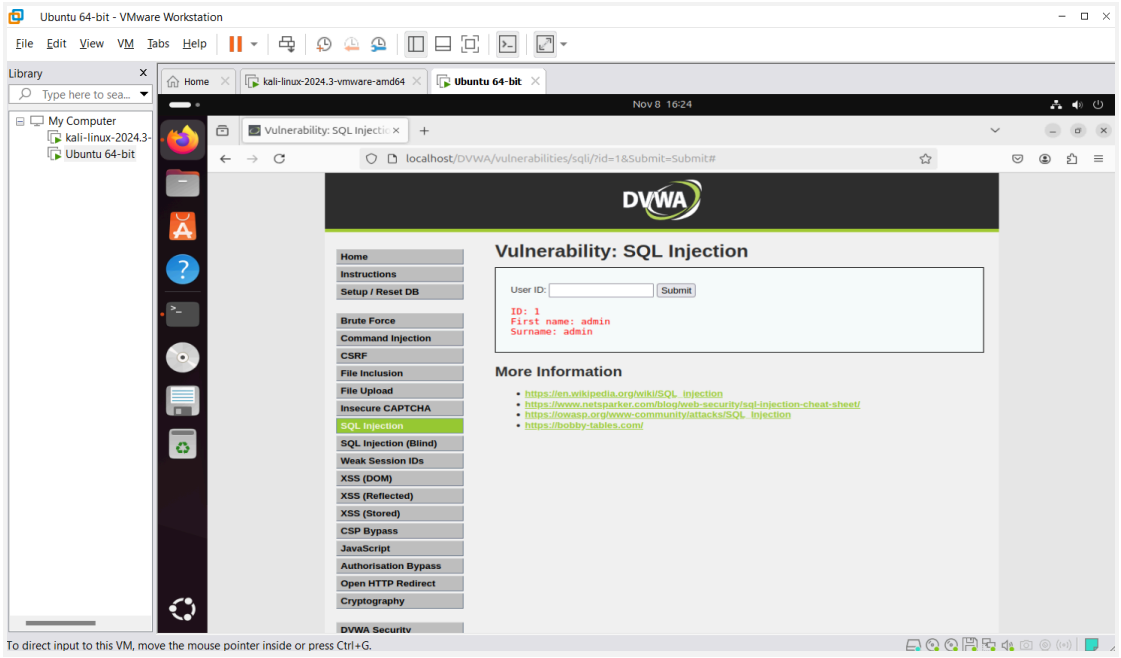
Part 2: Conducting an SQL Injection Attack

The security level in DVWA is set to low:

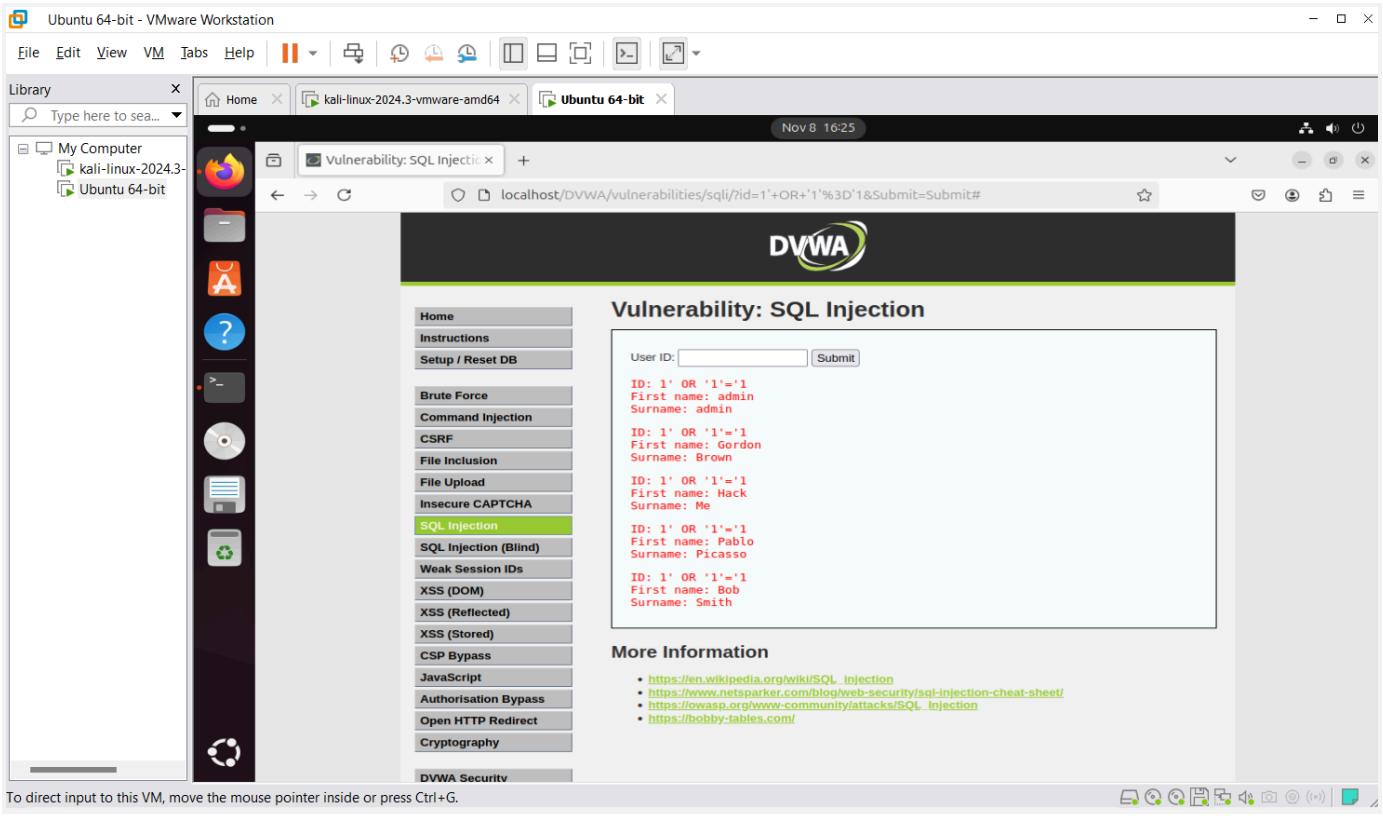


Step 2: Access SQL Injection Page

This result is given by clicking on SQL Injection in the left column and entering a User ID of 1. The expected result occurs, which gives information on the user with this ID number:



Step 3: Perform SQL Injection Attack
Now, we enter the payload of: `1' OR '1'='1`



All user information is displayed because the payload of 1 or 1 = 1 will always be evaluated as true. Since there was no password parameter, the payload did not even need to comment anything out.

Part 3: Understanding SQL Injection Attack
When we entered
1' OR '1'='1, it modified the SQL query to:

SELECT * FROM users WHERE id = '1' OR '1'='1';

The OR '1'='1' clause always evaluates to true, returning all rows in the database instead of just one specific user, indicating a successful SQL injection attack.

Part 4: SQL Injection Mitigation Techniques
Here are some commonly used techniques to mitigate SQL injection vulnerabilities:

1. Parameterized Queries:
    - Use prepared statements and parameterized queries to separate data from code.
    - This method prevents user input from being executed as SQL code.
2. Escaping User Input:

- Escaping special characters in SQL queries can reduce the risk of SQL Injection.
- This method should only be used in combination with other techniques.

3. Input Validation:
    - Validate and sanitize user inputs. For instance, ensure numerical fields contain only numbers.
4. Using ORM or Stored Procedures:
    - Using Object-Relational Mapping (ORM) libraries or stored procedures helps prevent SQL injection by abstracting SQL queries.

Part 5: Demonstrating SQL Injection Mitigation

1. Example Code for a Parameterized Query (PHP):

- In a real-world scenario, code like the following would replace vulnerable queries:

```
$stmt = $conn->prepare("SELECT * FROM users WHERE id = ?");
$stmt->bind_param("i", $id);
$stmt->execute();
$result = $stmt->get_result();
```

- This code binds user input ($id) as a parameter, which prevents the input from being interpreted as part of the SQL code.

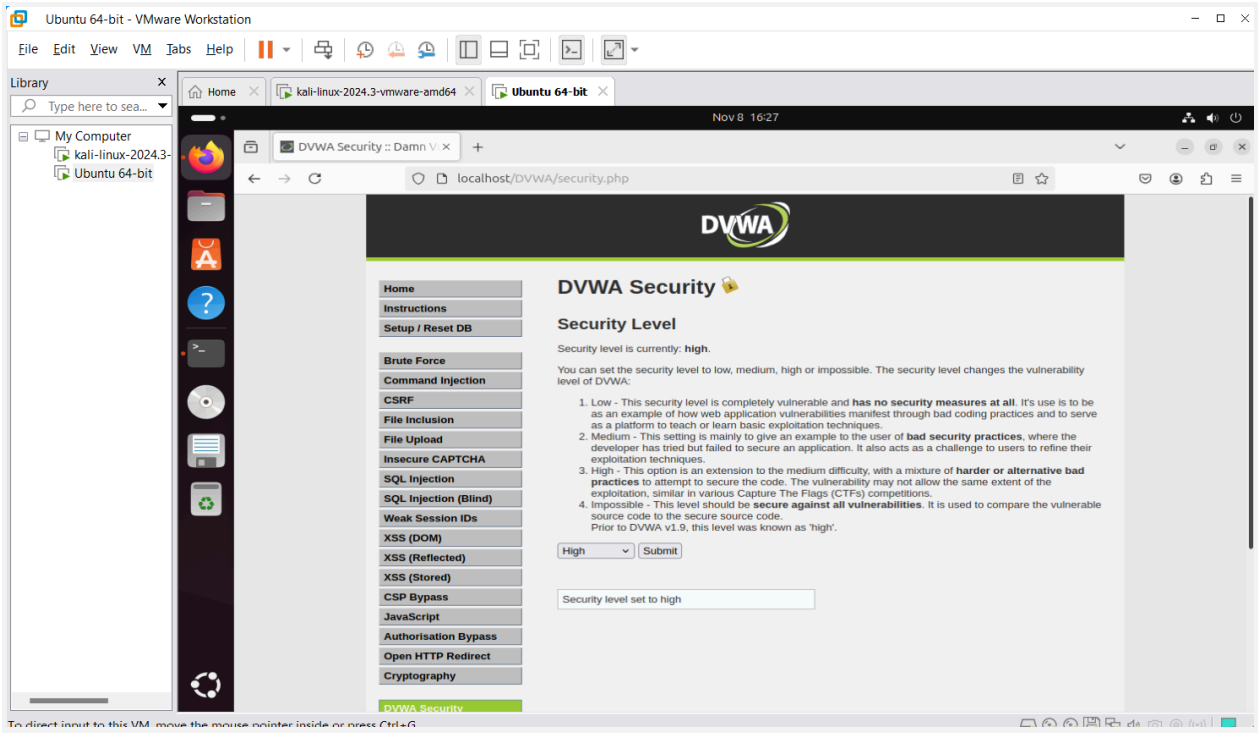
2. Testing the Mitigation:

- In a fully implemented environment, if you attempt the SQL injection payload (1' OR '1'='1), it will be treated as a string and not executed as SQL code.
- Since this is a hypothetical scenario, the DVWA app itself would need to be rewritten to reflect this approach.

