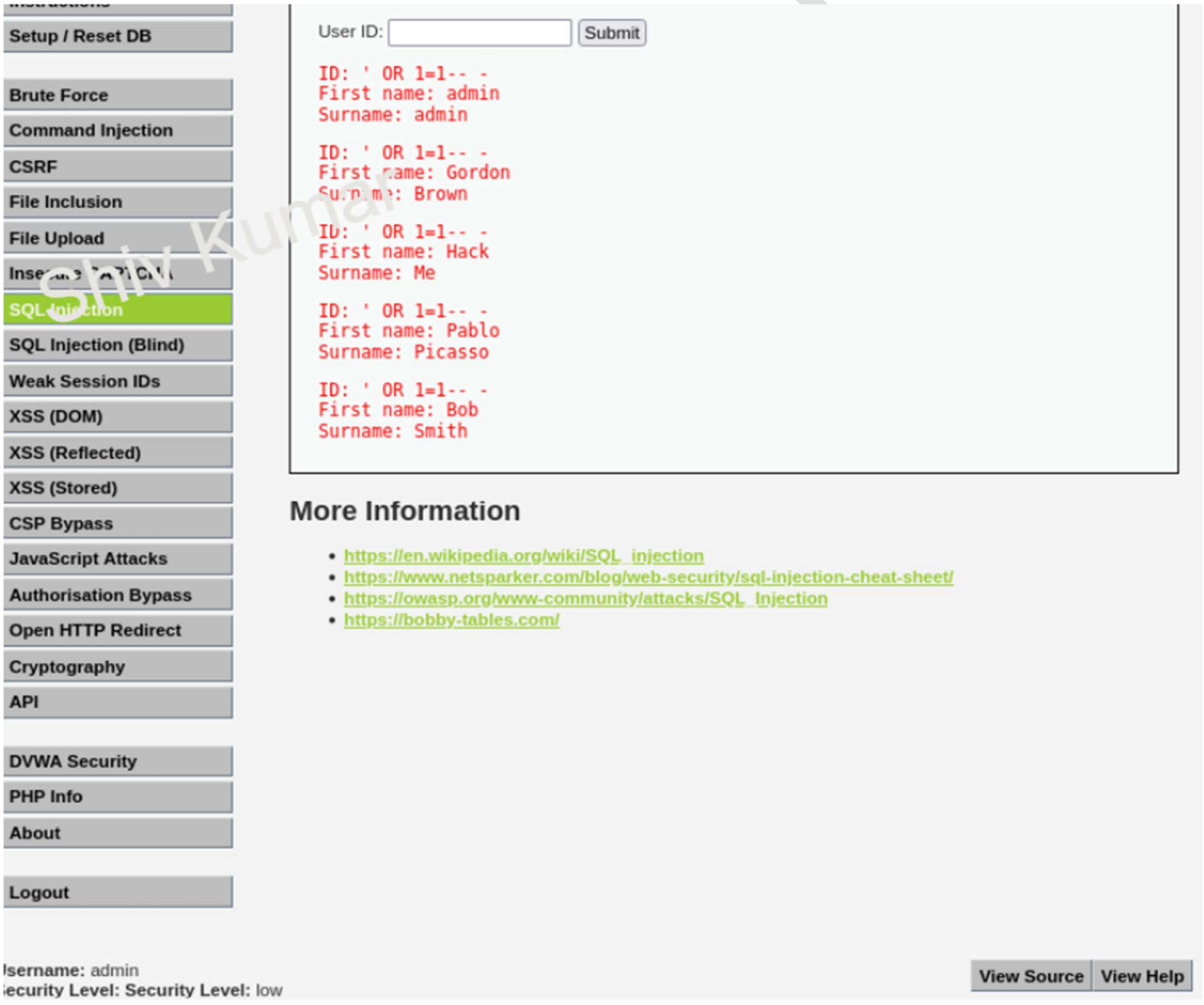
Project 1 : DVWA Security Level Comparison Project

Objective: The object of project is to Compare how DVWA behaves at different security levels (Low, Medium, High, Impossible)

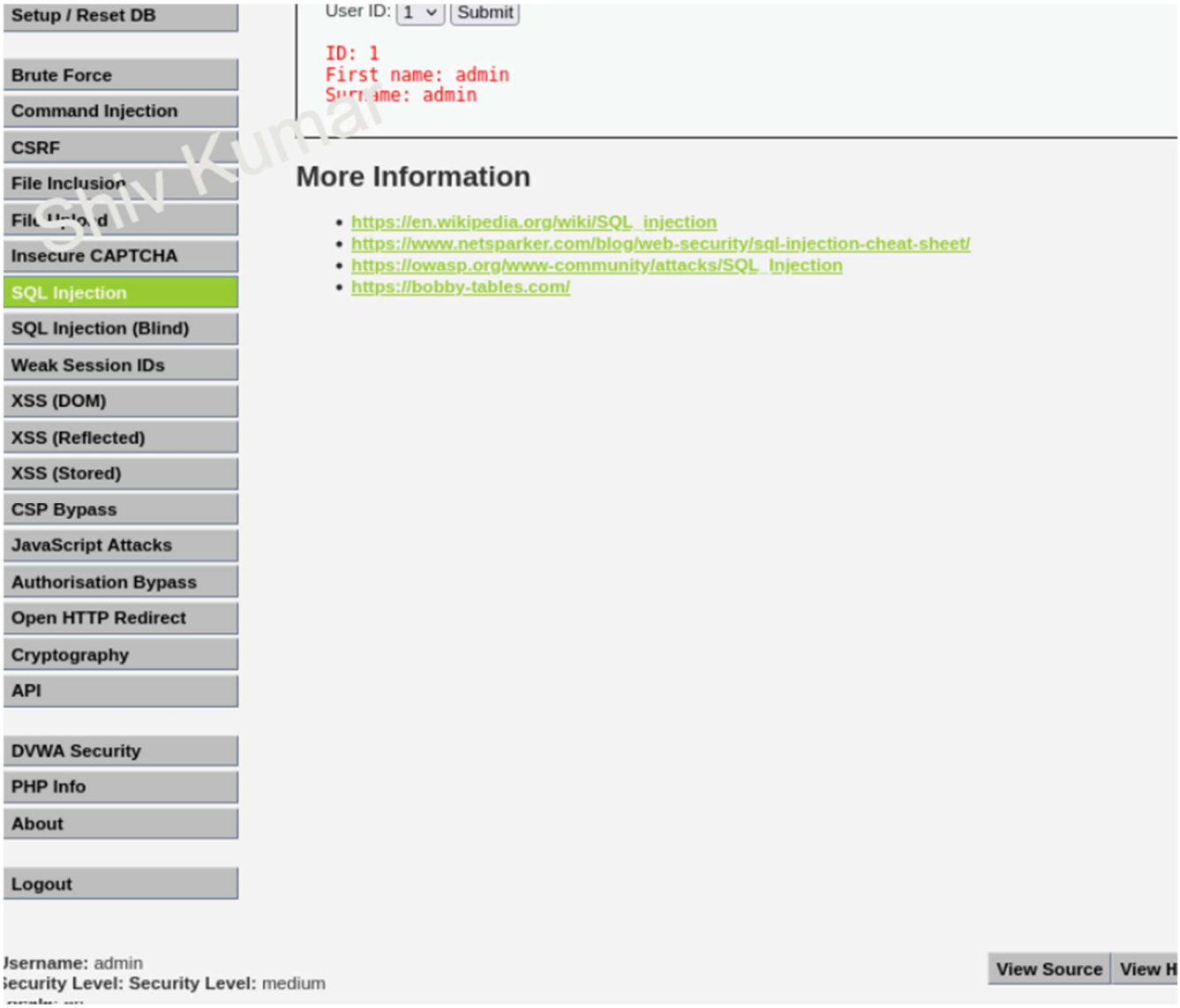
Tasks:We have picked 2 vulnerabilities

Sql injections-When user input (e.g., from a login form or URL parameter) is directly concatenated into an SQL query, attackers may inject malicious SQL code that the database executes.

