Final Document

ANALYSIS AND PREVENTION OF OWASP WEB APPLICATION SECURITY RISKS

Vibha Garg
19BCE0350
8968130219
vibha.garg2019@vitstudent.ac.in

Parth Maheshwri
19BCT0221
8401291012
parth.maheshwari@vitstudent.ac.in

Bhulakshmi Bonthu
Assistent Professor (Senior)
9790862137
bhulakshmi.b@vit.ac.in

B.Tech.

in

Computer Science and Engineering

School of Computer Science & Engineering

1. KEYWORDS

Xamp, Owasp, Broken Authentication, Sensitive Data Exposure, XSS Injections, Remote Control Execution, Security Misconfiguration, Vulnerabilities, PHP, JavaScript.

2. Introduction

2.1 Theoretical Background

With the advent of cloud computing, web applications are growing more and more popular every day. They provide a common interface independent of the host computer. Web apps are faster, more intuitive, and can reach more people than everyday installed software. But since web apps are open, this creates a bigger attack surface for hackers to exploit and take advantage of. There have been many public attacks that result in millions of users being exposed, like Facebook. This resulted in the formation of a non-profit foundation, known as OWASP. It gives preventive methodologies, documentation, and tools available for free to help make web apps safer. They also release a list of the top 10 most known vulnerabilities, along with safety measures to prevent it from being exploited. In this paper, we will create a vulnerable web app to showcase these vulnerabilities, then elaborate on how to mitigate these risks afterward.

2.2 Motivation

With the revolution in Internet and the amount of traffic on Internet related applications like Web applications and Websites, data privacy and security remain a big challenge for Security Analysts. Our agenda is to explore the most common Owasp vulnerabilities and then perform in-depth research to analyses the security and impact of the risks to get an in-depth knowledge of how dangerous weak web apps can be.

2.3 Aim of the proposed work

Web applications are versatile and have proven to be better than conventional applications which require specific conditions. But this has also led to increased security risks due to the open nature of these apps. The objective of this paper will be to recreate the vulnerabilities given in the OWASP Top 10 vulnerabilities list, then perform in-depth research to analyses the security and impact of the risks to get an in-depth knowledge of how dangerous weak web apps can be.

2.4 Objectives of the proposed work

Our methodology to test out these security risks will be by first creating a web app based on the WAMP web stack, where Windows is the host, Apache is the server, the

backend is MySQL and the programming language will be PHP. Since this has one of the largest communities of support and majority of the websites are either coded in WAMP or LAMP stack. Then we will test the vulnerabilities given in the OWASP Top 10 list, following which we will try to secure the risks and compare the results before and after reinforcing our web app.

3. Literature Survey

1	Effective Filter for Common Injection Attacks in Online Web Applications	Santiago ibarra-fiallos1 , javier bermejo higuera 1, Monserrate intriago-pazmiño2 , juan ramón bermejo higuera 1, Juan antonio sicilia montalvo 1 , and javier cubo1 Institution/Department: 1escuela superior de ingeniería y tecnología, universidad internacional de la rioja, 26006 logroño, spain 2departamento de informática y ciencias de la computación, escuela politécnica nacional, quito 170450, ecuador	This paper designs an effective protection of web applications against common injection attacks. It includes a validation filter of input fields that is based on OWASP Stinger, a set of regular expressions, and a sanitization process. It validates both fundamental characters (letters, numbers, dot, dash, question marks, and exclamation point) and complex statements (JSON and XML files) for each field. The goal of this article is to contribute to reducing the injection attacks through the design of a new filter based on OWASP Stinger and set as a module on a Jboss/Wildfly application server. The filter helps web applications to create at least one layer of protection against common injection attacks (SQL Injection (SQLi), Command Injection (CI), Cross Site Scripting (XSS), etc.), which occur when entries are not validated.
2	XSS Vulnerability Assessment and Prevention in Web Application	Ankit Shrivastava, Santosh Choudhary & Ashish Kumar. Email: ankitshrivastavaieee@gmail.-com shellysamota09@gmail.com aishshub@gmail.com	Cross site scripting (XSS) is a type of scripting attack on web pages and accounts as one of the unsafe vulnerabilities that exist in web applications. It is not sufficient in the prevention of more dangerous XSS payloads. The use of one or two existing approaches can stop the direct malicious inputs from the web browser, but not strong enough to handle middleware a6acks. The re-

			searchers are using javascript validation for user input, javascript signature mechanism to identify valid javascript, assigning a unique token for client server request during communication, using escape method to prevent script characters, saniEzaEon method to clean-up HTML text.
3	Automatic Detection of Security Misconfigurations in Web Applications	Sandra Kumi, ChaeHo Lim, Sang-Gon Lee	Improper configuration of web applications or servers can lead to various security flaws. The exploitation of this kind of vulnerabilities can lead to exploitation of other severe vulnerabilities and complete compromise of web applications. Attackers exploit misconfiguration vulnerabilities through unprotected files and directories, unused web pages, unpatched flaws, and unauthorized access to default accounts. The proposed approach uses DAST to detect security misconfiguration vulnerabilities in web applications. Their tool simulates an attack against target web applications through HTTP requests and analyses responses from the web application's server to determine security misconfigurations. The steps used by them in detecting security misconfigurations in web applications are crawling web applications, identification of input parameters, attack generation, and report generation.
4	Broken Authentication and Session Management Vulnerability: A Case Study of Web Application	Md. Maruf Hassan*1,2, Shamima Sultana Nipa1 , Marjan Akter1 , Rafita Haque2 , Fabiha Nawar Deepa2 , Mostafijur Rahman1,2, Md. Asif Siddiqui1 , Md. Hasan Sharif1 Emails: maruf.swe@diu.edu.bd,	The survey was performed on the various types of SQLi and XSS vulnerabilities in a web application where the author of the article suggested some countermeasures to defend those attacks. Review on the most prevailing vulnerabilities on web applications was exploited using different hacking tools and pre-

		shamima nipa743@gmail.com, marjanshifat95@gmail.com, rafitahaque93@gmail.com, fabiha.deepa@gmail.com, mostafijur.swe@diu.edu.bd, sharif.swe@diu.edu.bd	ventive guidelines were also provided through a solution of those attacks. We can prevent broken authentication using the following: Session ID Life Cycle, Session Reset, Session Expiration, Cookies, Session Attacks Detection, Client-Side Defences for Session Management, Generating an Access Token. Regulate session length: The web application must be able to end web sessions after a period of inactivity that depends on the type of requirements of the user.
5	Sensitive Data Exposure Prevention using Dynamic Database Security Policy	Jignesh Doshi and Bhushan Trivedi Institutions/Department: LJ Institute of Management Studies, Ahmedabad, Gujarat, India GLS Institute of Computer Technology, Ahmedabad, Gujarat, India	Attackers use different paths for executing their tasks. One attack can have several impacts on business. Application software with security leakages is dejecting our financial, defence, energy, healthcare and other critical infrastructure. It is observed that there are many leakages in the security of web applications. The authors have proposed a highly dynamic and flexible automated approach to prevent sensitive data exposure. This solution is deployed at database level on Oracle database. Major 3 phases of the proposed solution are Build SDF catalogue, creating custom security functions/ policies and configuring custom policies using fine grained access control mechanism of oracle database.

4. Overview of the Proposed System

4.1 Introduction and Related Concepts

4.1.1 Cross Site Scripting (XSS)

XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. Attackers can execute scripts in a victim's browser to hijack user sessions, deface web

sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.

4.1.2 Security Misconfiguration

Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, framework, and custom code. Attacker accesses default accounts, unused pages, un-patched flaws, unprotected files, and directories, etc. to gain unauthorized access to or knowledge of the system.

4.1.3 Sensitive Data Exposure

Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud. For all sensitive data deserving encryption, do all of the following, at a minimum: 1. Ensure all sensitive data should be kept encrypted/hashed with a strong encryption/hashing algorithm within the database. 2. Ensure all keys and passwords are protected from unauthorized access.

4.1.4 Broken Authentication and Session Management

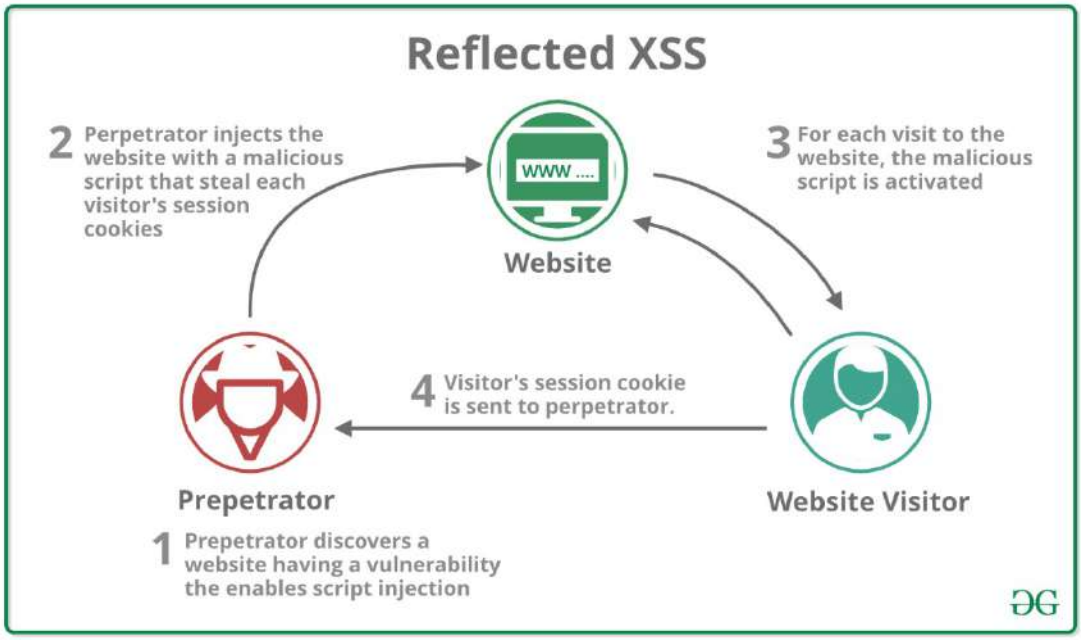
Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities. Such flaws may allow some or even all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted.

4.1.5 Remote Control Execution

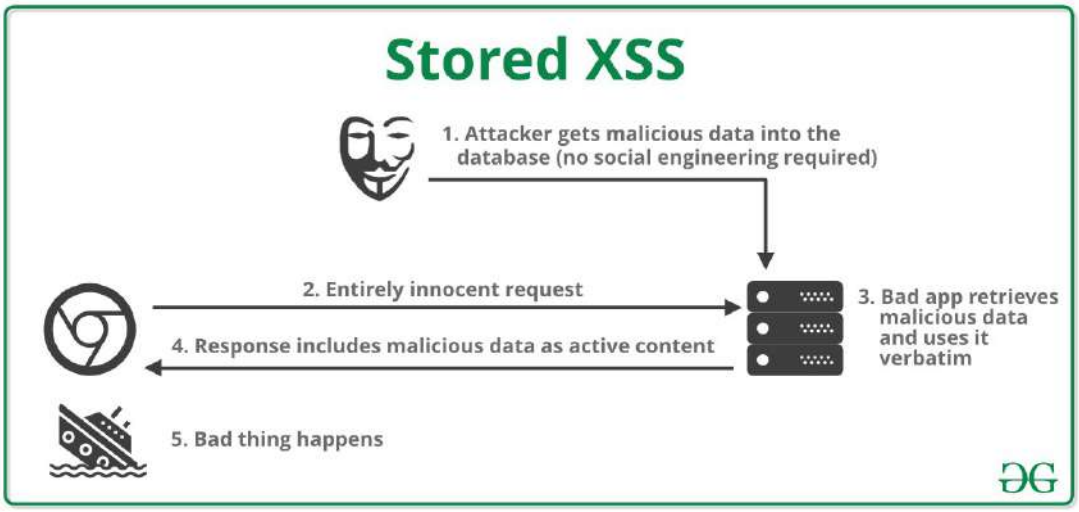
Remote code execution (RCE) attacks allow an attacker to remotely execute malicious code on a computer. The impact of an RCE vulnerability can range from malware execution to an attacker gaining full control over a compromised machine. Free Trial 2022 Cyber Security report.