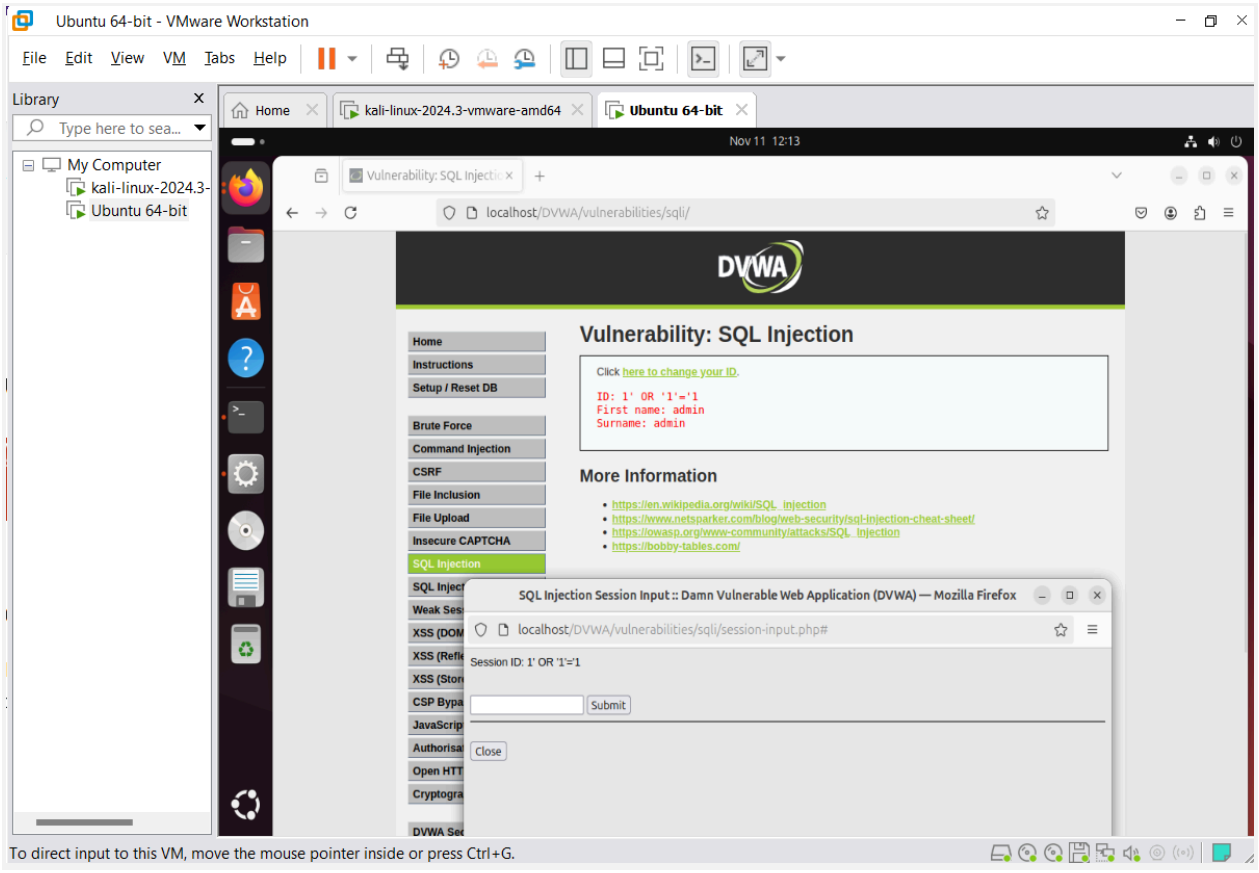
3. Setting DVWA Security to High:

- Go to DVWA Security settings and set the security level to High.
- Now, try the same SQL injection payload. The app should prevent this attack because of more secure code practices.

Now, the security level in DVWA is set to high:

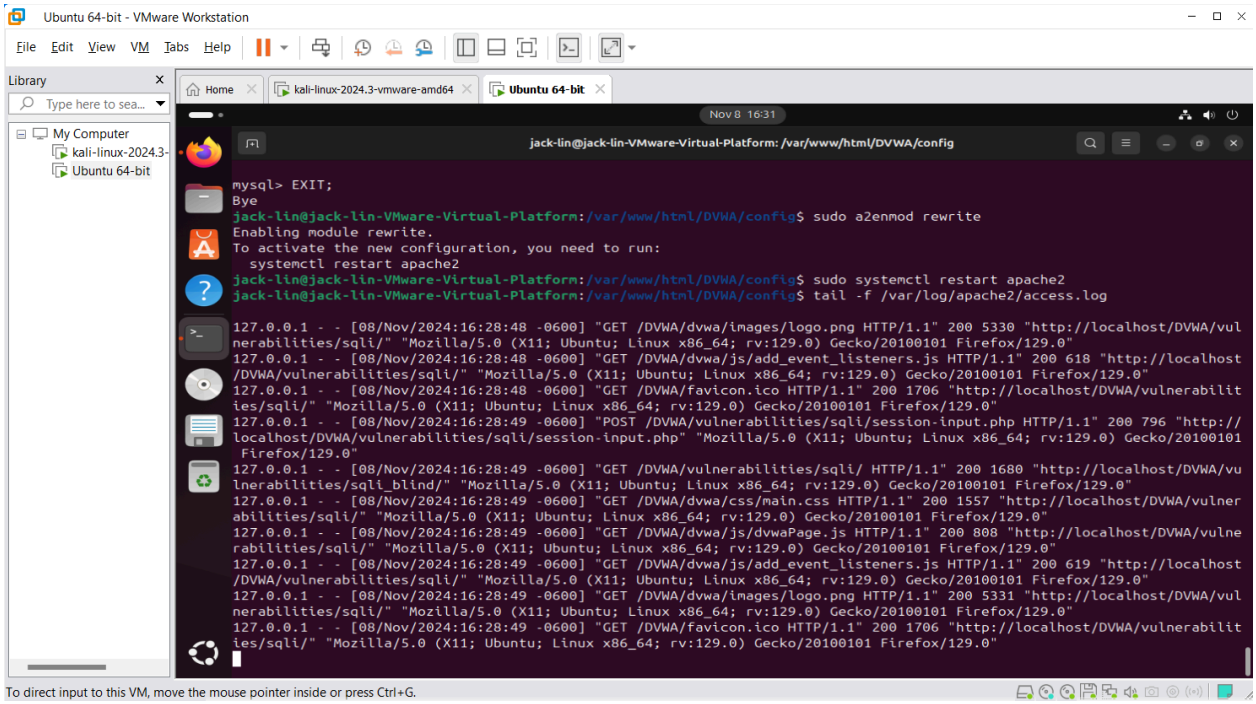


The same payload of `1' OR '1'='1` only gives the result for one user:

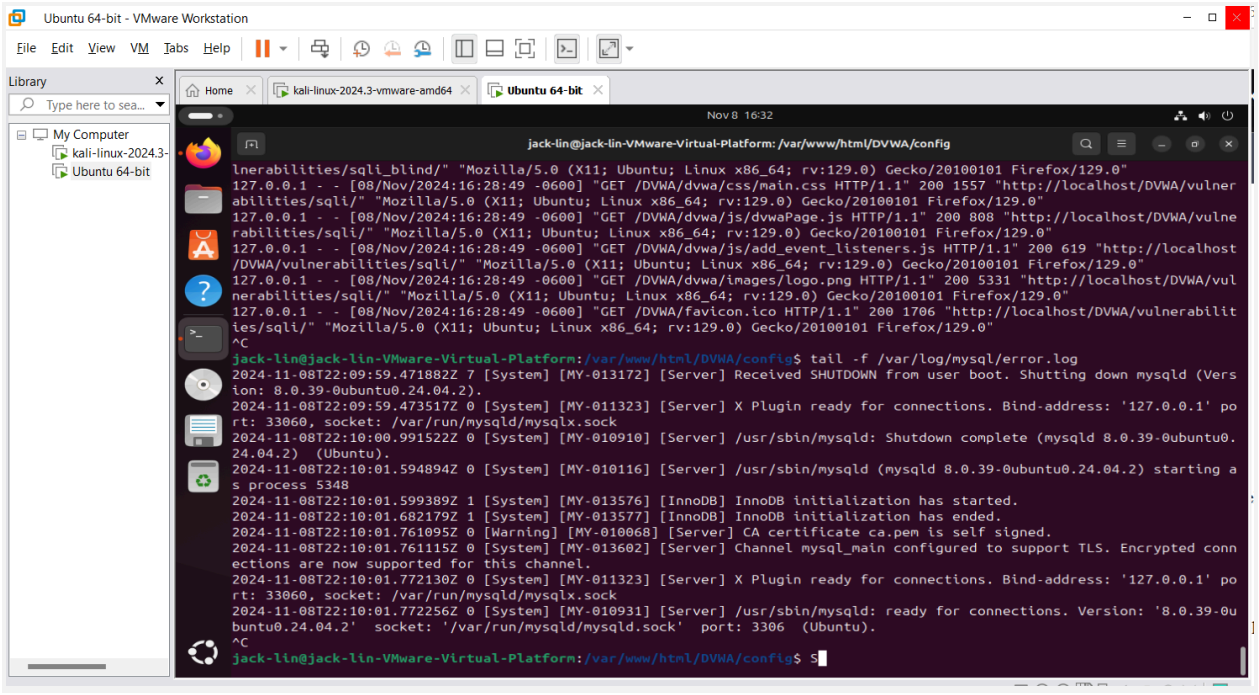


All entries are not shown. Instead, only the user information for ID 1 is shown because, in the payload, 1 was the first character. This security level uses better coding practices, such as parameterized queries and input sanitization. Thus, the entry for the given ID was displayed as it was supposed to, not the entire list.