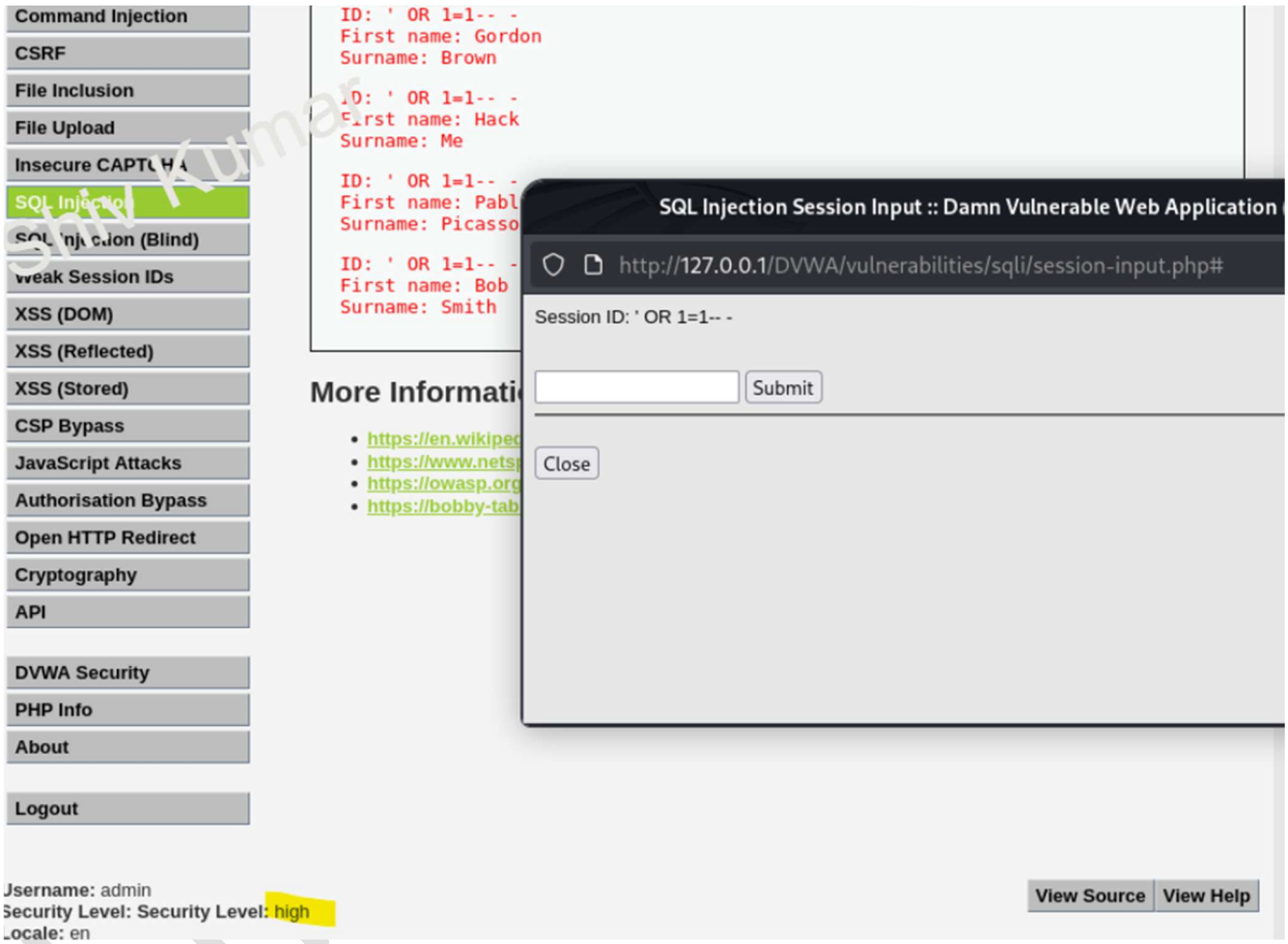
Using payload:- 'OR 1=1-- - with low Security