4.2 Attack Architecture

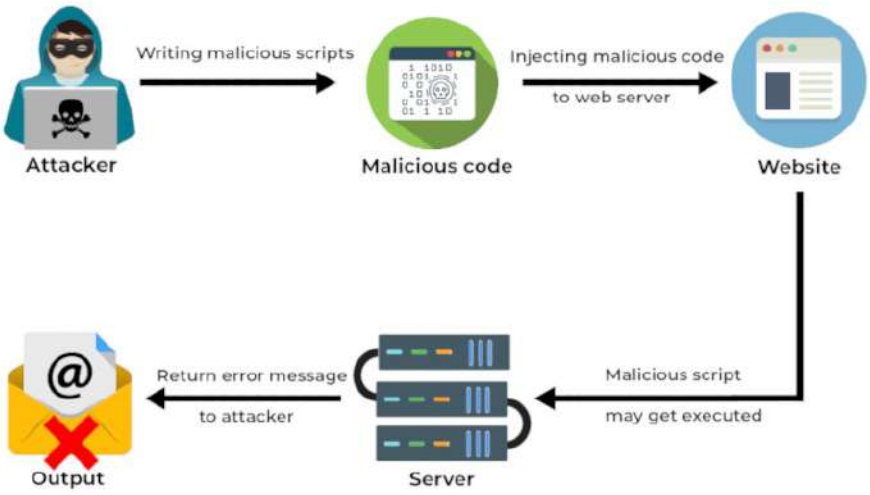
1. XSS (Reflected)



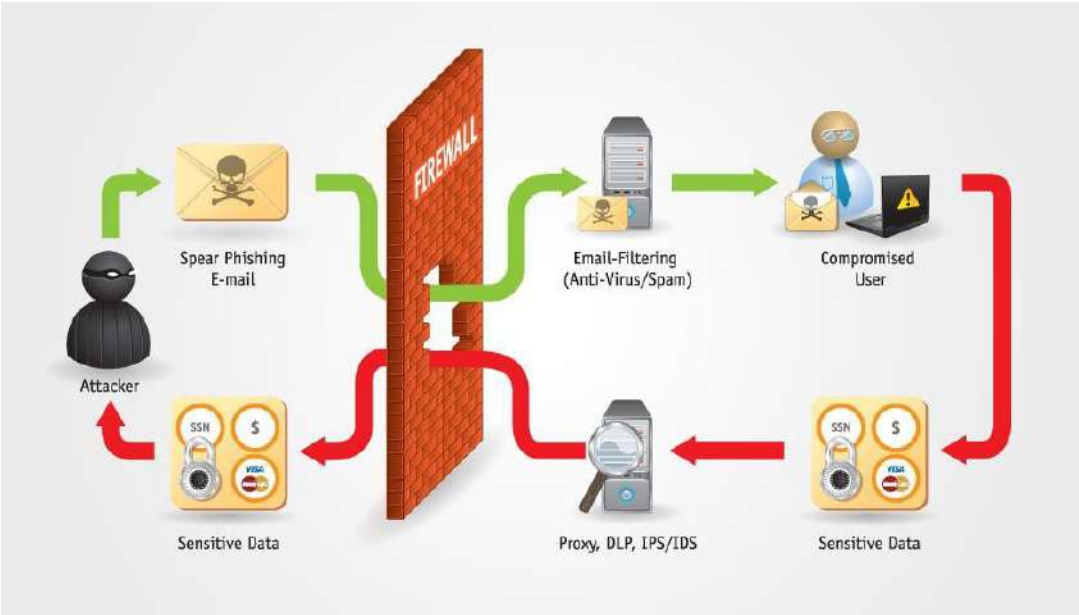
1. XSS (Stored)



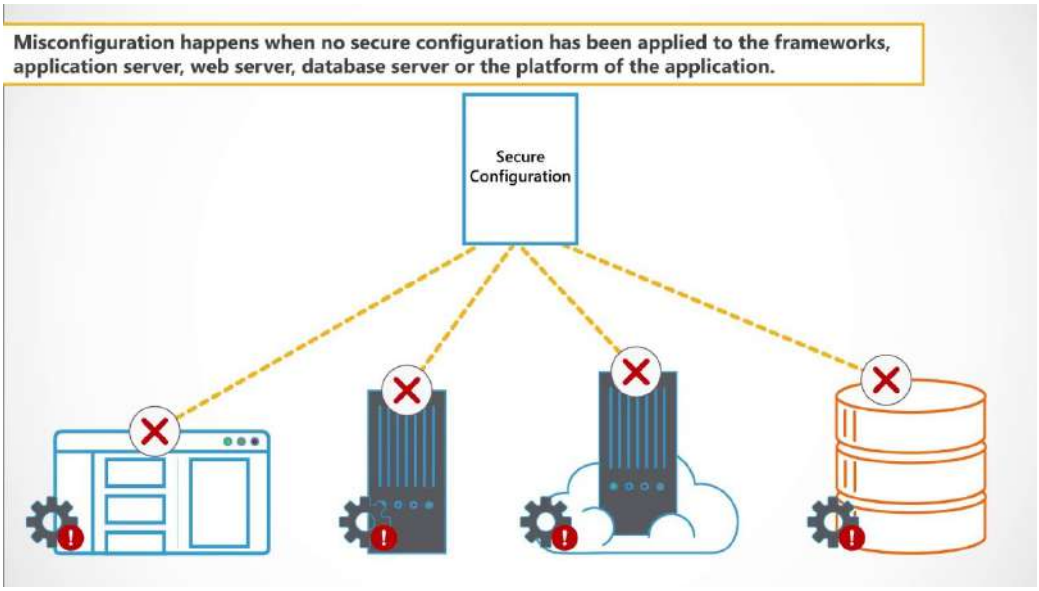
2. Remote Control Execution:



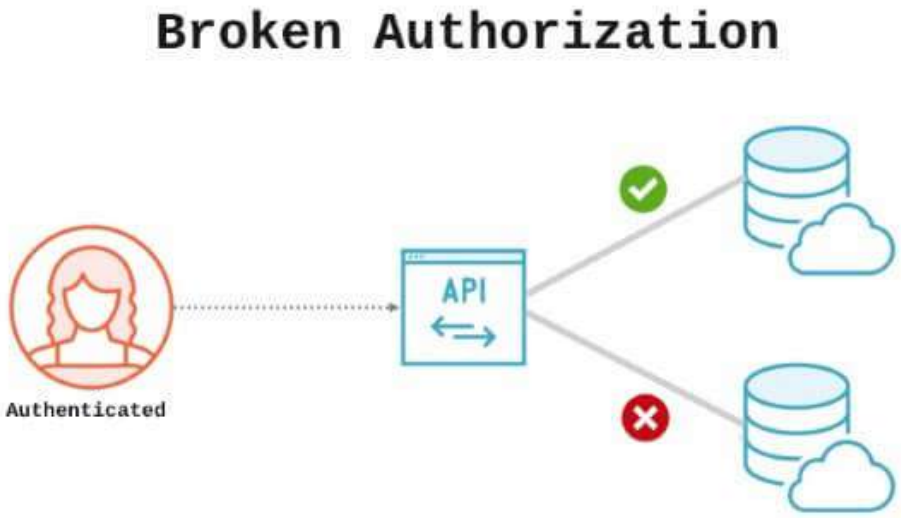
3. Sensitive Data Exposure:



4. Security Misconfiguration:



5. Broken Authentication:

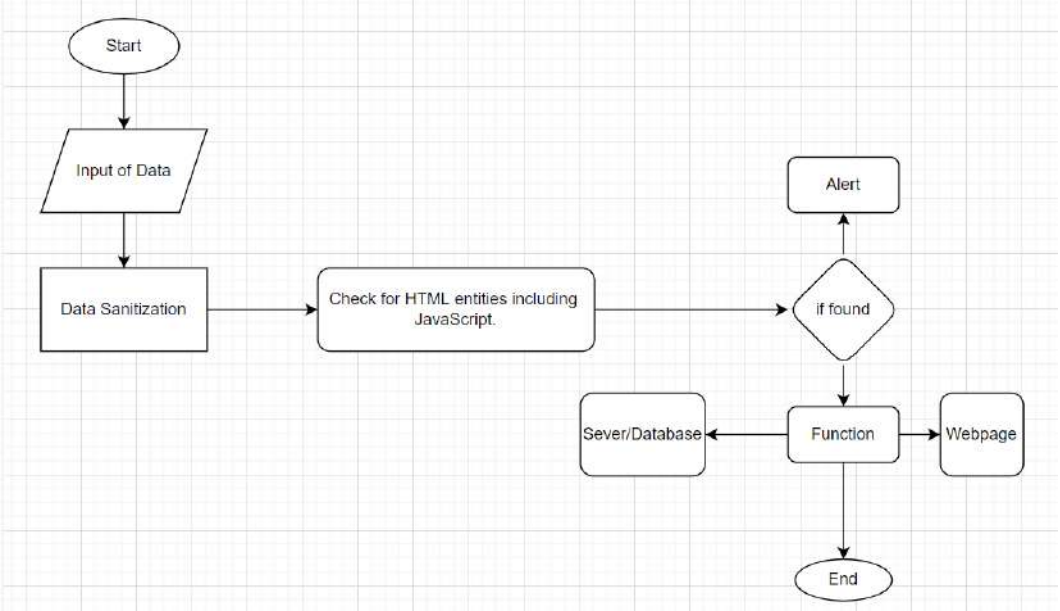


5. Proposed System Analysis and design

5.1 Introduction

5.1.1 XSS

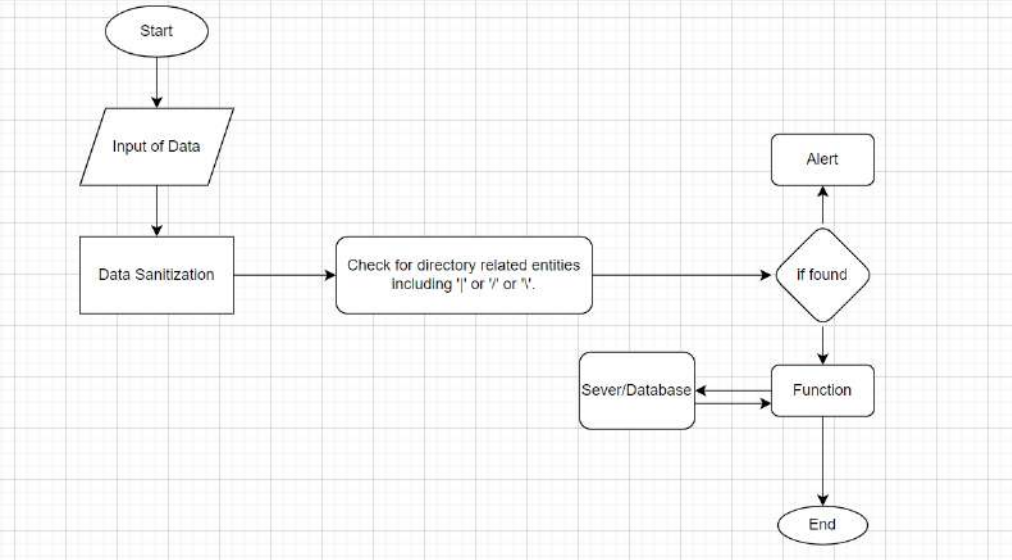
To keep yourself safe from XSS, you must sanitize your input. Our application code should never output data received as input directly to the browser without checking it for malicious code. Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.



5.1.2 RCE

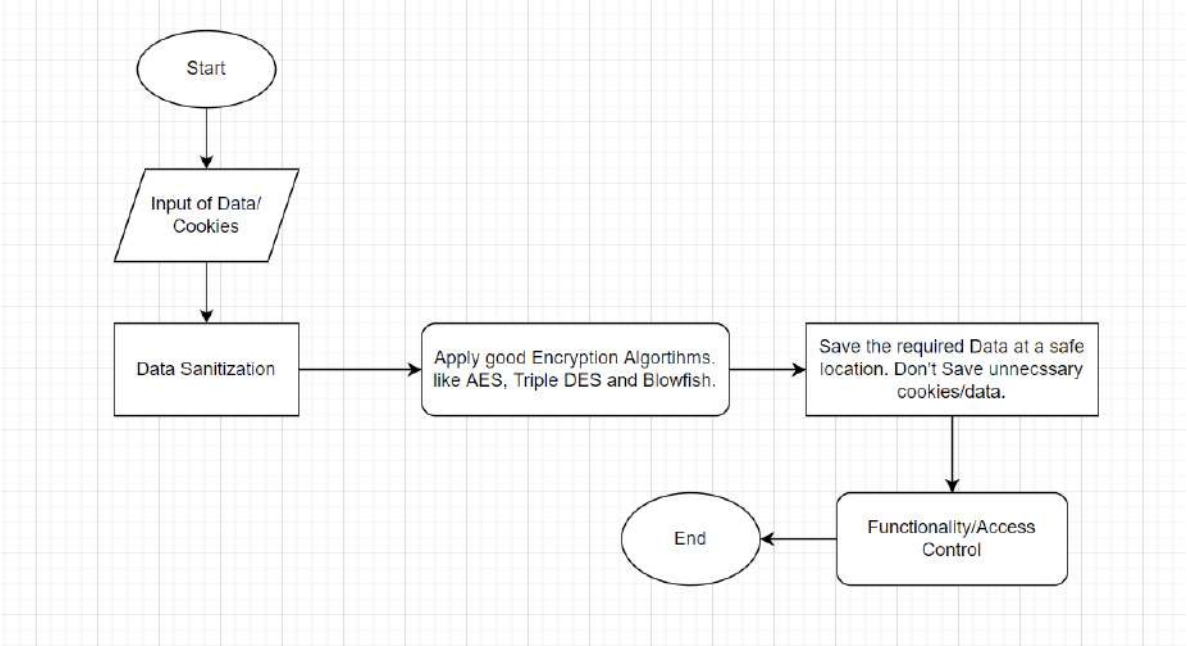
It is necessary to focus on the importance of having robust security measures in place. We should always be aware of how our server handles user-provided information. We can mitigate remote code execution by using the following techniques:

1. Timely patching or installation of software updates is an essential preventative measure
2. Avoid using user input inside the evaluated code
3. Don’t use functions such as eval at all
4. Use safe practices for secure file uploads and never allow a user to decide the extension or content of files on the web server



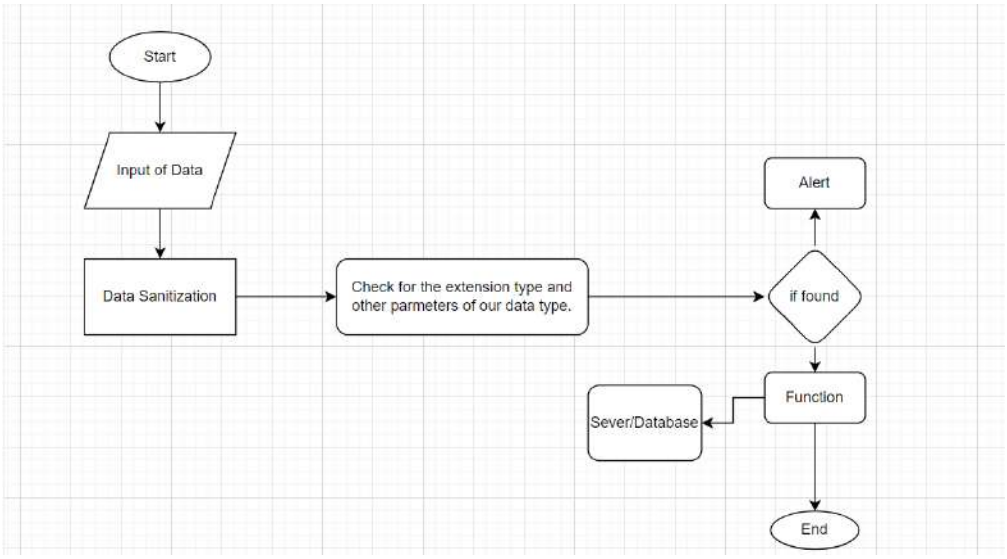
5.1.3 Sensitive Data Exposure

1. Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all sensitive data at rest and in transit in a manner that defends against these threats.
2. Don’t store sensitive data unnecessarily. Discard it as soon as possible. Data you don’t have can’t be stolen.
3. Ensure strong standard algorithms and strong keys are used, and proper key management is in place.
4. Ensure passwords are stored with an algorithm specifically designed for password protection,
5. Disable autocomplete on forms collecting sensitive data and disable caching for pages that contain sensitive data.



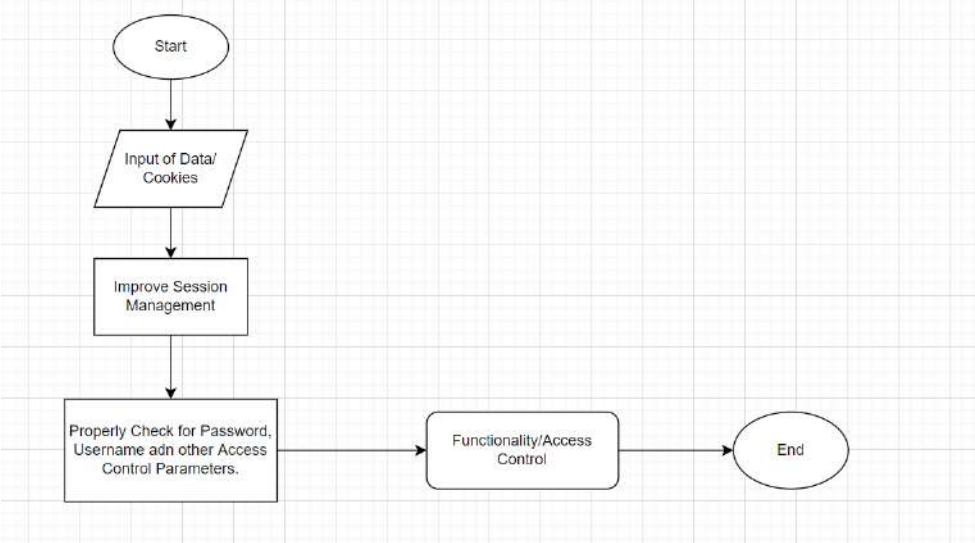
5.1.4 Security Misconfiguration

1. Limit access to administrator interfaces
   Part of your deployment policy should be disabling admin portals to all but certain permitted parties. The implementation of the policy should also be reviewed via regular audits.
2. Disable debugging
   This is especially critical when deploying to a production environment. You'll want to pay particular attention to the configuration for debugging features, and all of them should be disabled.
3. Disable the use of default accounts and passwords
   The first step after installing any device or piece of software should be to create a new set of credentials. The default should never be used. You'll want to make this a mandatory part of your company policy. You'll want to employ other password-related best practices as well, like having maximum lengths for passwords and limiting the number of permitted login failures.



5.1.5 Broken Authentication

1. Regulate session length:
   The web application must be able to end web sessions after a period of inactivity that depends on the type of requirements of the user. A secure banking portal, for example, must automatically log out the user after a few minutes to avoid any risks of hijacked session IDs
2. Improve session management:
   The web application must be able to issue a new Session ID after every successful authentication. These IDs must be invalidated as soon as a session ends in order to prevent any misuse. Web URLs must be secure and must not include the Session ID in any form.
3. Multi-factor Authentication (MFA):
   Among the OWASP top 10 broken authentication, the first tips is to implement Multi-factor Authentication to prevent attacks. MFA requires an additional credential to verify the user's identity. An example of MFA would be a One-Time Password (OTP) mailed or messaged to the user that allows for verification.



5.2 Requirement Analysis

5.2.1 System requirements:

Hardware:

1. Normal functional computer/laptop.
2. Any OS with good Internet Connectivity.

Software:

1. XAMPP.
2. Web Browser.
3. PHP, MySQL, Apache.
4. Visual Studio Code.
5. Base64 Decode.

6. Modules Identified

6.1 Cross site scripting

6.1.1 Code :

Reflected XSS

```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="preconnect" href="https://fonts.googleapis.com">
```

```
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link href="https://fonts.googleapis.com/css2?
family=Zen+Kurenaido&display=swap" rel="stylesheet">
        <style type="text/css">
            * {
                font-family: 'Zen Kurenaido', sans-serif;
                font-size: 20px;
            }
            body {
                color: white;
                margin: 0;
                display: flex;
                background-image: url("bg2.jpg");
                background-size: cover;
                background-repeat: no-repeat;
                justify-content: center;
                align-items: center;
                width: 100%;
                height: 100%;
            }
            #vulninput {
                display: flex;
                width: 50vw;
                height: 50vh;
                justify-content: center;
                align-items: center;
            }
            input {
                border: none;
            }
        </style>
    </head>
    <body>
        <div id="vulninput">
            <form method="get">
                Print a message on the screen:<p></p>
                <input type="text" placeholder="Message" name="m">
                <input type="submit" value="Send">
                <p></p>
                <?php
                    if(isset($_GET['m'])) {
                        echo htmlentities($_GET['m']);
```

```
                    }
                ?>
            </div>
        </div>
    </body>
</html>
```

Stored XSS

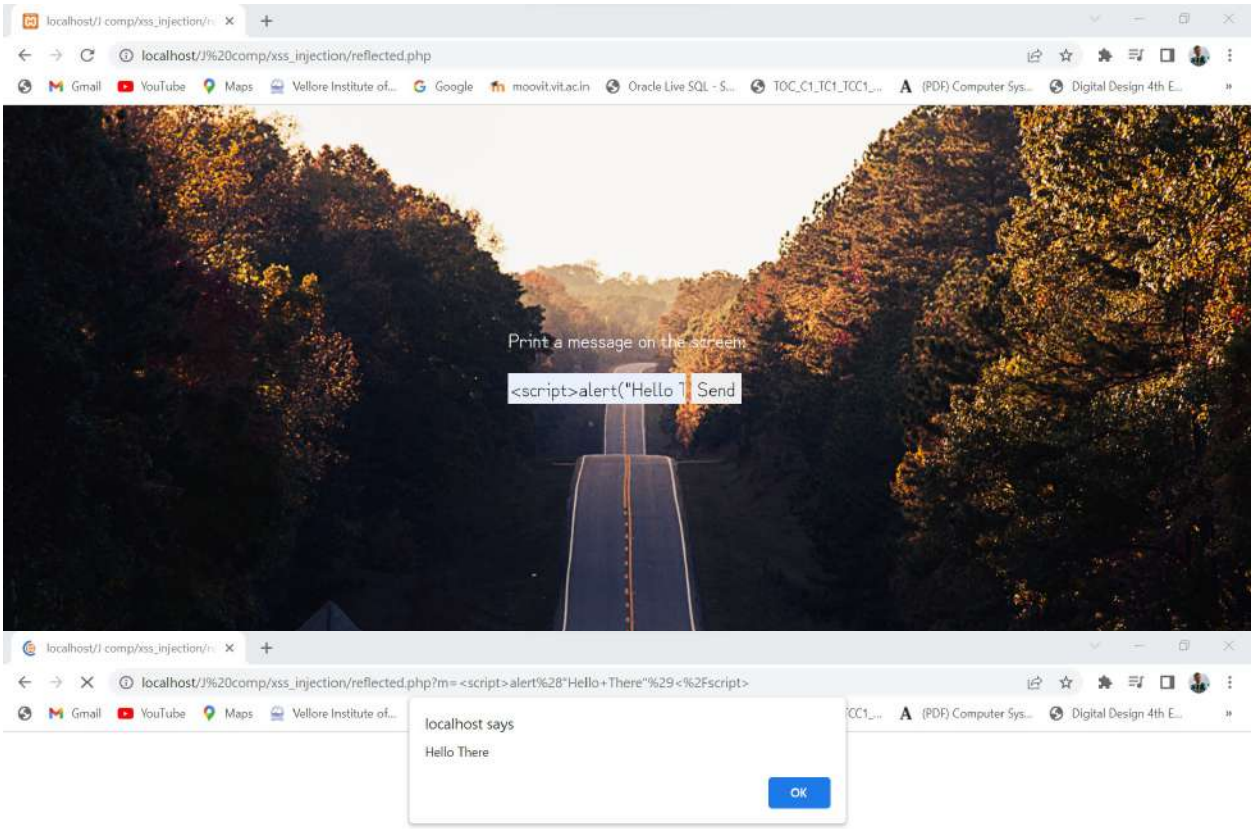
```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link href="https://fonts.googleapis.com/css2?
family=Zen+Kurenaido&display=swap" rel="stylesheet">
        <style type="text/css">
            * {
                font-family: 'Zen Kurenaido', sans-serif;
                font-size: 20px;
            }
            body {
                background-image: url("bg2.jpg");
                background-size: cover;
                background-repeat: no-repeat;
                margin: 0;
                display: flex;
                justify-content: center;
                align-items: center;
                width: 100%;
                height: 100%;
                flex-flow: column nowrap;
            }
            #vulninput {
                display: flex;
                width: 50vw;
                height: 20vh;
                justify-content: center;
                align-items: center;
            }
            .ele {
                height: 80vh;
```

```
                width: 100vw;
                display: flex;
                align-items: center;
                justify-content: center;
            }
            input {
                border: 1px black 0.3px;
            }
            #comments {
                width: 50vw;
                height: 50vh;
                border-radius: 10px;
                background-color: rgba(255, 255, 255, 0.2);
            }
            td {
                color: white;
            }
        </style>
    </head>
    <body>
        <div class="ele" id="vulninput">
            <form method="post">
                <input type="text" placeholder="Your comment"
name="comment">
                <input type="submit" value="Comment">
            </form>
        </div>
        <div class="ele" id="comments">
            <table>
                <tbody>
                    <?php
                        class usersDB extends SQLite3 {
                            function __construct() {
                            $this->open('comments.db');
                            }
                        }
                        $conn = new usersDB();
                        if(!$conn) {
                            echo "<tr>Connection to DB failed</tr>";
                            exit;
                        }
                        if(isset($_POST['comment'])){
                            $comm = $_POST['comment'];
```