Part 6: Securing DVWA and Reviewing Logs

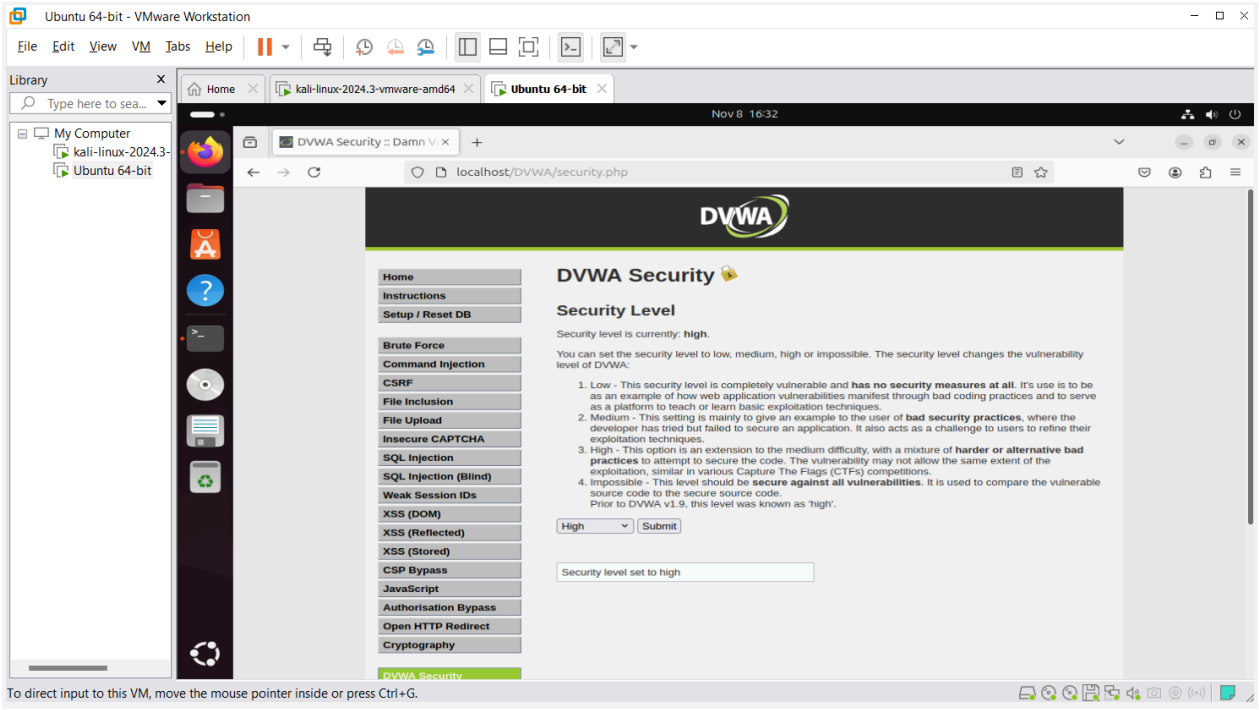


These are the most recent entries in the Apache access log, which show the actions we took using DVWA on the Apache server just before running this command. This log could be used for troubleshooting or security monitoring on the Apache server.



These are the most recent entries in the error log for My SQL on our virtual machine. This could be used to view errors during connection and operation or by another user.

2. Disable DVWA or Set the Security Level to High:



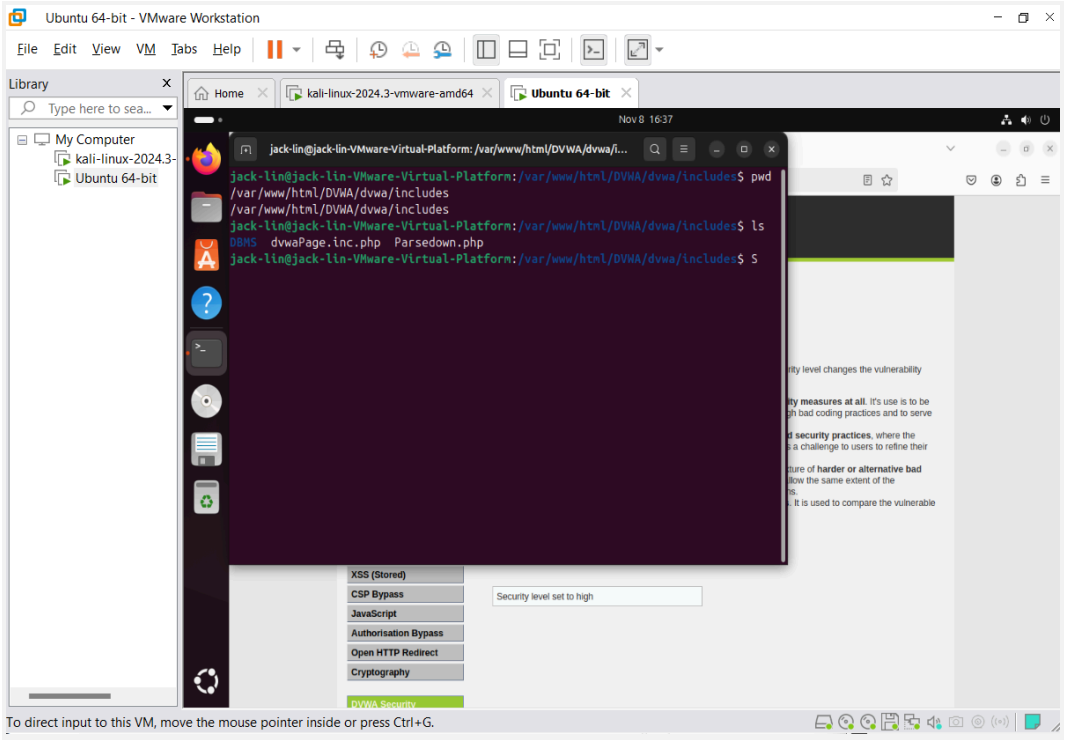
The security level in DVWA is now set to high.