Using payload: - ‘OR 1=1-- - with medium Security

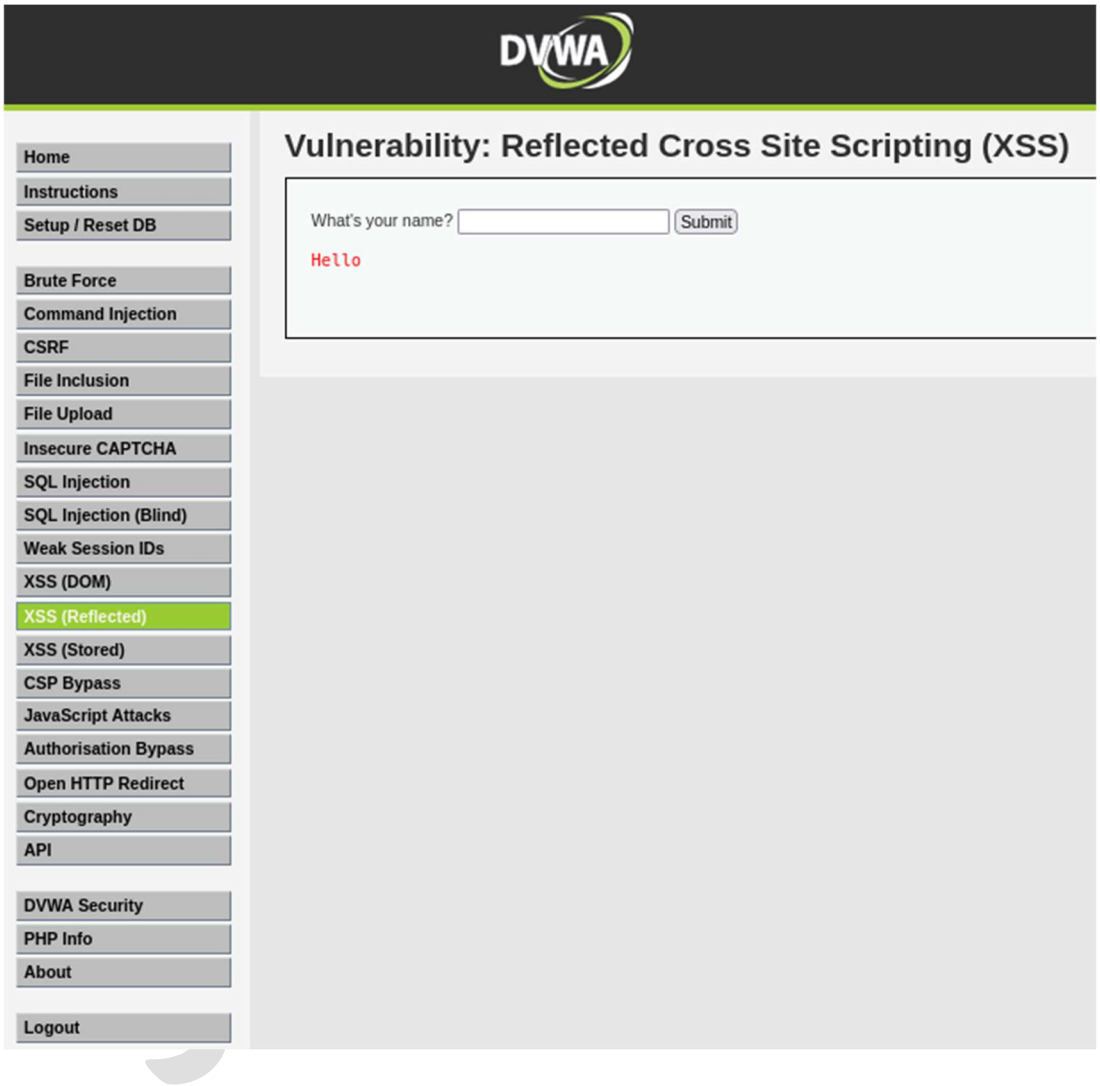


Using payload: - 'OR 1=1-- - with High Security

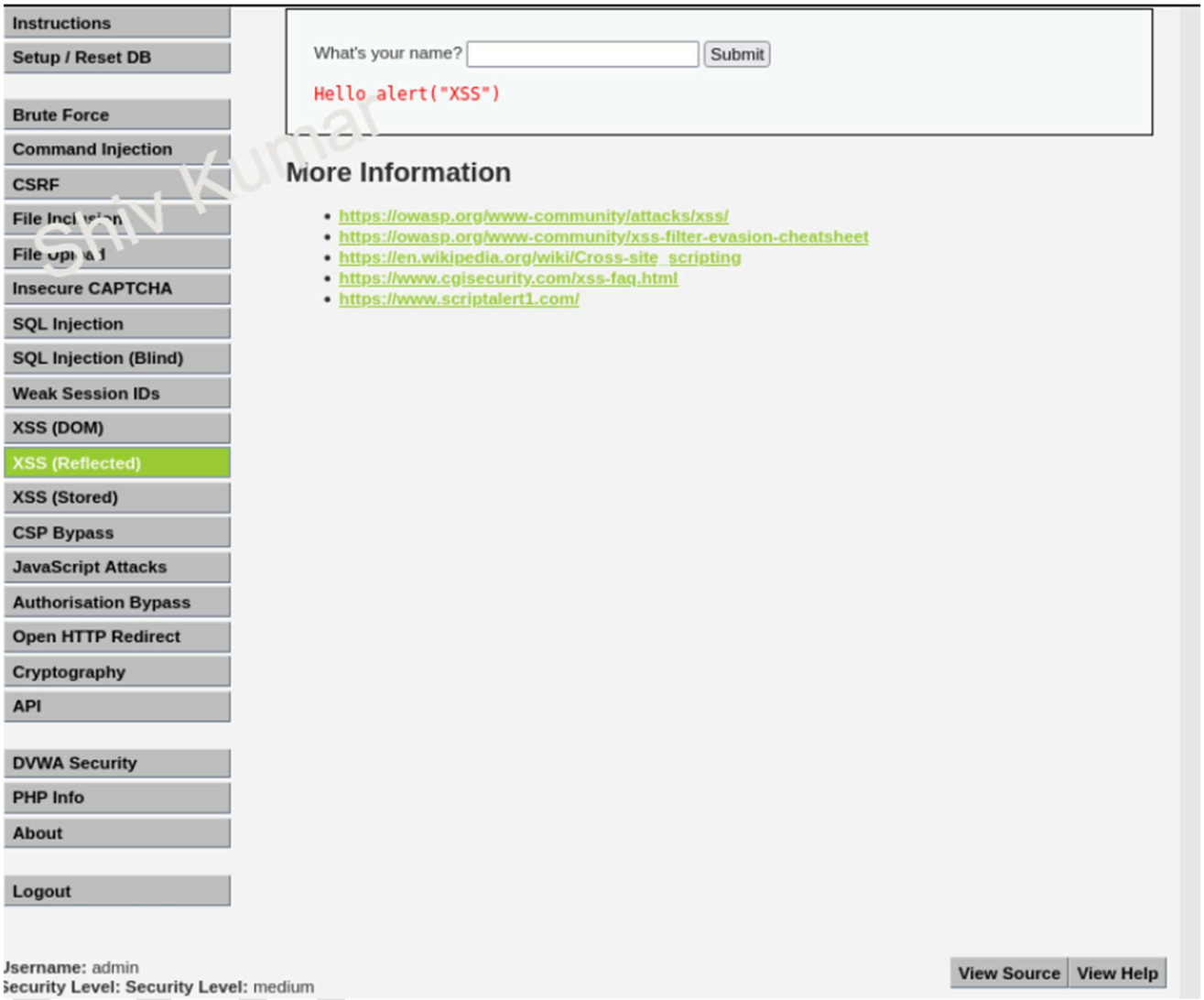


Cross-site scripting: - Cross-Site Scripting (XSS) is a web security vulnerability that occurs when an application allows untrusted input to be included in web pages without proper validation or output encoding. This enables attackers to inject malicious scripts that are executed in the victim's browser.

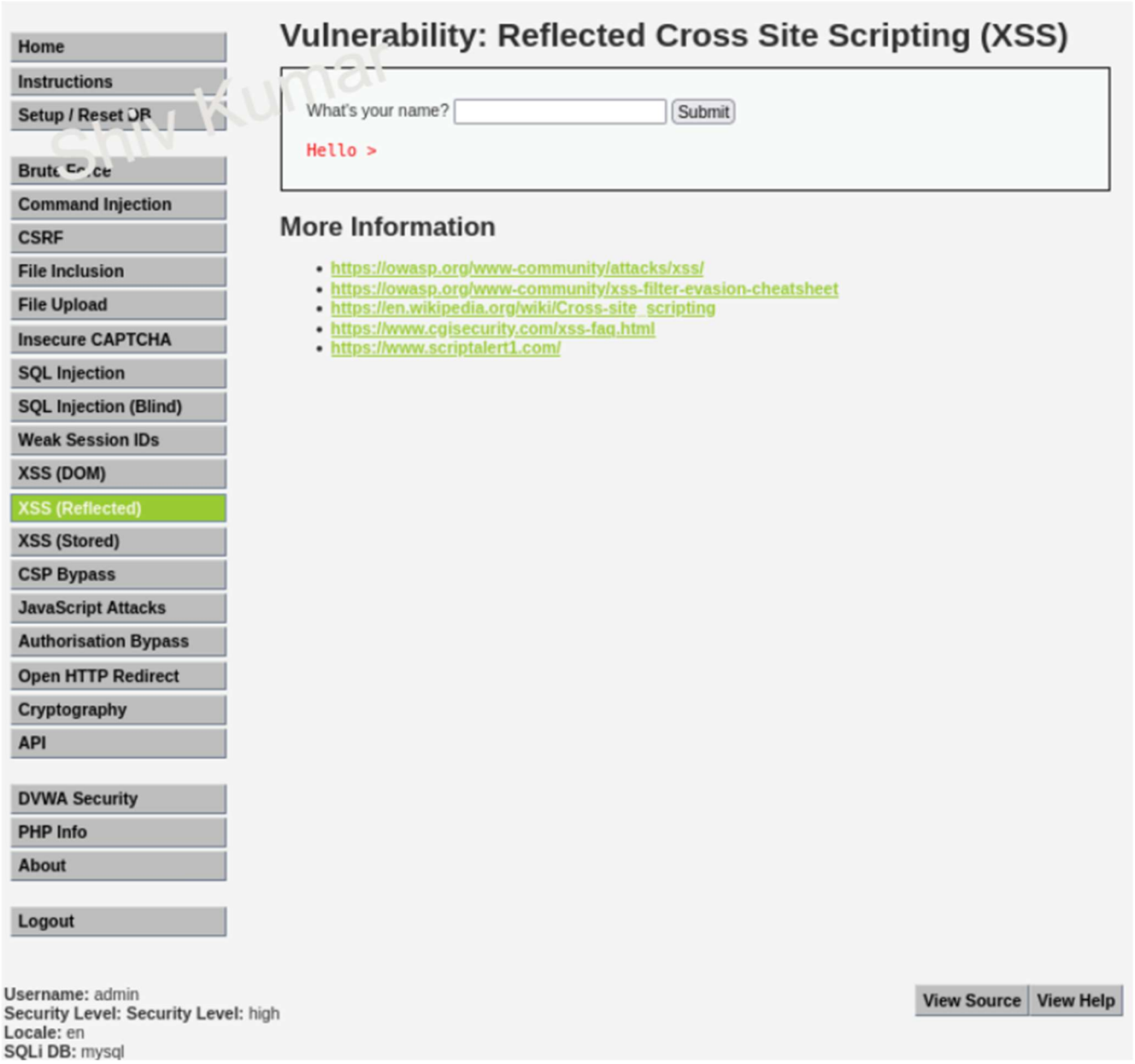
Using payload: - <script>alert("XSS")</script> with low Security



Using payload: - <script>alert(“XSS”)</script> with medium Security



Using payload: - <script>alert(“XSS”)</script> with High Security



Outcome of above project task: -

This project provided valuable insight into how web vulnerabilities appear in real-world applications and how they can be mitigated through secure coding practices. Testing identical payloads across multiple security levels demonstrated that effective security does not rely on a single solution, but rather on the implementation of layered controls.

DVWA proved to be a useful environment for gaining hands-on experience with both attack techniques and defensive measures. Through this comparison, practical understanding of SQL Injection, Cross-Site Scripting (XSS), and secure application development was significantly enhanced.

Project 2: Mini Pentest Project: DVWA + Another App

Objective: Is to Perform a controlled mini penetration test

Tasks: Set up DVWA and one more vulnerable app (Juice Shop, Mutillidae, bWAPP).