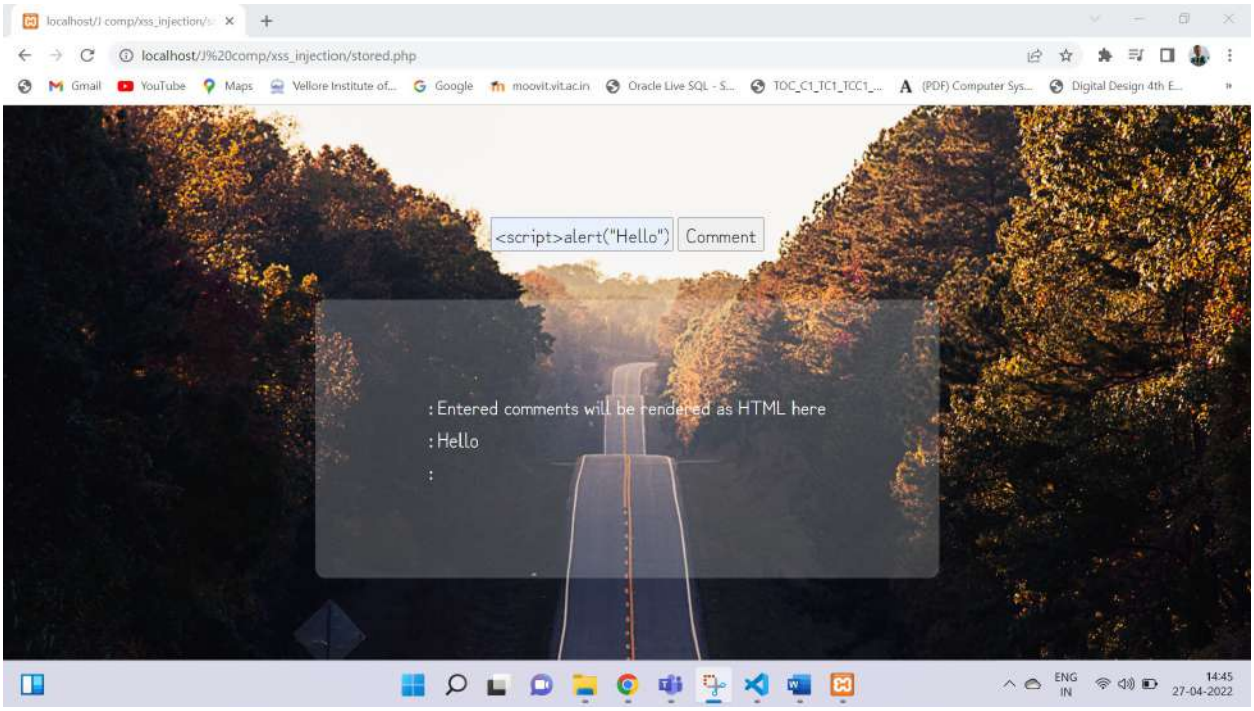
```
                            $com = "INSERT INTO comments (comment) VALUES
('".$comm."')";
                            $res = $conn->exec($com);
                            if(!$res){
                                echo "<tr>Error in entering
comment</tr>";
                                exit;
                            }
                        }
                        $query = "SELECT comment FROM comments;";
                        $res = $conn->query($query);
                        while($row = $res->fetchArray(SQLITE3_ASSOC)) {
                            echo "<tr><td>:</td><td
class=\"com\">".htmlentities($row['comment'])."</td></tr>";
                        }
                        $conn->close();
                    ?>
                </tbody>
            </table>
        </div>
    </body>
</html>
```

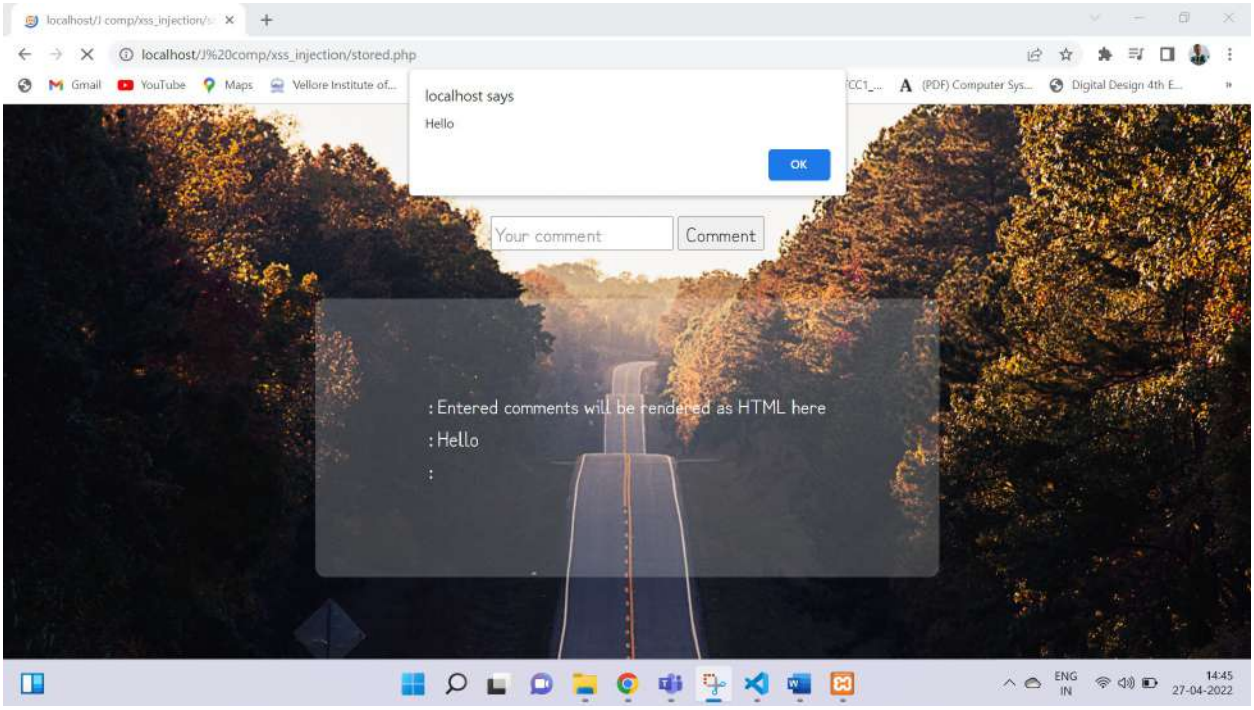
6.1.2 Output:

Attack Results (Reflected XSS):

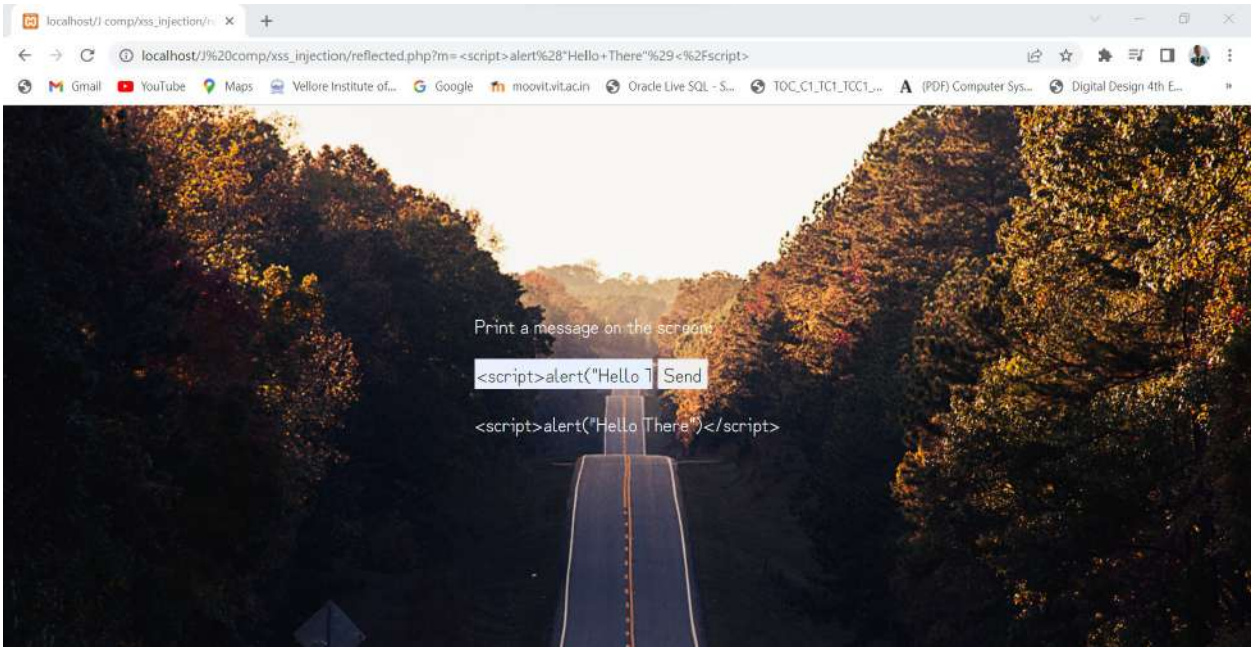


Attack Results (Stored XSS):

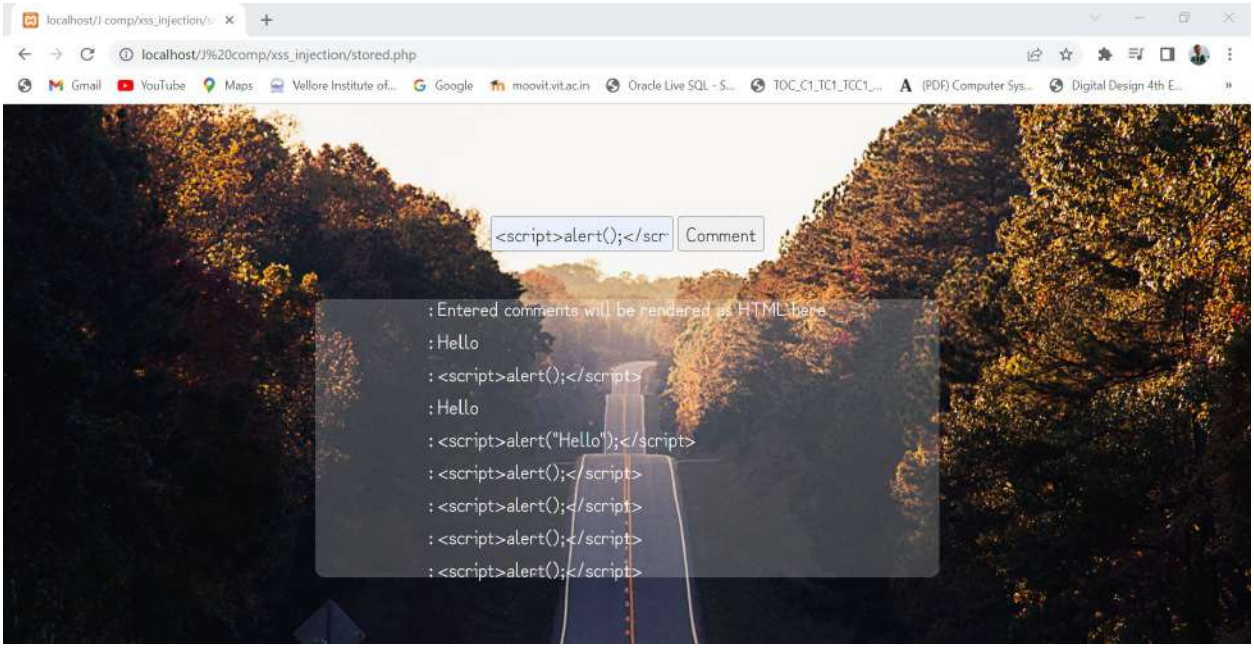




Prevention Results (Reflected XSS):



Prevention Results (Stored XSS):



6.2 Remote code execution

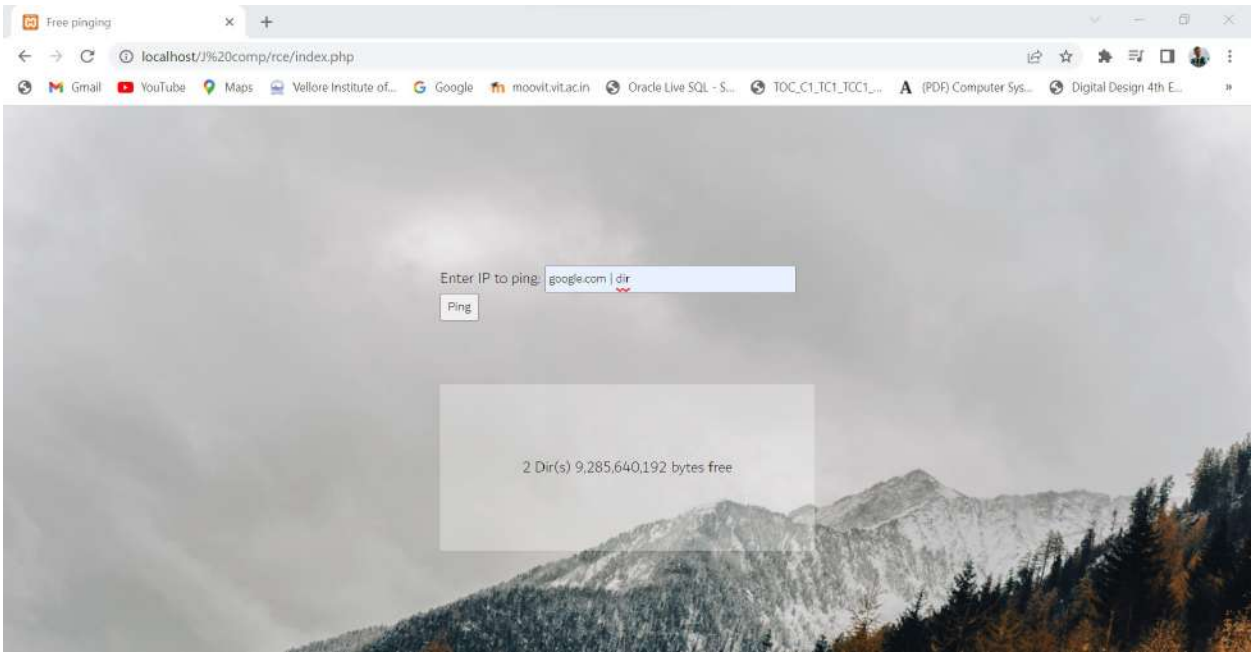
6.2.1 Code:

```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Free pinging</title>
        <style>
            * { font-family: 'Dubai Light', sans-serif; }
            body {
                margin: 0;
                display: flex;
                justify-content: center;
                align-items: center;
                background-image: url('bg.jpg');
                background-size: cover;
                background-repeat: no-repeat;
                width: 100%;
                height: 100%;
                flex-flow: column nowrap;
            }
            .divs {
                display: flex;
                justify-content: center;
                align-items: center;
                width: 30vw;
                height: 30vh;
            }
        </style>
    </head>
    <body>
        <div class="divs">
            <form method="post">
                Enter IP to ping:
                <input type="text" name="command" placeholder="Eg.
10.0.0.1">
                <input type="submit" value="Ping">
            </form>
        </div>
        <div class="divs" style="background-color: rgba(255, 255, 255,
0.3);">
```