Part 7: Exploring the source code of DVWA

Step 1: Locate the DVWA Directory

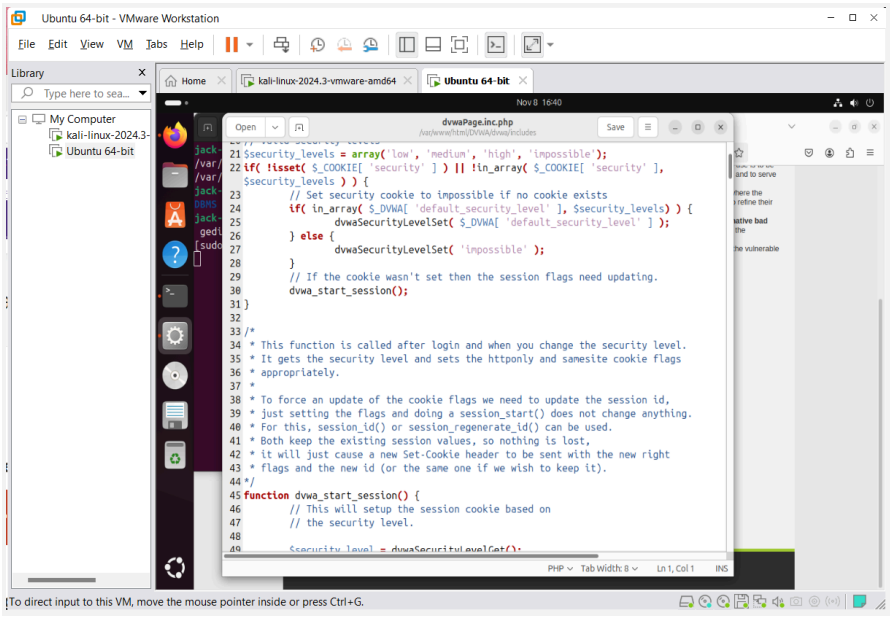
Step 2: Find the Security Level File

After changing to the correct directory (`/var/www/html/DVWA`), we can see the file that controls the security level: `dvwa/includes/dvwaPage.inc.php`

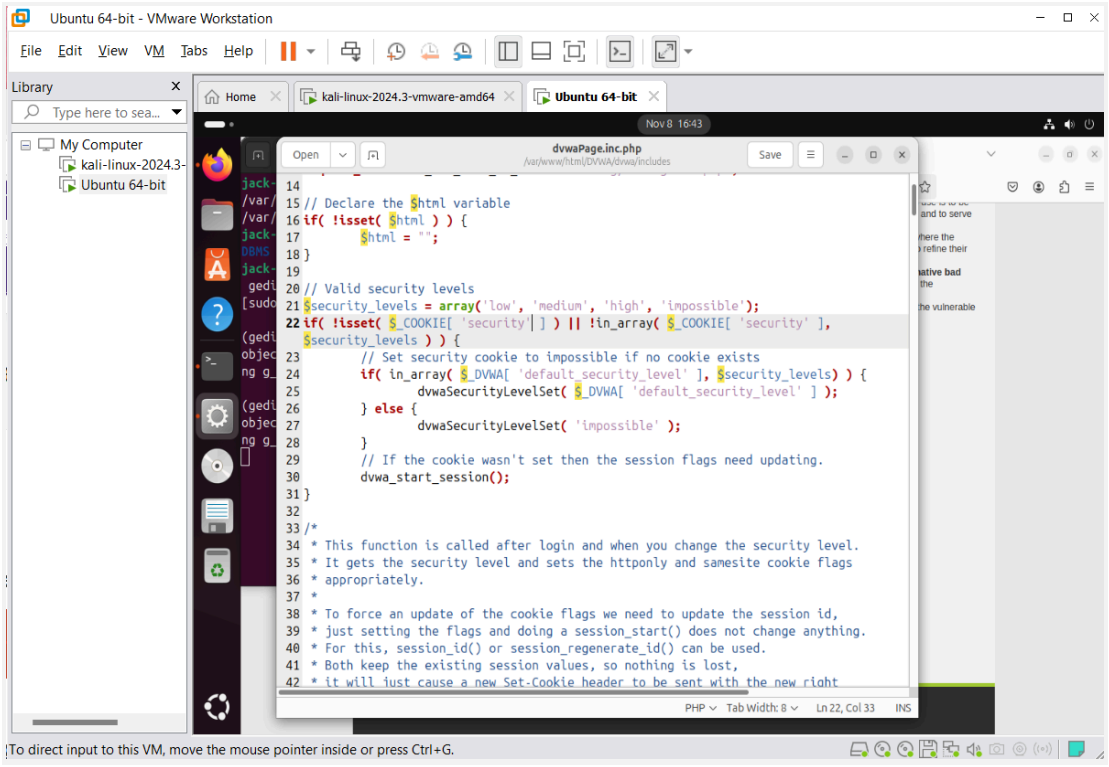


Step 3: Check dvwaPage.inc.php for Security Level Code

Open the File:



Search for the Security Level Logic: Look for the following lines or similar code within this file:
$security_level = isset($_SESSION['security'])? $_SESSION['security'] : 'impossible';