Vulnerable app Setup done- for bWAPP

Perform recon scanning using nmap on bWapp ip

```
┌──(shiv㉿kali)-[~]
└─$ nmap -sV 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-17 20:00 IST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000090s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.7 ((Ubuntu))

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.57 seconds
```

Using Directory enumeration using DIRB

http://127.0.0.1:42001

```
┌──(shiv㉿kali)-[~]
└─$ dirb http://127.0.0.1:42001

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Wed Dec 17 21:07:42 2025
URL_BASE: http://127.0.0.1:42001/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://127.0.0.1:42001/ ----

(!) FATAL: Too many errors connecting to host
    (Possible cause: COULDNT CONNECT)

-----------------
END_TIME: Wed Dec 17 21:07:42 2025
DOWNLOADED: 0 - FOUND: 0
```

Using nikto -h 127.0.0.1

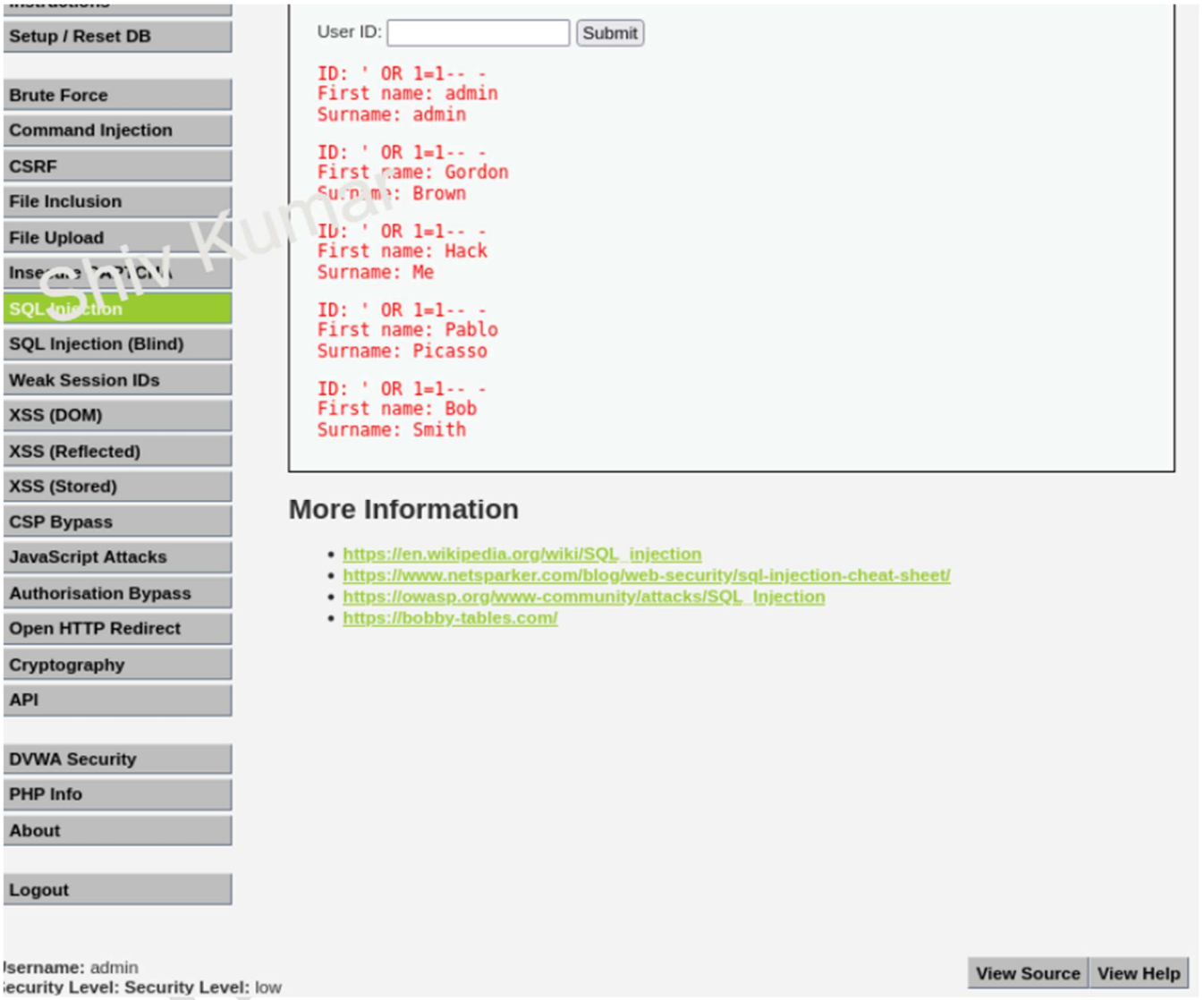
```
┌──(shiv㉿kali)-[~]
└─$ nikto -h 127.0.0.1
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    127.0.0.1
+ Target Port:        80
+ Start Time:         2025-12-17 21:15:59 (GMT5.5)
---------------------------------------------------------------------------
+ Server: Apache/2.4.7 (Ubuntu)
+ /: Retrieved x-powered-by header: PHP/5.5.9-1ubuntu4.14.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type.
See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ Root page / redirects to: portal.php
+ /passwords/: Directory indexing found.
+ /robots.txt: Entry '/passwords/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00600600_robots-txt-file
+ /documents/: Directory indexing found.
+ /robots.txt: Entry '/documents/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00600600_robots-txt-file
+ /images/: Directory indexing found.
+ /robots.txt: Entry '/images/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00600600_robots-txt-file
+ /robots.txt: Entry '/admin/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00600600_robots-txt-file
+ /robots.txt: contains 5 entries which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary/Robots.txt
+ Apache/2.4.7 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ /login.php: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /web.config: ASP config file is accessible.
+ ///etc/hosts: The server install allows reading of any system file by adding an extra '/' to the URL.
```

```
+ /phpinfo.php: Output from the phpinfo() function was found.
+ /admin/: This might be interesting.
+ /apps/: Directory indexing found.
+ /apps/: This might be interesting.
+ /db/: Directory indexing found.
+ /db/: This might be interesting.
+ /passwords/: This might be interesting.
+ /stylesheets/: Directory indexing found.
+ /stylesheets/: This might be interesting.
+ /admin/index.php: This might be interesting: has been seen in web logs from an unknown scanner.
+ /phpinfo.php: PHP is installed, and a test script which runs phpinfo() was found. This gives a lot of system information. See: CWE-552
+ /admin/phpinfo.php: Output from the phpinfo() function was found.