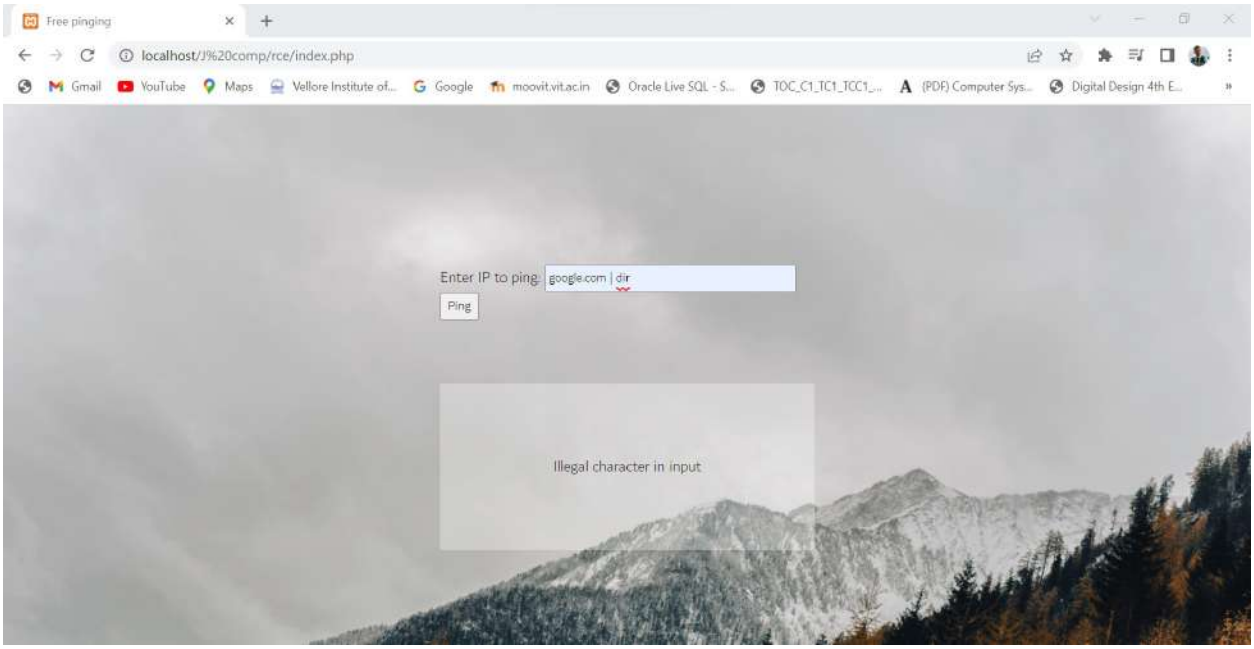
```
            <?php
                if(isset($_POST['command'])) {
                    if((strpos($_POST['command'], ';') == true) or
(strpos($_POST['command'], '|') == true)) {
                        echo "Illegal character in input";
                    } else {
                        $res = exec("ping ".$_POST['command']);
                        echo $res;
                    }
                }
            ?>
        </div>
    </body>
</html>
```

6.2.2 Output:

Attack Results:



Prevention Results:



6.3 Sensitive data exposure

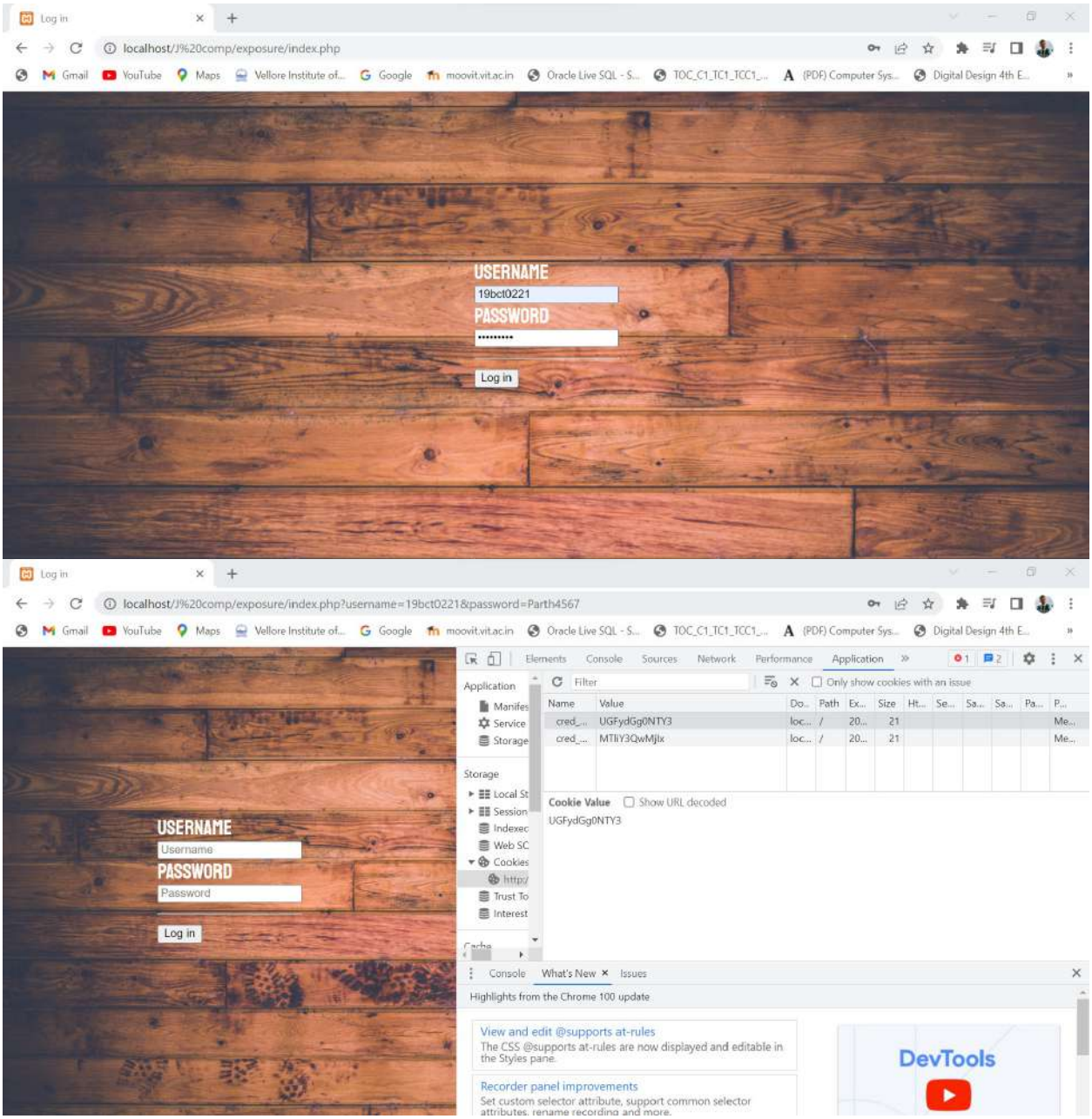
6.3.1 Code:

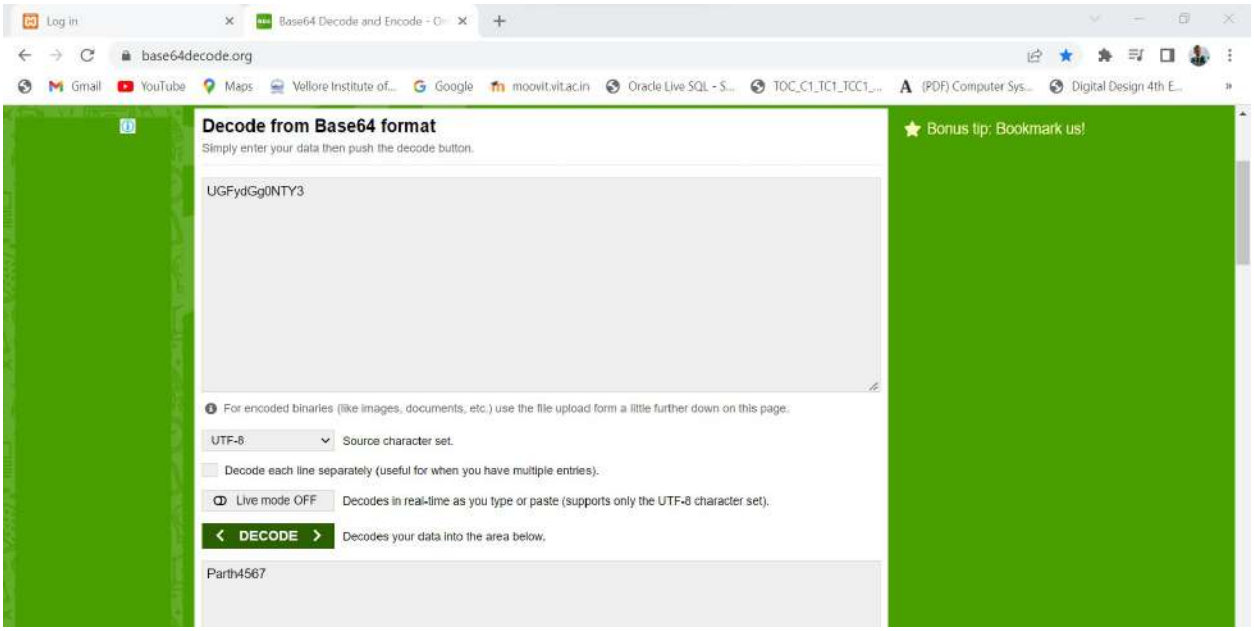
```
<?php
    if(isset($_COOKIE['cached_creds'])){
        // if($_COOKIE['cached_creds'] == 'y'){
        //     $us = base64_decode($_COOKIE['cred_user']);
        //     $pw = base64_decode($_COOKIE['cred_pass']);
        //     header("Location: mainpage.php?us=".$us."&pw=".$pw);
        // }
    } else if(isset($_GET['username']) && isset($_GET['password'])) {
        // setcookie("cred_user", base64_encode($_GET['username']),
time() + (60*30), "/");
        // setcookie("cred_pass", base64_encode($_GET['password']),
time() + (60*30), "/");
    }
?>
<html lang="en">
    <head>
        <meta charset="utf-8">
```

```
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link href="https://fonts.googleapis.com/css2?
family=Staatliches&display=swap" rel="stylesheet">
        <title>Log in</title>
        <style type="text/css">
            body {
                margin: 0;
                font-family: 'Staatliches', sans-serif;
                color: rgb(255, 255, 255);
                font-size: 24px;
                background-image: url('bg.jpg');
                background-repeat: no-repeat;
                background-size: cover;
                display: flex;
                justify-content: center;
                align-items: center;
            }
            #login {
                display: flex;
                justify-content: center;
                align-items: center;
                width: 50vw;
                height: 50vh;
            }
        </style>
    </head>
    <body>
        <div id="login">
            <form method="get">
                Username<br><input type="text" name="username"
placeholder="Username"><br>
                Password<br><input type="password" name="password"
placeholder="Password"><hr>
                <input type="Submit" value="Log in">
            </form>
        </div>
    </body>
</html>
```