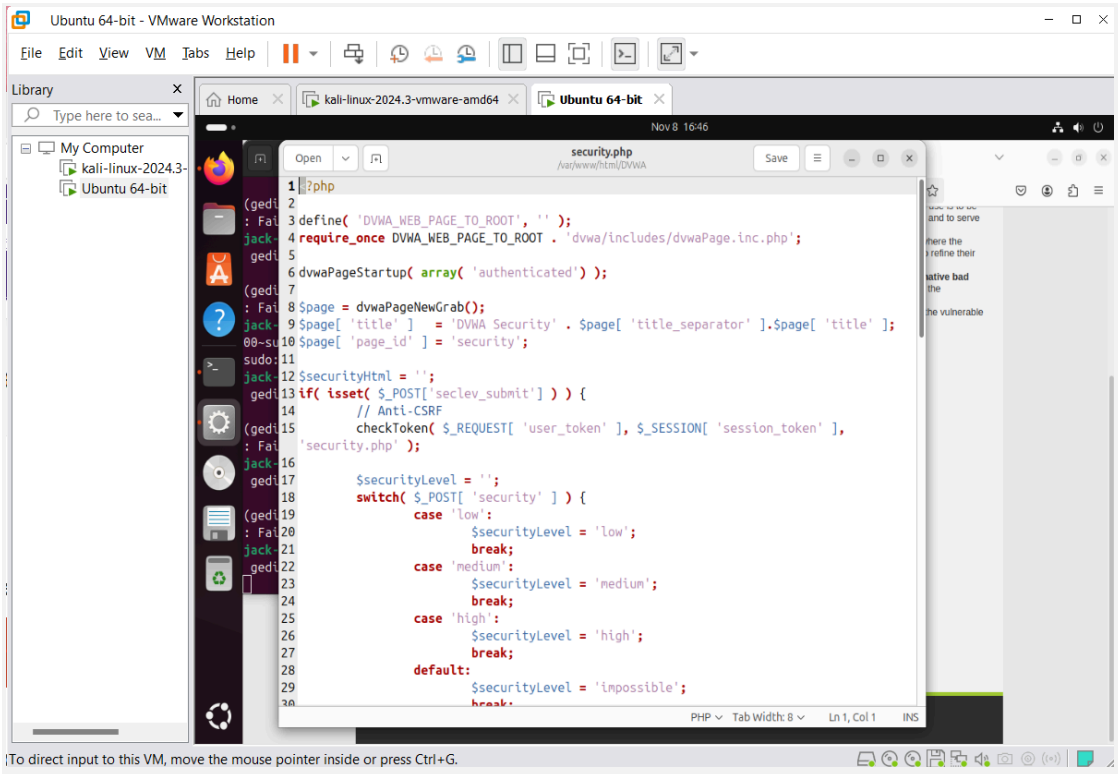


This code sets the default security level to impossible if the security level is not otherwise set.

Identify the Security Levels:
DVWA defines several security levels (low, medium, high, impossible) that are stored in PHP sessions. Changing the security level in the web interface updates this session variable.

Step 4: Explore security.php for Security Level Setting

Open security.php:



This code determines the security level for the session.

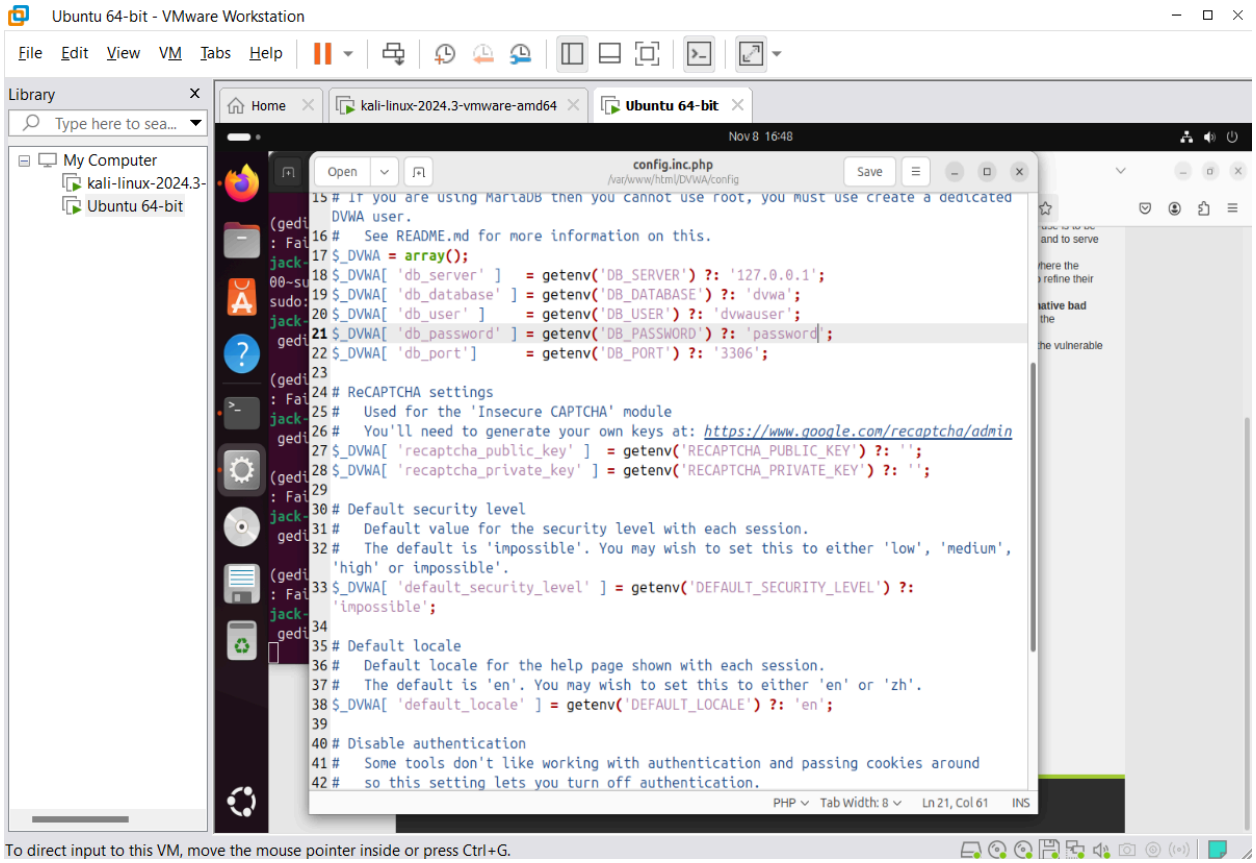
Examine the Code: Look for the form handler code that saves the selected security level to the session:

```
if(isset($_POST['security'] ) ) {
        $_SESSION['security']= $_POST['security'];
        header('Location: ' . $_SERVER['SCRIPT_NAME'] );
        exit;
}
```