+ /admin/phpinfo.php: Immobilier allows phpinfo() to be run. See: https://vulners.com/osvdb/OSVDB:35877
+ /config.inc: DotBr 0.1 configuration file includes usernames and passwords. See: OSVDB-5092
+ /update.php: Cookie admin created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /update.php: Cookie movie_genre created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /update.php: Cookie secret created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /update.php: Cookie top_security created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /update.php: Cookie top_security_nossl created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /update.php: Cookie top_security_ssl created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /install.php: install.php file found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /login.php: Admin login page/section found.
+ /test.php: This might be interesting.
+ /wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wordpress/wp-content/themes/twentyeleven/images/headers/server.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wordpress/wp-includes/Requests/Utility/content-post.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /wordpress/wp-includes/js/tinymce/themes/modern/Meuhy.php?filesrc=/etc/hosts: A PHP backdoor file manager was found.
+ /assets/mobirise/css/meta.php?filesrc=: A PHP backdoor file manager was found.
+ /login.cgi?cli=aa%20aa%27cat%20/etc/hosts: Some D-Link router remote command execution.
+ /shell?cat+/etc/hosts: A backdoor was identified.
+ 8885 requests: 0 error(s) and 49 item(s) reported on remote host
+ End Time:           2025-12-17 21:16:29 (GMT5.5) (30 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

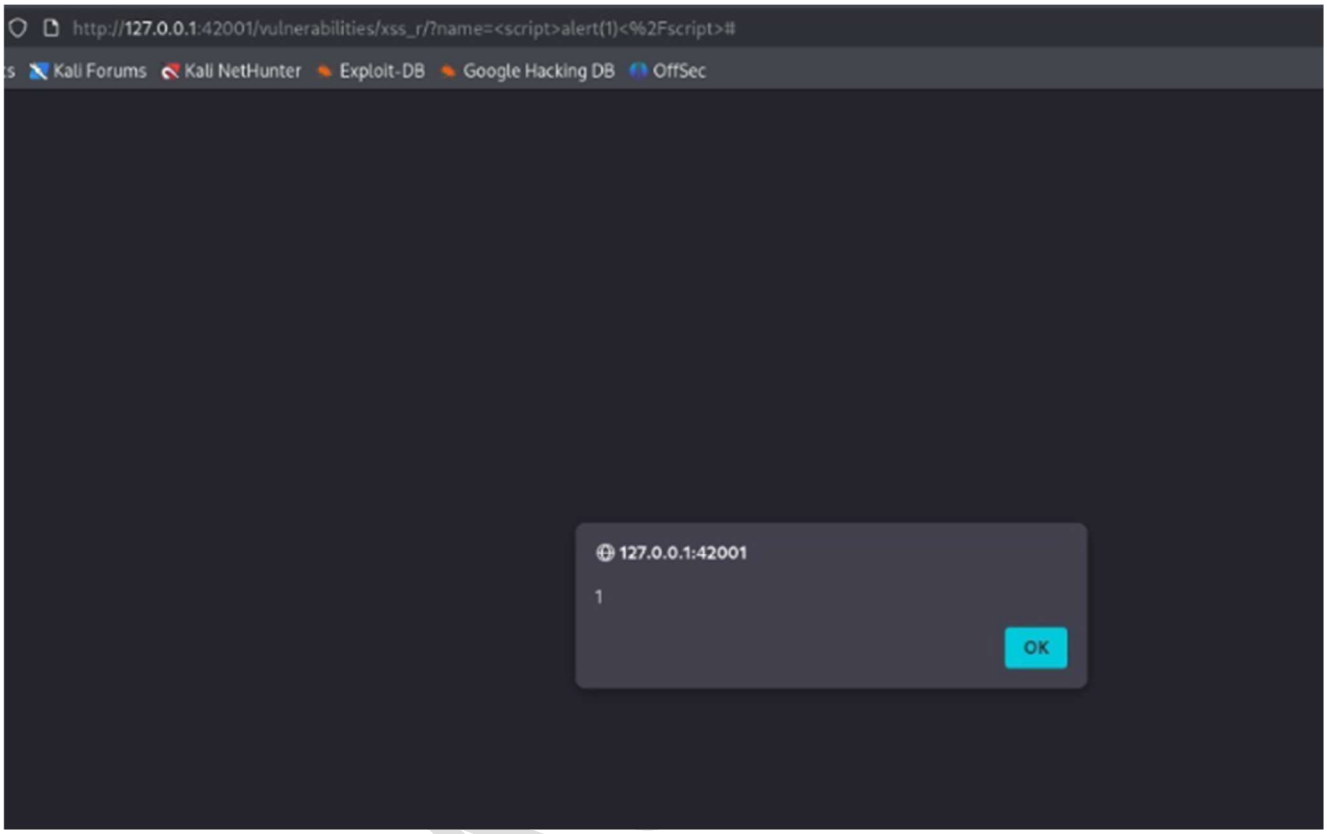
Finding of Vulnerability using DVWA

SQL Injection

Using payload: - 'OR 1=1-- - with low Security



Cross-Site Scripting
<script>alert(“1”)</script>



Executive Summary

This mini penetration testing exercise evaluated two deliberately vulnerable web applications, DVWA and bWAPP, within a controlled laboratory environment. The assessment focused on identifying common and critical web application vulnerabilities, including SQL Injection and Cross-Site Scripting (XSS).

Testing confirmed the presence of Cross-Site Scripting vulnerabilities in DVWA, demonstrating the security risks associated with improper input validation and output handling. These findings highlight how easily client-side attacks can be introduced when secure coding controls are insufficient.

For bWAPP, several high-risk security weaknesses were discovered during reconnaissance and vulnerability scanning activities. These included unnecessary service exposure and the use of outdated software components. Overall, the results emphasize the need for strong secure development practices, system hardening, and continuous security testing to minimize potential attack vectors.

Remediations

- Utilize parameterized queries and prepared statements to ensure safe interaction with databases
- Enforce strict validation and sanitization for all user-provided input
- Apply appropriate output encoding techniques to mitigate Cross-Site Scripting (XSS) risks