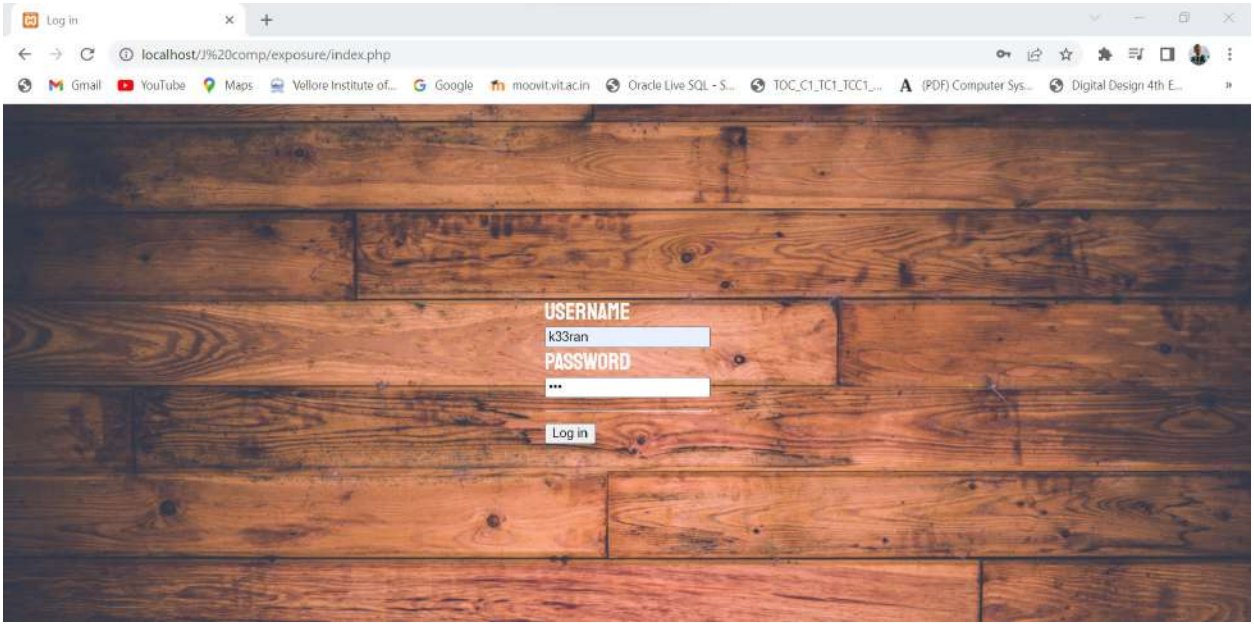
6.3.2 Output:

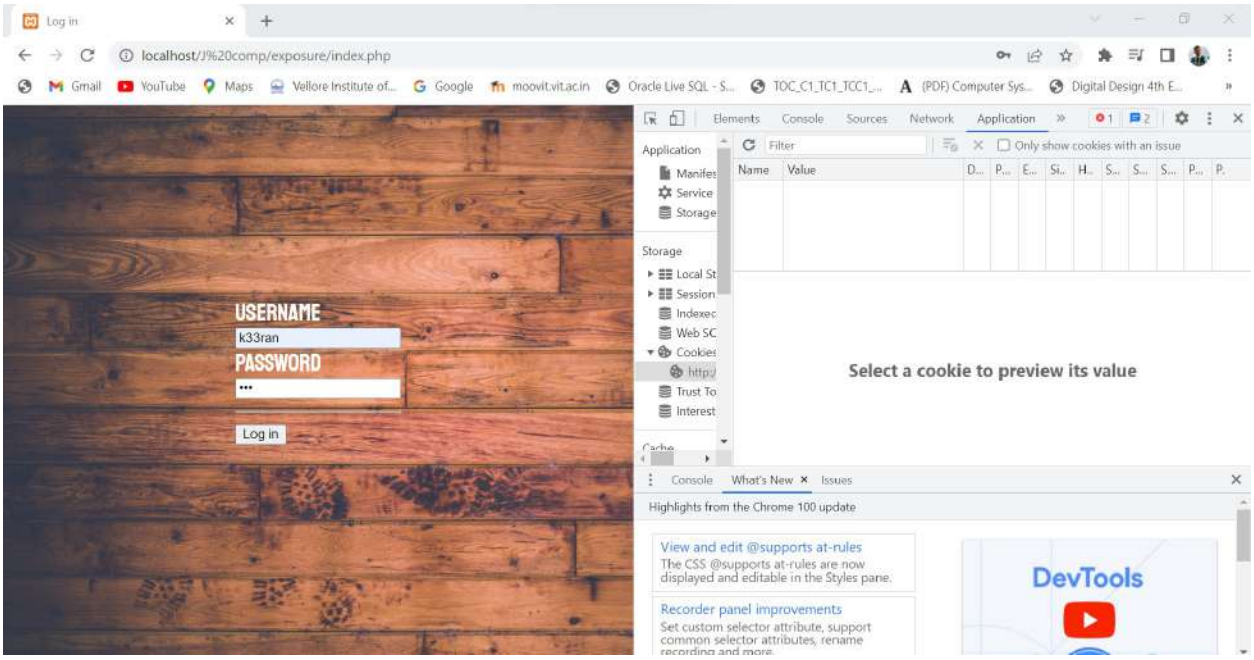
Attack Results:





Prevention Results:





6.4 Security misconfiguration

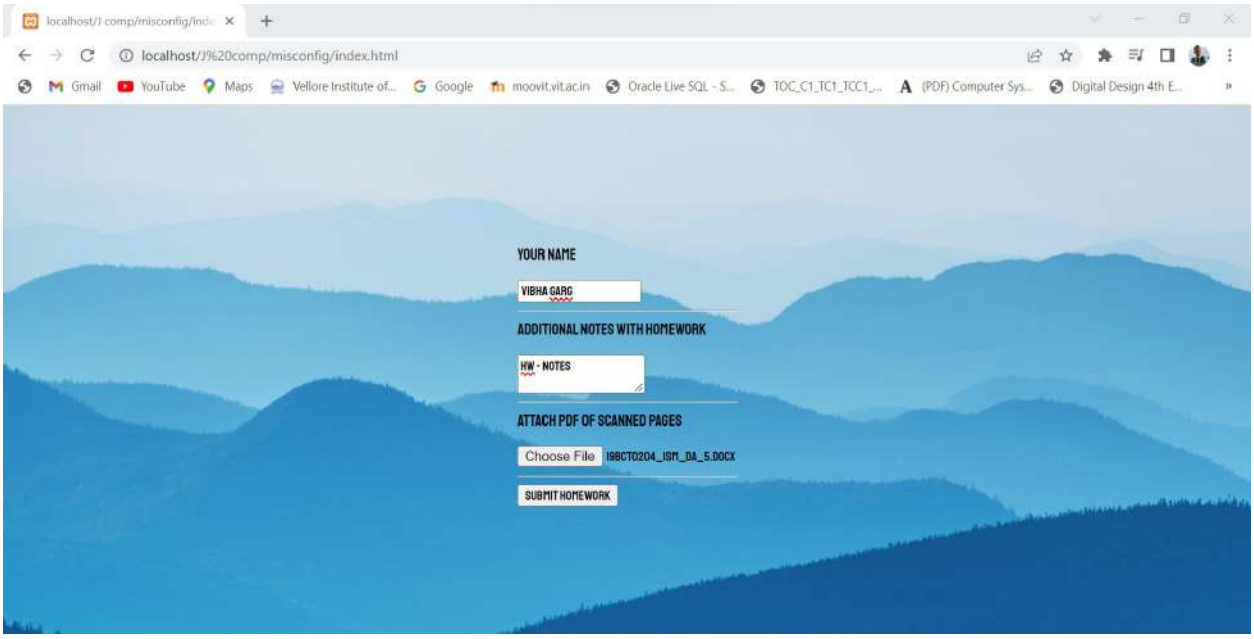
6.4.1 Code:

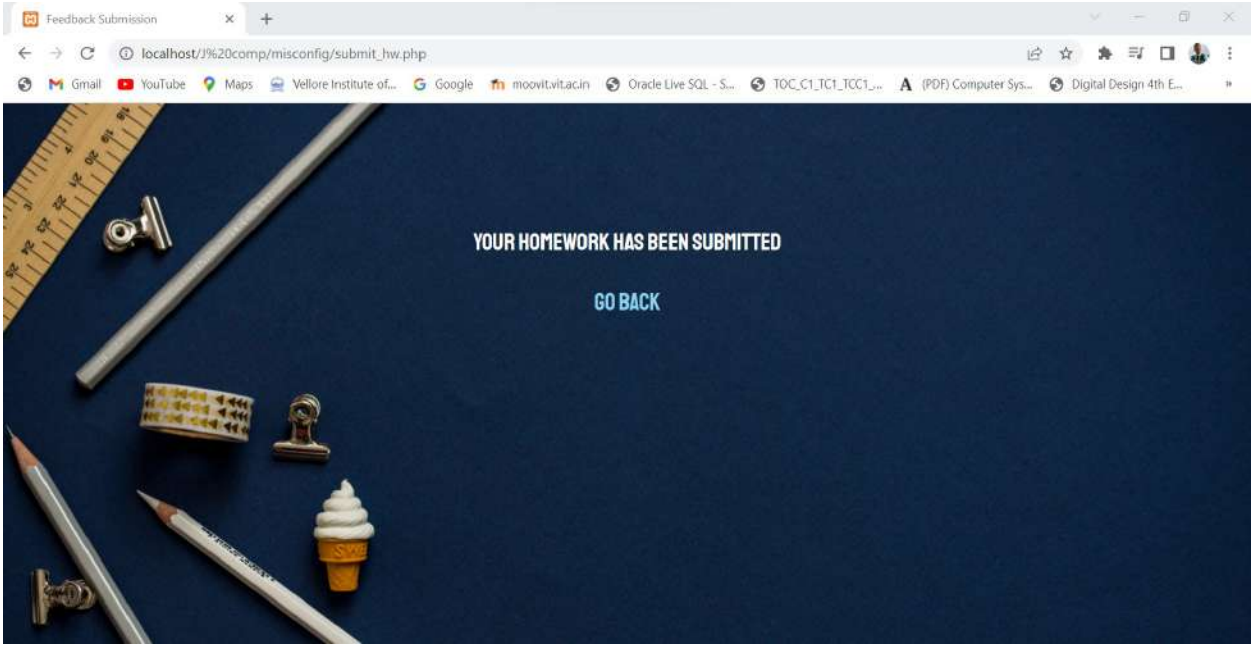
```
<?php
    error_reporting(0);
    if(!empty($_POST)){
        $name = $_POST["name"];
        $notes = $_POST["feedback"];
        $filename = "uploads/".rand(1111,9999).$_FILES["attachedFile"]
["name"];
        $file_ext = "uploads/".basename($_FILES["attachedFile"]["name"]);
        $ext = strtolower(pathinfo($file_ext,PATHINFO_EXTENSION));
        if($ext == 'pdf') {
            move_uploaded_file($_FILES["attachedFile"]["tmp_name"],
$filename);
            $writtenFeedback = fopen("uploads\\".$name.".txt","w");
            fwrite($writtenFeedback, "------------------\n");
            fwrite($writtenFeedback, $name."\n");
            fwrite($writtenFeedback, $filename."\n");
            fwrite($writtenFeedback, $notes."\n");
            fwrite($writtenFeedback, "------------------\n");
            echo "Your homework has been submitted";
        } else {
```

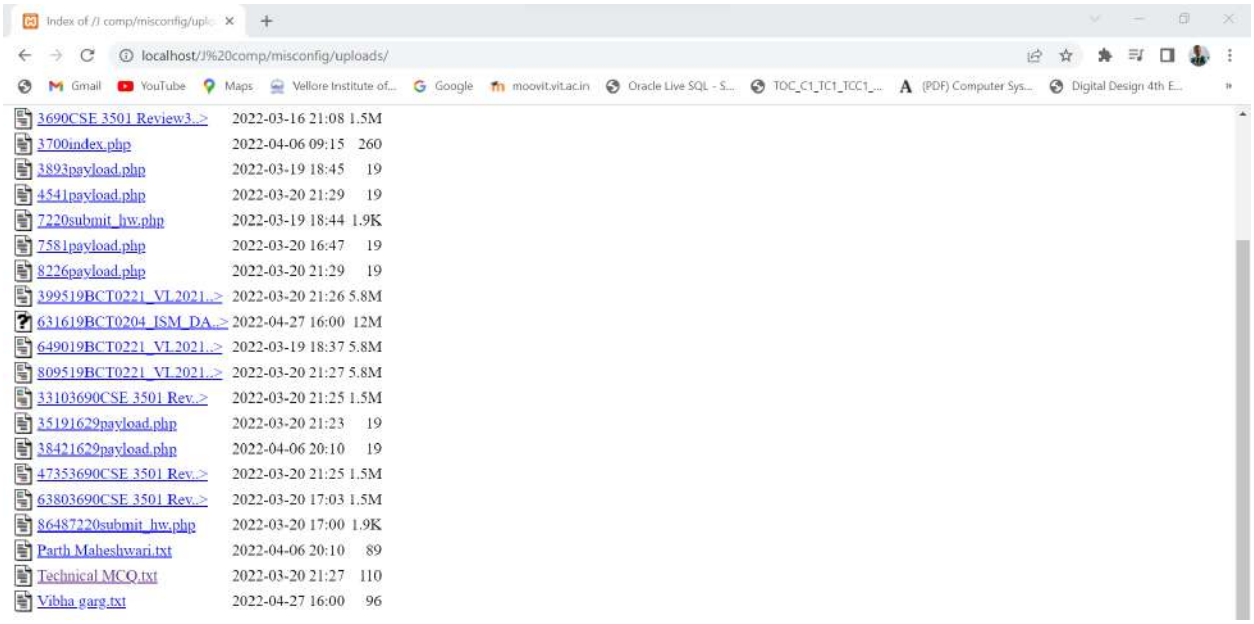
```
            echo "File not of PDF type";
        }
    }
?>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Feedback Submission</title>
        <link rel="preconnect" href="https://fonts.gstatic.com">
        <link href="https://fonts.googleapis.com/css2?
family=Staatliches&display=swap" rel="stylesheet">
        <style type="text/css">
            * {
                font-family: 'Staatliches', cursive;
                color: white;
                font-size: 24px;
            }
            body {
                background-image: url("bg2.jpg");
                background-size: cover;
                text-align: center;
                margin-top: 10%;
            }
            button {
                border: none;
                background-color: transparent;
                color: skyblue;
                cursor: pointer;
            }
        </style>
    </head>
    <body>
        <br><br>
        <button onclick="window.location.replace('index.html')">Go
back</button>
    </body>
</html>
```