This code checks if the form to change the security level has been submitted. If so, it updates the session variable $_SESSION['security'] with the new value selected by the user (low, medium, high, or impossible).

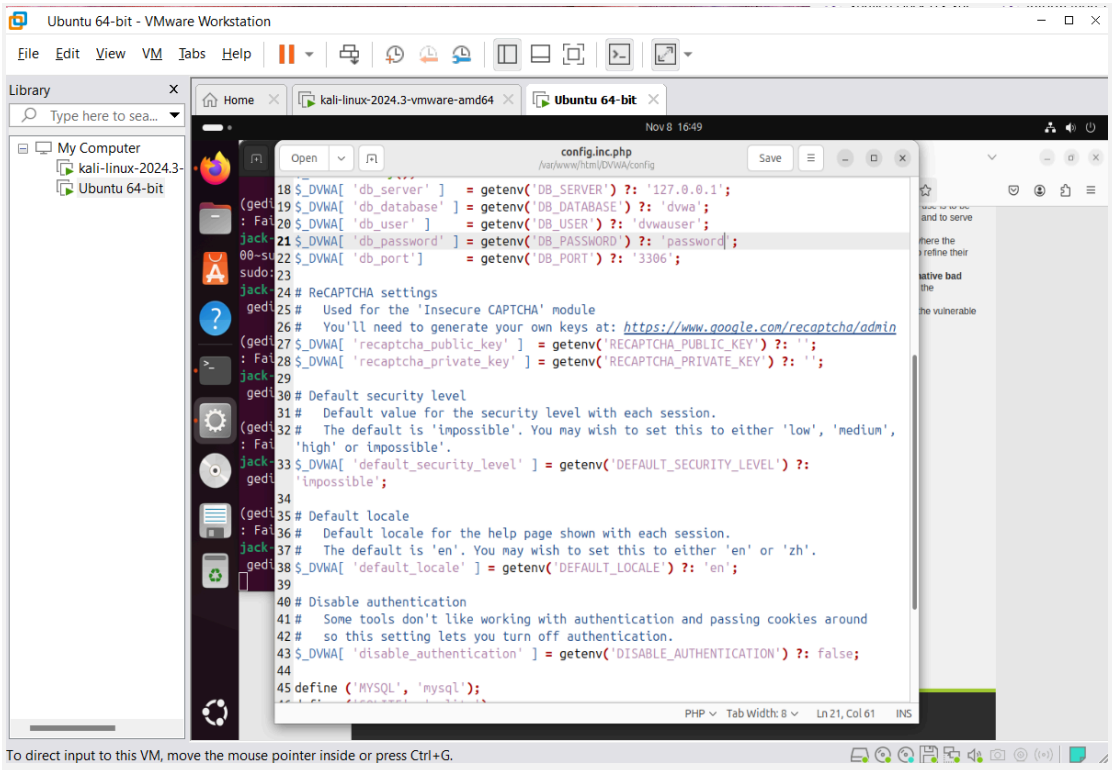
Step 5: Review config.inc.php for Default Security Level

Open the Config File:



Our configuration file looks slightly different than it did before because these settings worked better for the virtual machine we were using.

Look for Security Level:
In this file, you may find a line that sets the default security level:



Our default security level is currently set to impossible but can be easily changed.

Conclusions:

DVWA is an intentionally vulnerable web application that allows users to explore different types of attacks. In this project, we explored SQL injection, a vulnerability in weak web applications where user input is executed as part of a query and can be manipulated to display restricted database information. SQL injection can be prevented with secure coding practices, such as using parameterized queries so that user input is not executed as code, escaping user input so that users cannot input executable scripts, input validation to ensure users are only entering necessary input, and by using ORM or stored procedures that ensure SQL queries are abstracted. This project shows an example of an application with low security being vulnerable to SQL injection and how SQL injection can be prevented in that same application along with different logs and configuration files. The security level can also be easily adjusted in the configuration files for DVWA specifically.