- Strengthen authentication mechanisms and securely manage user sessions
- Remove or disable unnecessary services and restrict access to unused network ports
- Keep all software components up to date by applying patches and updates regularly
- Align development and security practices with the OWASP Top 10 guidelines

Outcome of above project task: -

This project offered hands-on experience with penetration testing methods within a simulated real-world laboratory setup. By evaluating applications hosted across multiple platforms, it enhanced understanding of web application security issues as well as network-level attack vectors.

Integrating manual assessment techniques with automated security tools helped develop stronger technical expertise, improved problem-solving abilities, and strengthened skills in producing clear and professional security reports.

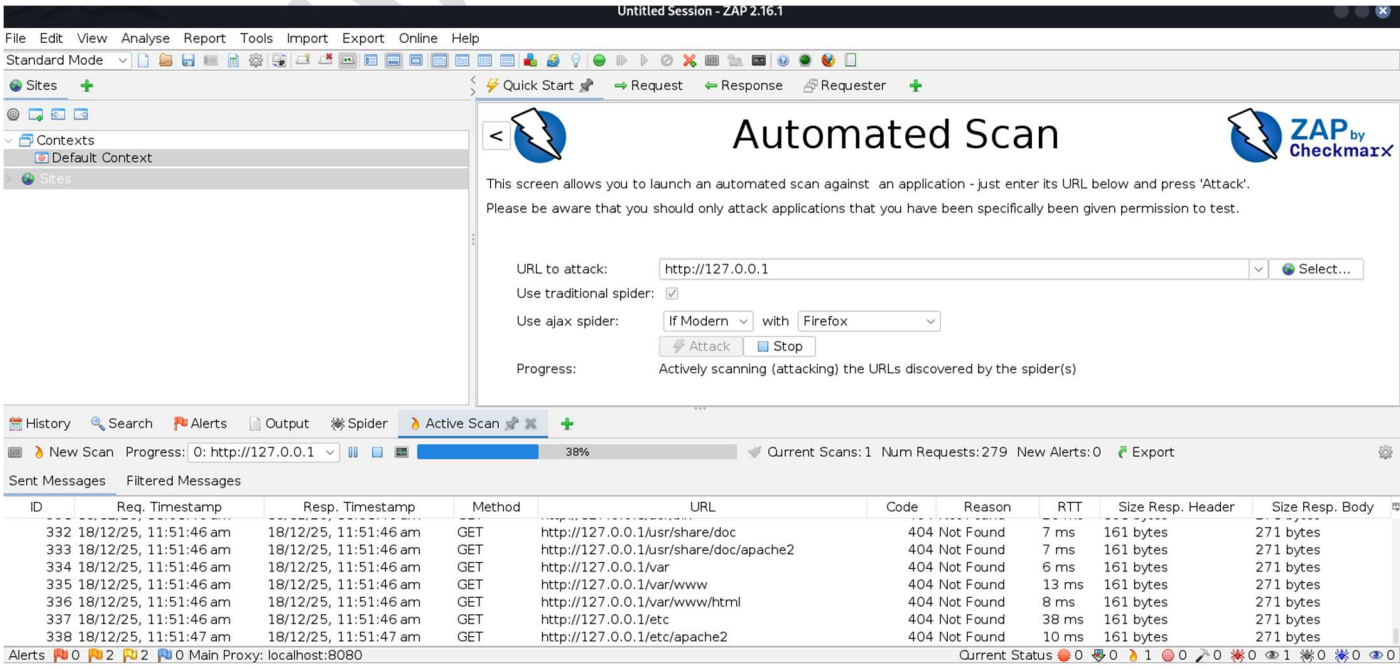
Project 3: Automated vs. Manual Testing Project

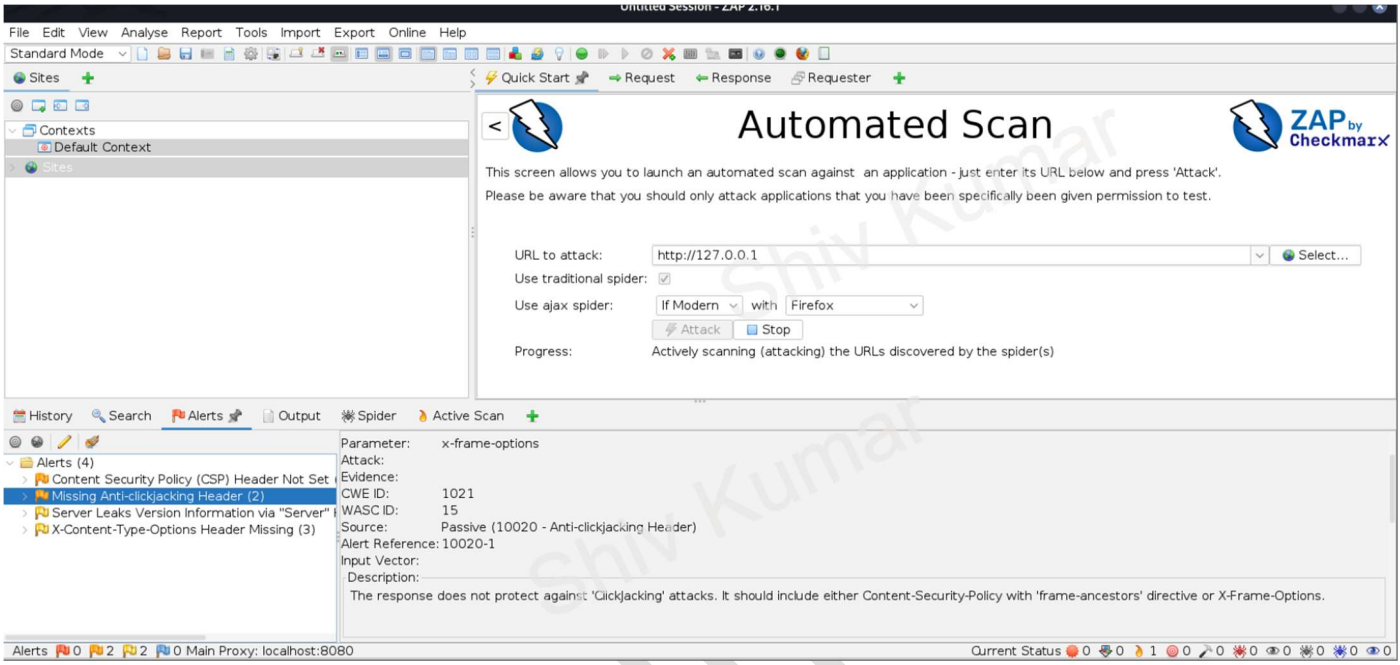
Objective:Analyze differences between automated scanners and manual testing

Tasks:Run Nikto, OWASP ZAP, sqlmap on DVWA.

Using Nikton Scan on DVWA

Using zap on DVWA

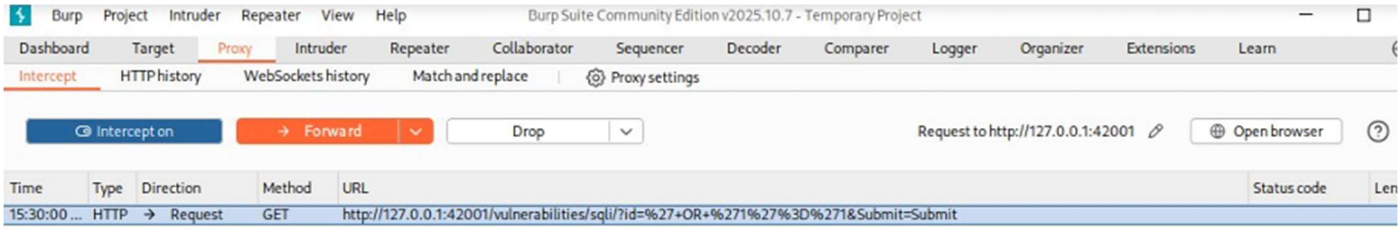




Using SQLMap on DVWA

Manually testing of the pages using Burp Suite

Using Burp suite on DVWA IP



Compare: Which issues automation caught, which only manual testing revealed.

Automated scanning tools rapidly detected well-known security weaknesses, including absent security headers, SQL injection points, and basic cross-site scripting issues. These tools improved efficiency by reducing assessment time and covering a large portion of the application. In contrast, manual security testing provided a more in-depth analysis, enabling accurate payload creation and confirmation of whether vulnerabilities could actually be exploited. Certain problems, such as business logic errors and manipulation of parameters, were more effectively uncovered through hands-on testing. While automated solutions occasionally generated inaccurate or misleading results, manual testing demanded greater expertise and significantly more time.

The “Strengths and Weaknesses of Automated Scanners.”

Advantages:

- Perform scans quickly with minimal effort
- Examine a wide range of application components
- Helpful during the early stages of security assessment
- User-friendly, even for those with limited experience

Limitations:

- May report issues that are not actual vulnerabilities
- Lack insight into business logic and workflow behavior
- Unable to completely confirm the real-world impact of exploits
- Often fail to detect advanced or multi-step security flaws

Project 4: Build Your Own Vulnerable Web App Module

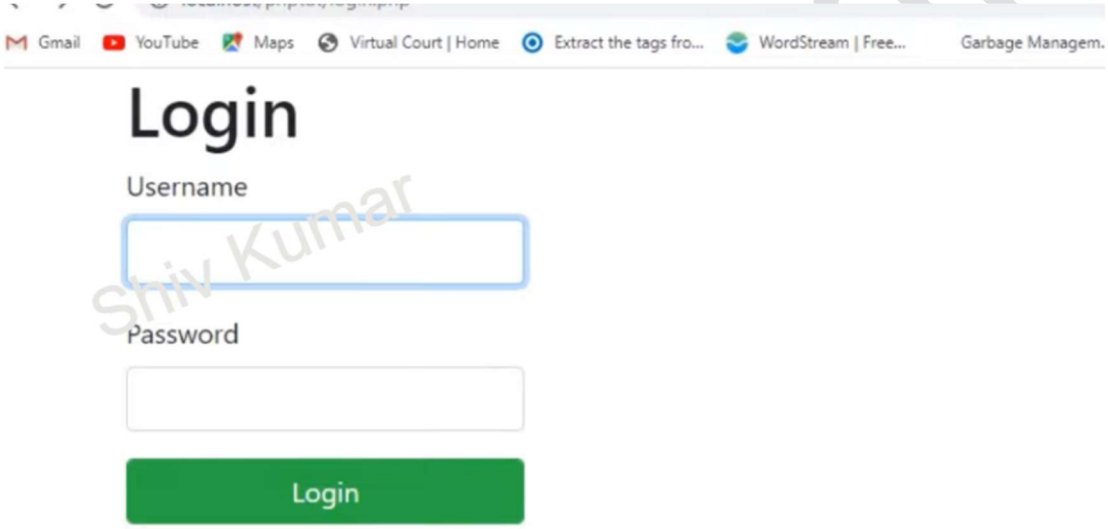
Objective: Understand insecure coding practices.

Tasks: Create a simple PHP page vulnerable to one attack (SQLi or XSS).

We have created: A login page without input validation.

We are using Vulnerable PHP

File name using= login.php

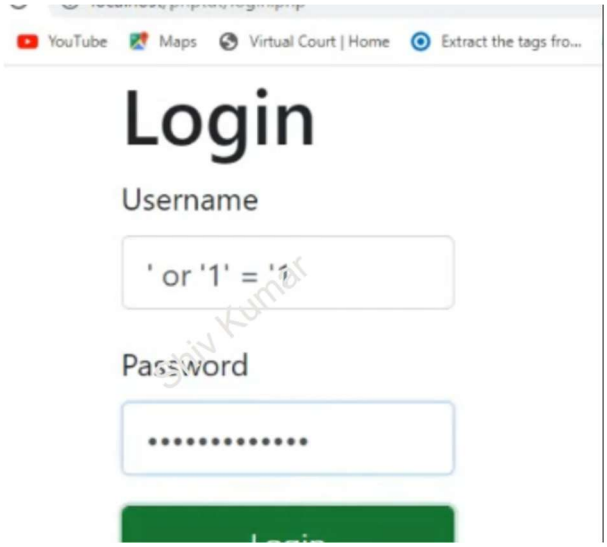


Vulnerable PHP Code

PHP File