6.4.2 Output:

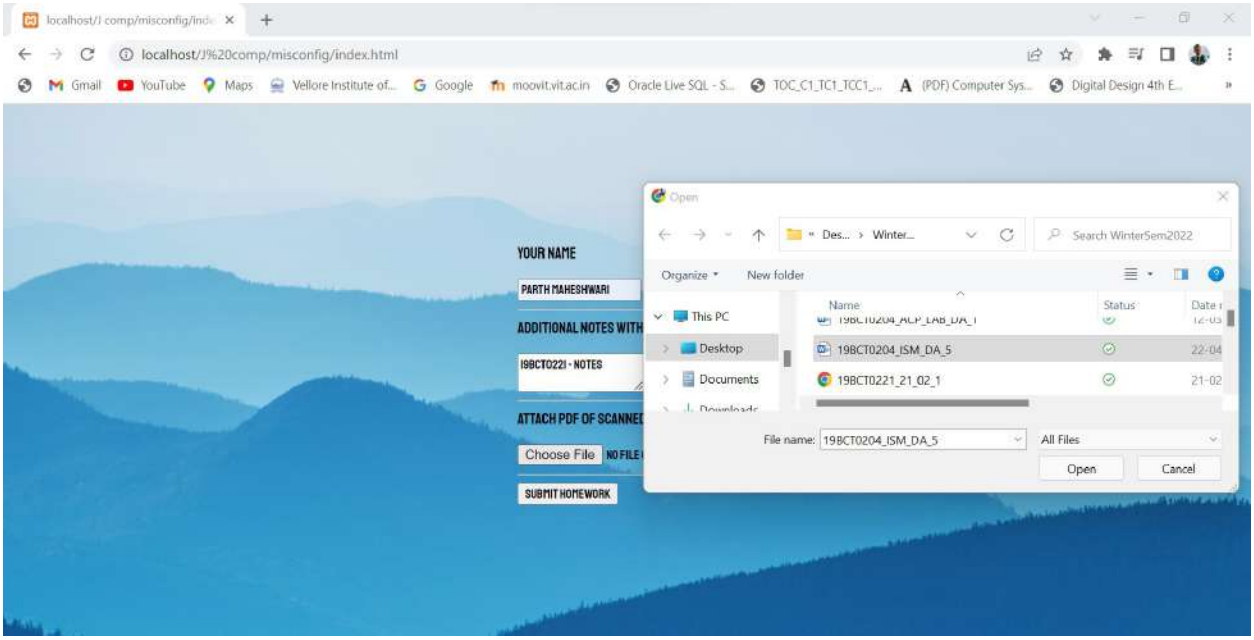
Attack Results:

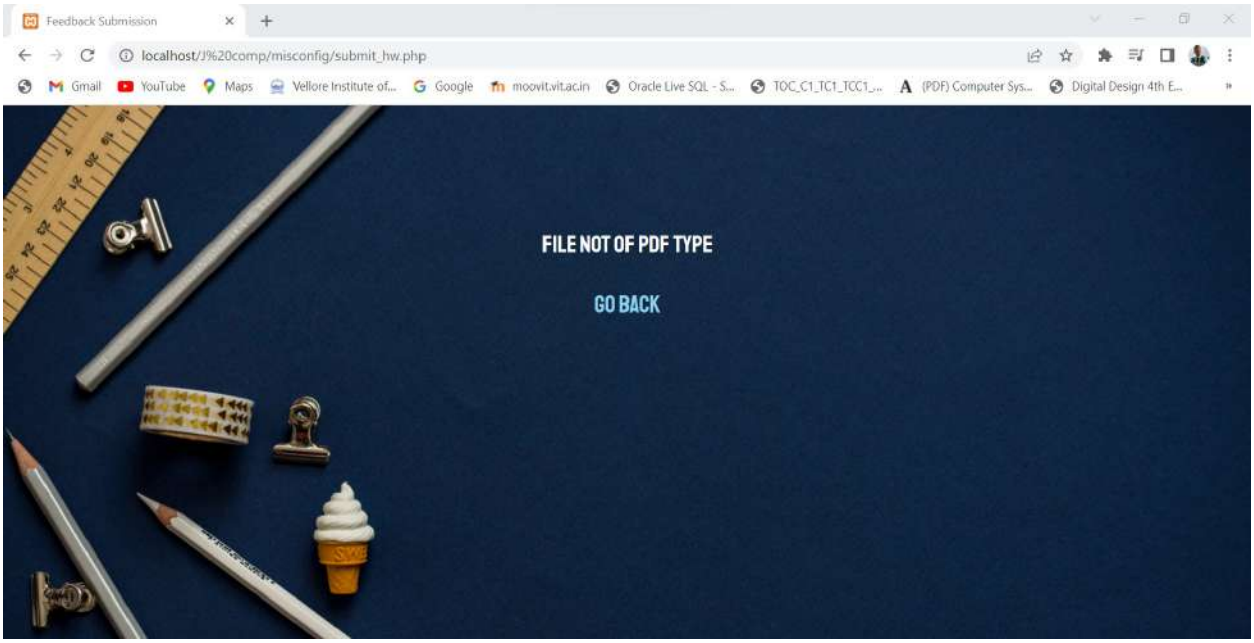

localhost/J%20comp/misconfig/index.html
YOUR NAME
VIBHA GARG
ADDITIONAL NOTES WITH HOMEWORK
HW - NOTES
ATTACH PDF OF SCANNED PAGES
Choose File
19BCT0204_ISM_DA_5.DOCX
SUBMIT HOMEWORK


Feedback Submission
localhost/J%20comp/misconfig/submit_hw.php
YOUR HOMEWORK HAS BEEN SUBMITTED
GO BACK



Prevention Results:





6.5 Broken Authentication

6.5.1 Code:

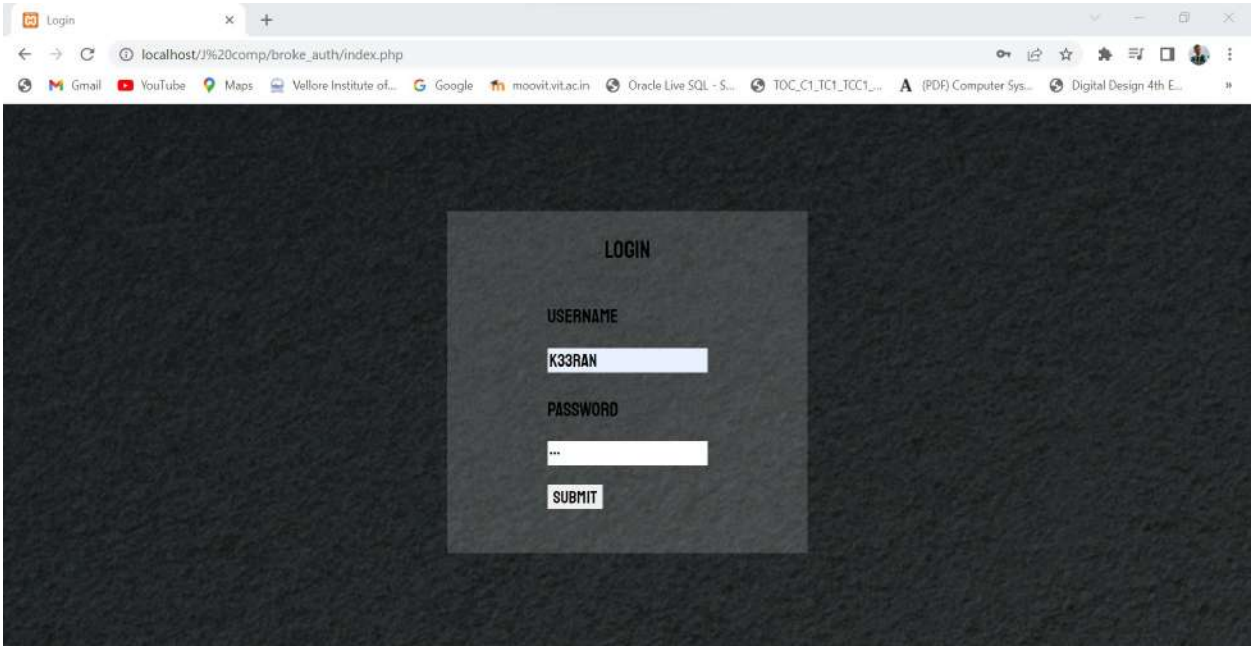
```php
<?php
    $us = '';
    $pw = '';
    $name = '';
    $phone = '';
    // if(isset($_COOKIE['username'])) {
    //     $us = $_COOKIE['username'];
    //     class usersDB extends SQLite3 {
    //         function __construct() {
    //           $this->open('users.db');
    //         }
    //     }
    //     $conn = new usersDB();
    //     if(!$conn) {
    //         echo "Connection to DB failed";
    //         exit;
    //     }
    //     $query = "SELECT * FROM users WHERE username='".$us."';";
    //     $res = $conn->query($query);
```

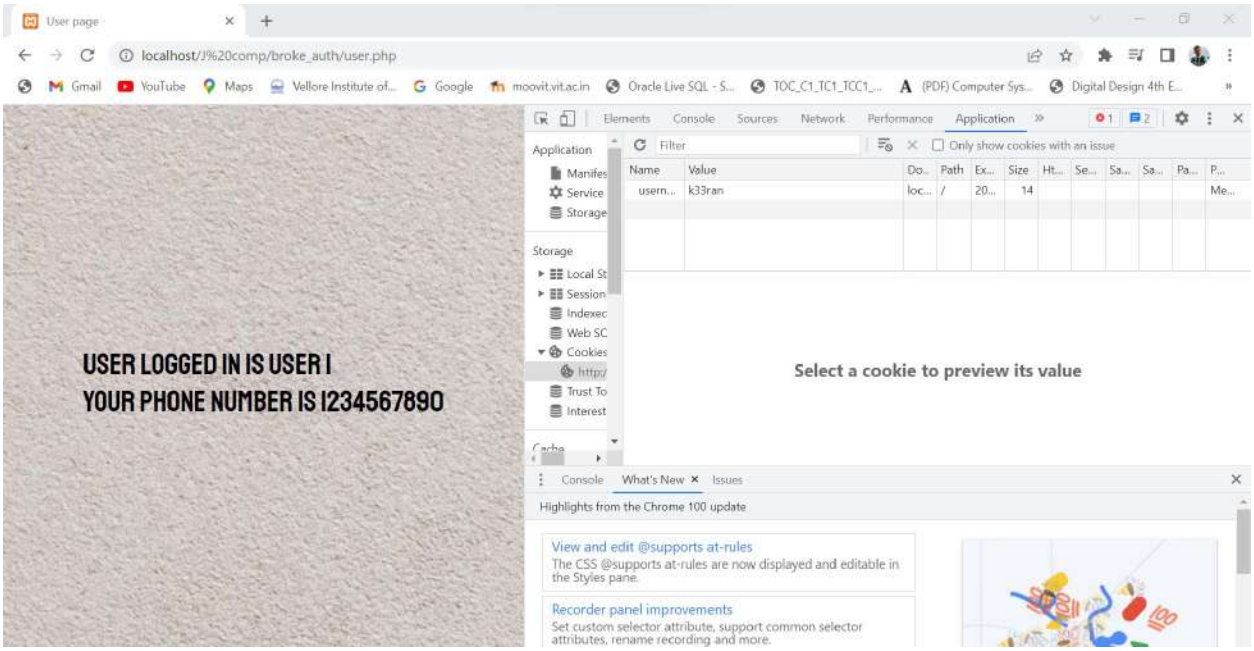
```
    //     $flag = TRUE;
    //     while($row = $res->fetchArray(SQLITE3_ASSOC)) {
    //         if($row['username'] == $us) {
    //             $flag = FALSE;
    //             $name = $row['name'];
    //             $phone = $row['phone'];
    //         }
    //     }
    //     $conn->close();
    // }
    // else
    if(isset($_POST['username']) && isset($_POST['password'])) {
        $us = $_POST['username'];
        $pw = $_POST['password'];
        class usersDB extends SQLite3 {
            function __construct() {
              $this->open('users.db');
            }
        }
        $conn = new usersDB();
        if(!$conn) {
            echo "Connection to DB failed";
            exit;
        }
        $query = "SELECT * FROM users WHERE username='".$us."' AND
password='".$pw."';";
        $res = $conn->query($query);
        $flag = TRUE;
        while($row = $res->fetchArray(SQLITE3_ASSOC)) {
            if($row['username'] == $us) {
                $flag = FALSE;
                $name = $row['name'];
                $phone = $row['phone'];
            }
        }
        $conn->close();
        if($flag) {
            echo "Credentials not found";
            exit;
        } else {
            setcookie("username", $us, time() + (60*30), "/");
        }
    } else {
```