```php
<?php
    include 'conn.php';
    if(isset($_POST['login'])){

    $user = $_POST['user'];
    $pass = $_POST['pass'];

    $sql = "SELECT username,password FROM register WHERE username='$user' and password = '$pass'";
        $check =   mysqli_query($conn,$sql);
    if(mysqli_num_rows($check)>0){
        echo "Login success";
    }else{
        echo "Incorrect details";
    }
}
?>
```

We are providing payload ‘or ‘1’ =’1



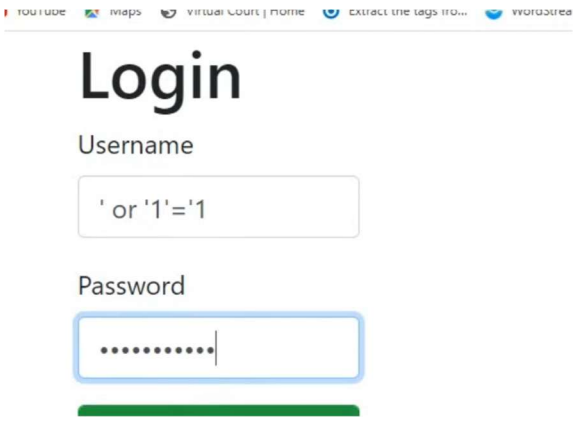
Once we click on login button the result is, we are able to login by using above payload

Now we are applying fix for above vulnerability

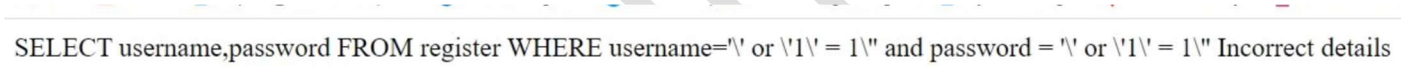
Solution: -We are using escape string in our code to fix the vulnerability

```php
<?php
    include 'conn.php';
    if(isset($_POST['login'])){

    $user = mysqli_real_escape_string($_POST['user']);
    $pass = mysqli_real_escape_string($_POST['pass']);

    $sql = "SELECT username,password FROM register WHERE username='$user' and
        password = '$pass' ";
         $check =   mysqli_query($conn,$sql);
    if(mysqli_num_rows($check)>0){
        echo "Login success";
    }else{
        echo "Incorrect details";
    }
}
?>
```

We are trying payload again to login-



Once we click on login button the result is, we are now not able to login by using above payload



Conclusion from above project activity

This project illustrated how poor coding practices can lead to serious security weaknesses in web applications. It was shown that even a basic login page can be exposed to SQL Injection attacks when user input is not handled correctly.

However, after applying prepared statements, the vulnerability was fully eliminated. This demonstrates the critical role of integrating secure coding techniques throughout the development process, rather than depending only on security testing after deployment.

Project 5: Research Project: Common Vulnerabilities Across Real Web Apps

Objective: Link DVWA learning to real-world CVEs.

Tasks: Pick one DVWA vulnerability (File Upload).

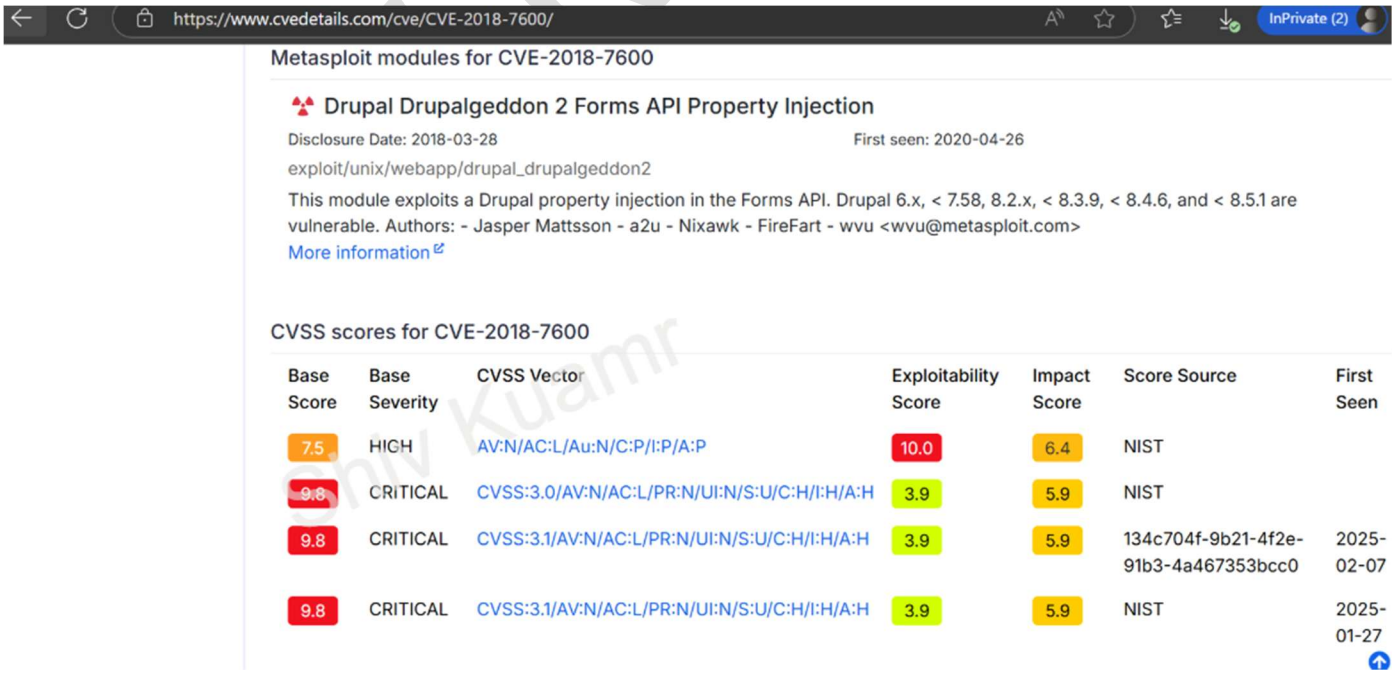
Real World Scenario Case 1: CVE-2018-7600 — Drupal File Upload Leading to Remote Code Execution

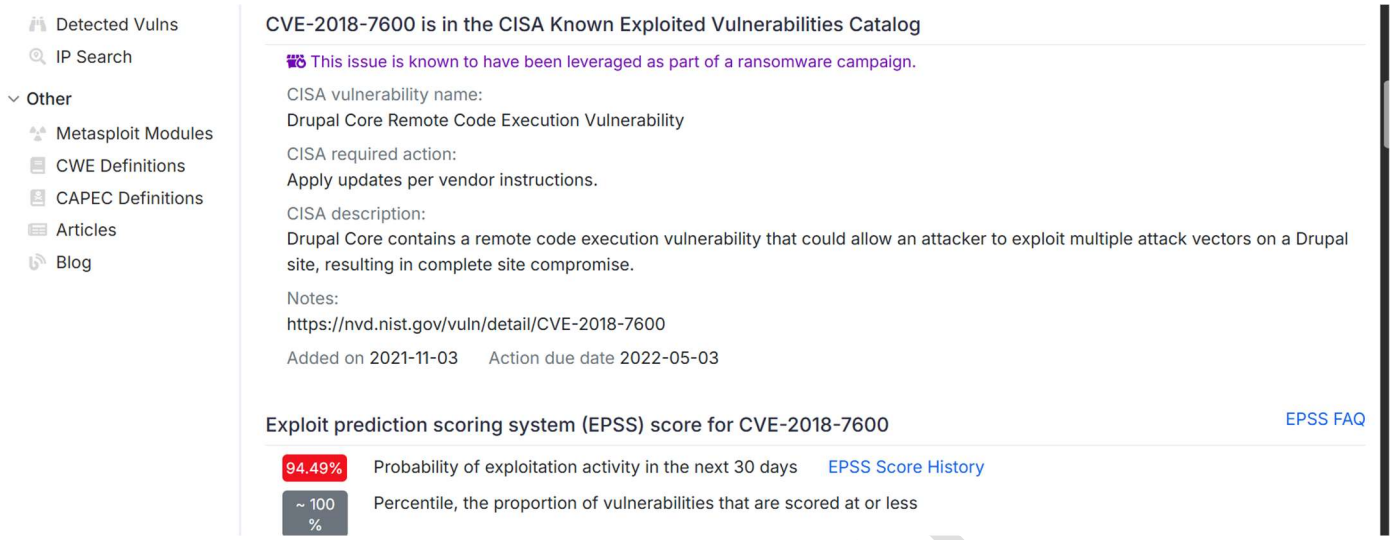
Description of above CVE:- CVE-2018-7600 is a real-world vulnerability in the **Drupal CMS** where improper handling of user input and file uploads allowed attackers to execute malicious code on the server.

This is very similar to the **DVWA File Upload vulnerability**, where the application does not properly restrict uploaded files.

How Attackers Exploited It

- Drupal allowed attackers to upload malicious files through a vulnerable form.
- The uploaded file contained executable code (such as a PHP script).
- Once uploaded, the attacker accessed the file through the browser.
- The server executed the malicious file, giving the attacker control.





Impact

- Attackers gained **remote code execution**.
- Full compromise of affected websites.
- Data theft, website defacement, and malware installation.

How It Was Fixed

- Drupal released a security patch.
- File uploads were restricted to safe file types only.
- Better server-side validation was added.
- Uploaded files were prevented from being executed.

Real world scenario case-2: CVE-2017-12615 — Apache Tomcat File Upload Vulnerability

Description: - CVE-2017-12615 is a vulnerability in **Apache Tomcat** where improper handling of uploaded files allowed attackers to upload and execute malicious **JSP files**.

This is very similar to DVWA's File Upload issue, where file type validation is missing or weak.

Both **DVWA File Upload** and **CVE-2017-12615 (Apache Tomcat)** suffer from the same core security issue: **improper handling of uploaded files**.

DVWA File Upload

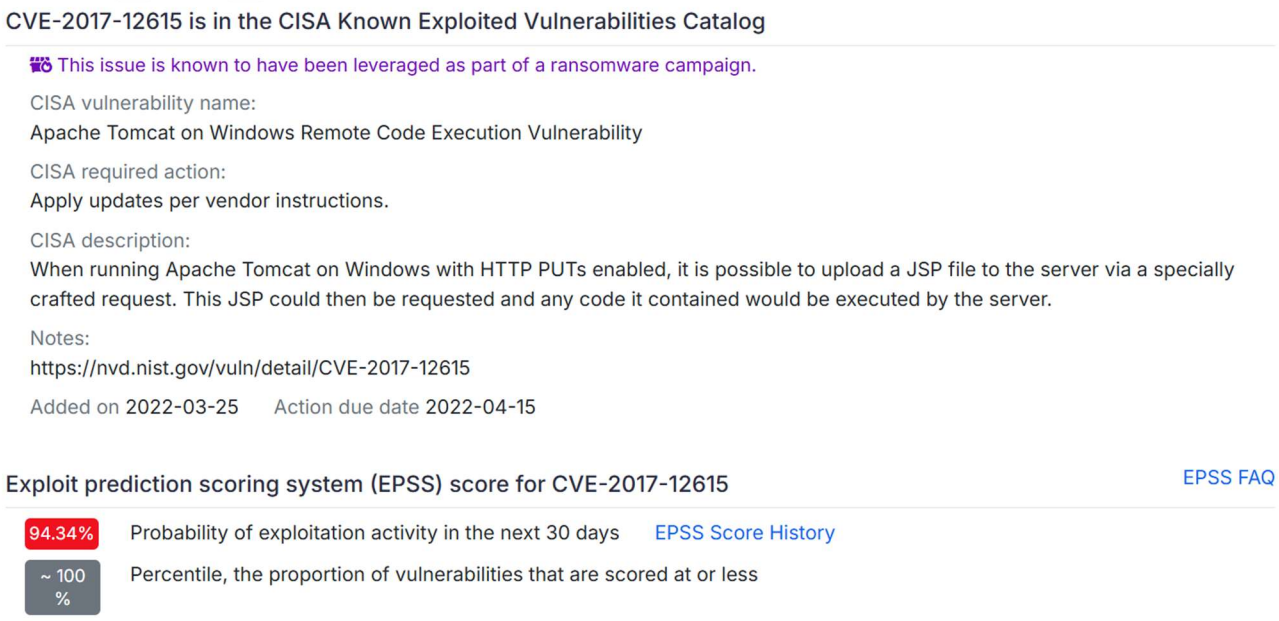
- DVWA allows users to upload files without properly checking the file type.
- An attacker can upload a malicious file (e.g., shell.php).
- When the uploaded file is accessed, the server executes it.
- This gives the attacker control over the application.

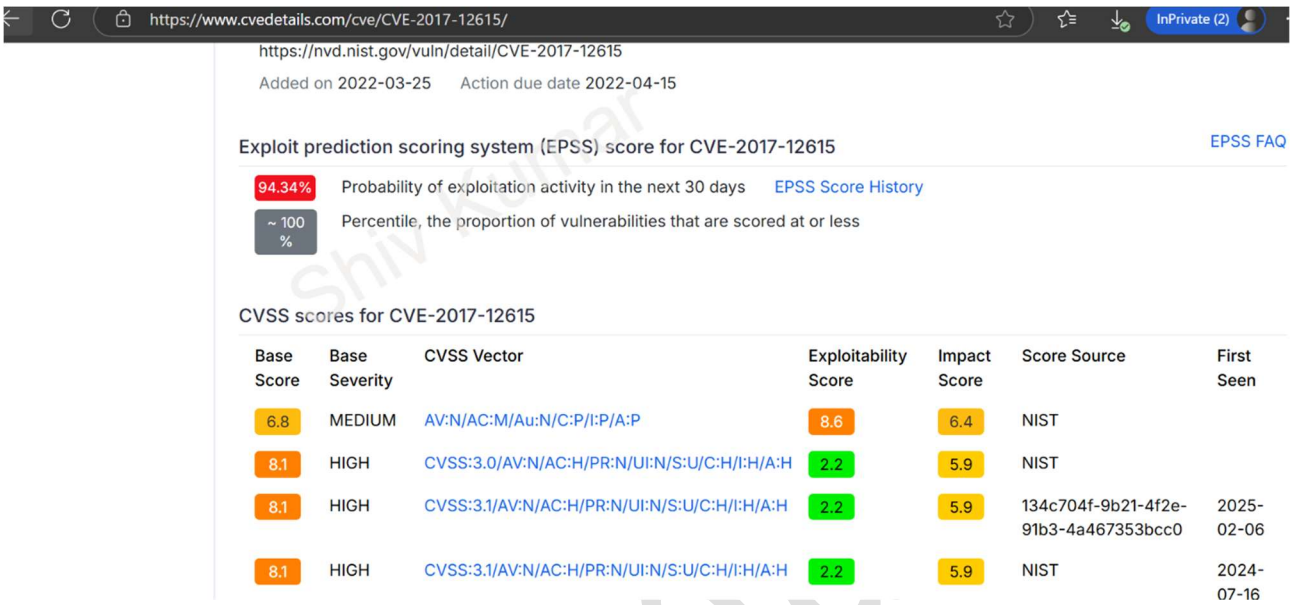
CVE-2017-12615 (Apache Tomcat)

- Apache Tomcat allowed attackers to upload malicious JSP files.
- The server did not correctly restrict file uploads or execution.
- When the uploaded JSP file was accessed, it executed on the server.
- This resulted in remote code execution.

How Attackers Exploited It

- Tomcat allowed file uploads using the HTTP PUT method.
- Attackers uploaded a malicious file such as shell.jsp.
- Once uploaded, the attacker accessed the file through a browser.
- The server executed the malicious JSP file.





Impact

- Remote Code Execution (RCE).
- Full control over the web server.
- Website defacement, data theft, or malware installation.

How It Was Fixed

- Apache released a security update.
- File upload permissions were restricted.
- Execution of uploaded files was disabled.
- Unsafe HTTP methods were blocked.

Conclusion from above project activity

The findings of this study show that the security flaws explored through DVWA closely mirror those found in real-world web applications. In particular, file upload weaknesses have been responsible for serious security events, such as unauthorized server access and exposure of sensitive data.

Practicing these attack scenarios in a controlled laboratory setting allows security practitioners to better identify potential risks and apply effective defenses before similar exploits can occur in production environments.