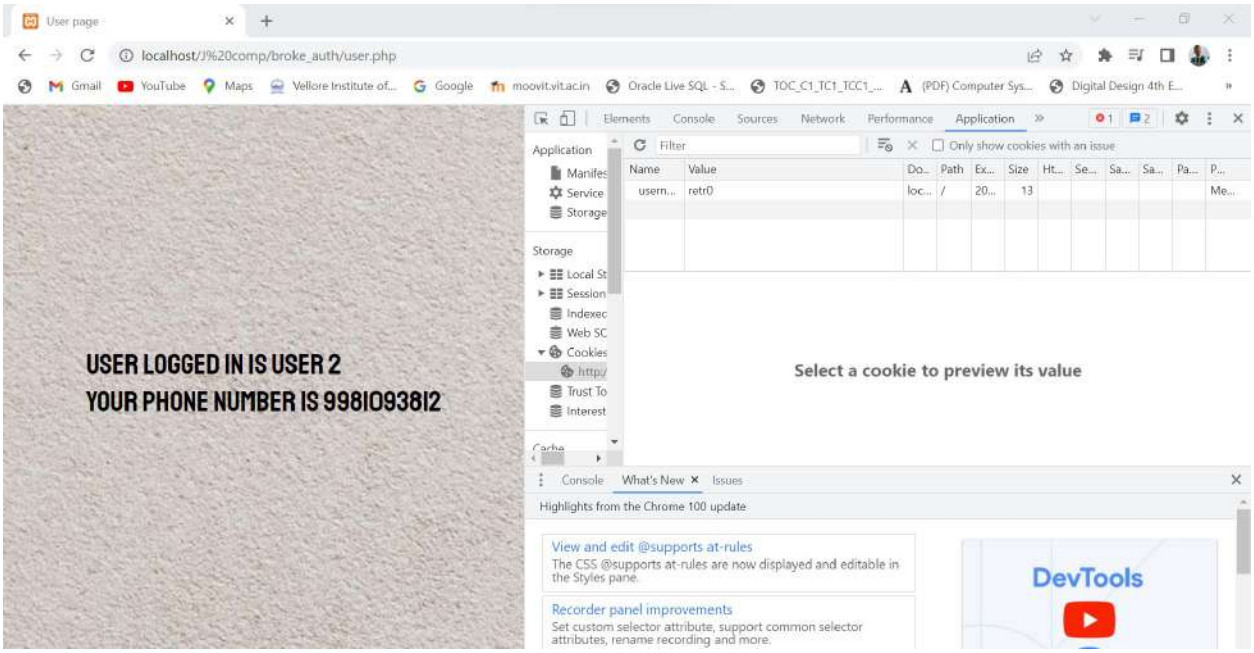
```
        header("Location: index.php");
    }
?>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link href="https://fonts.googleapis.com/css2?
family=Staatliches&display=swap" rel="stylesheet">
        <title>User page</title>
        <style type="text/css">
            body {
                background-image: url("bg2.jpg");
                background-position: cover;
                background-repeat: no-repeat;
                font-family: 'Staatliches', sans-serif;
                font-size: 30px;
                margin: 0;
                width: 100%;
                height: 100%;
            }
            #userdetails {
                width: 100vw;
                height: 100vh;
                display: flex;
                justify-content: center;
                align-items: center;
            }
        </style>
    </head>
    <body>
        <div id="userdetails">
            User logged in is <?php echo $name;?><br>
            Your phone number is <?php echo $phone;?>
        </div>
    </body>
</html>
```

6.5.2 Output:

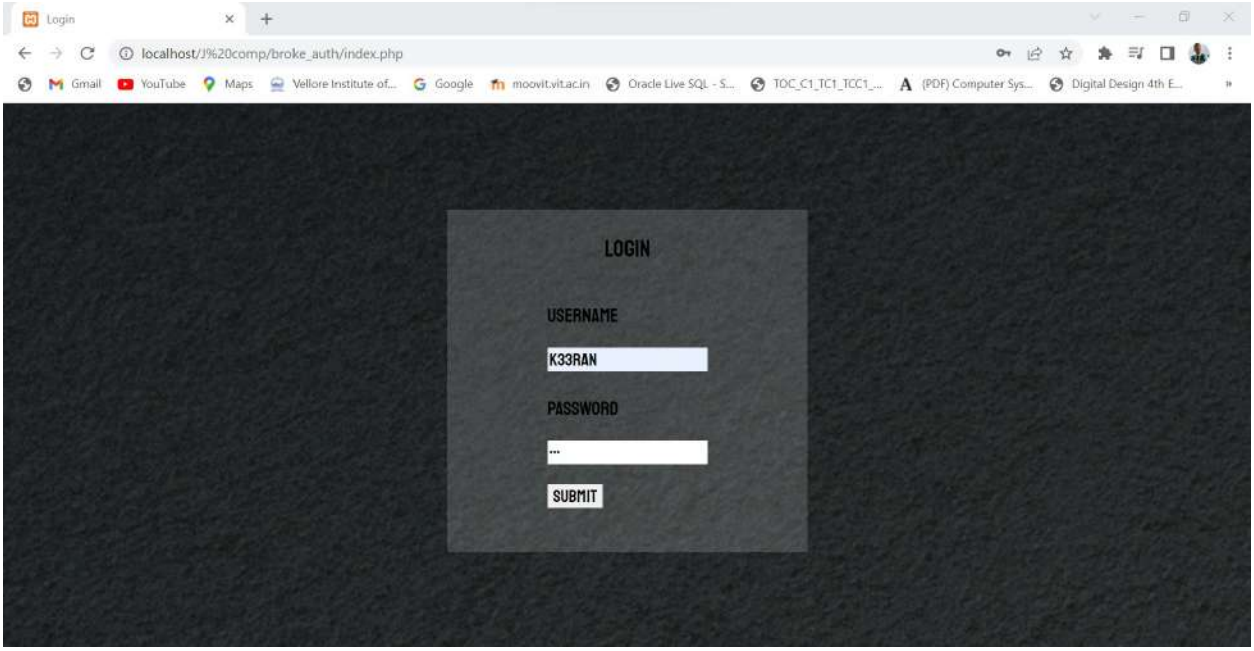
Attack Results:

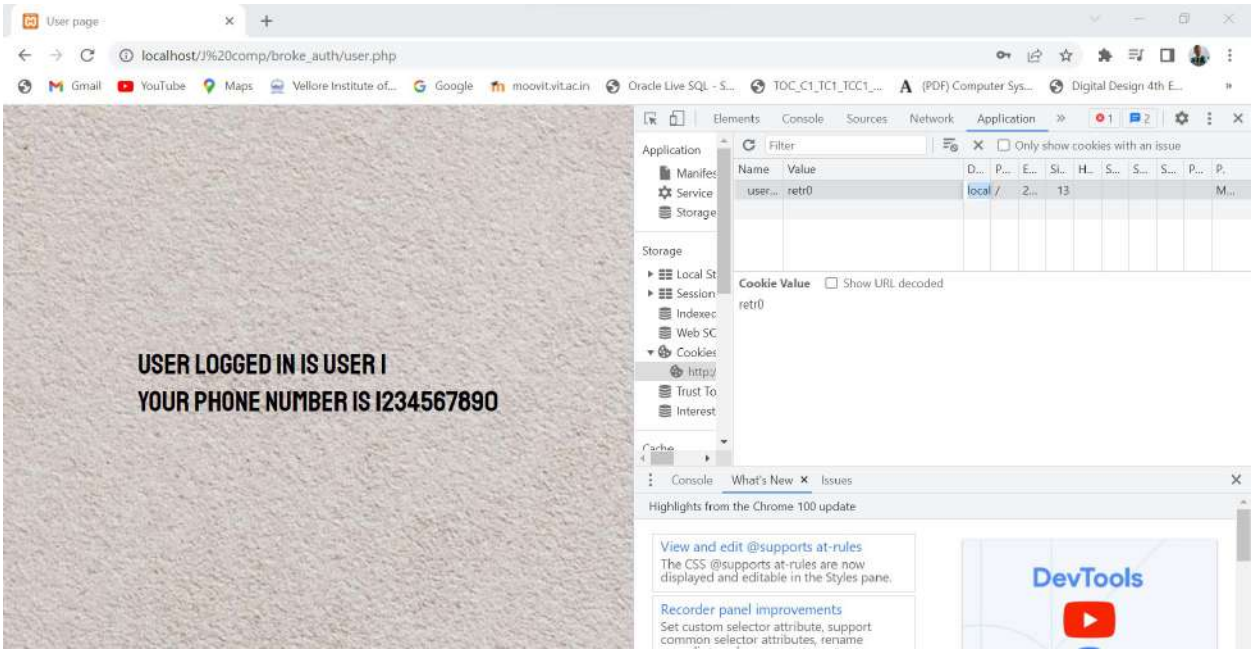


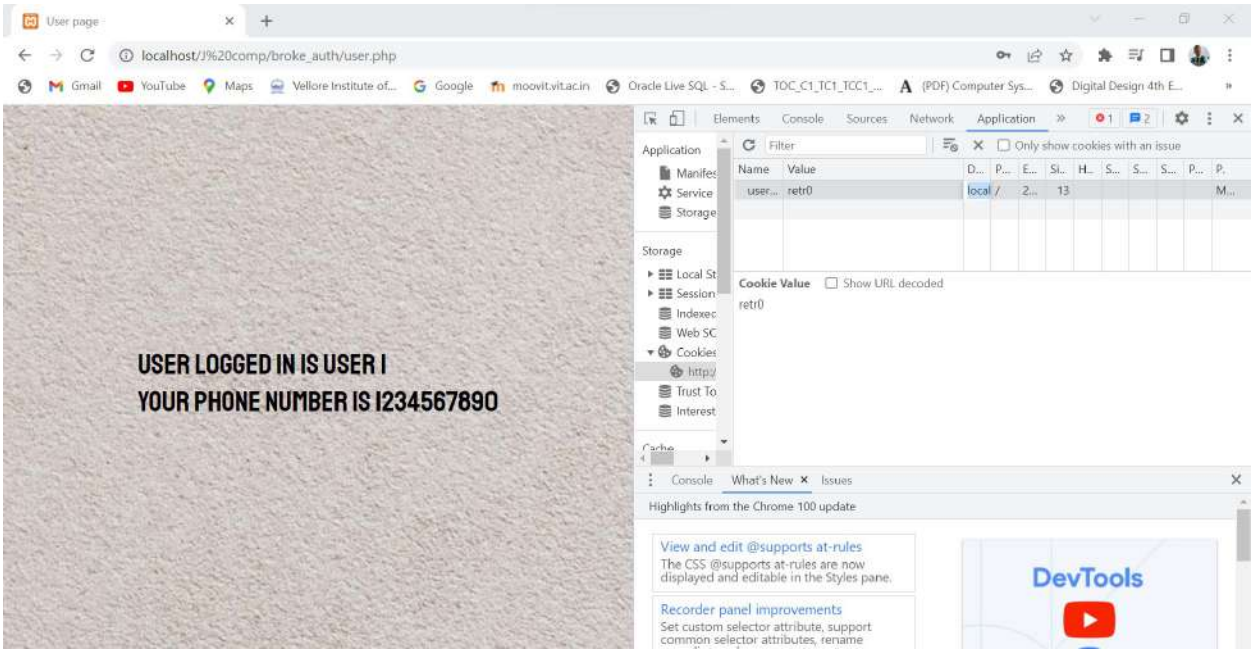




Prevention Results:




User page
localhost/J%20comp/broke_auth/user.php
USER LOGGED IN IS USER I
YOUR PHONE NUMBER IS I234567890
Elements
Console
Sources
Network
Application
Cookie Value
retr0
What's New
Issues
Highlights from the Chrome 100 update
View and edit @supports at-rules
The CSS @supports at-rules are now displayed and editable in the Styles pane.
Recorder panel improvements
DevTools


User page
localhost/J%20comp/broke_auth/user.php
USER LOGGED IN IS USER I
YOUR PHONE NUMBER IS I234567890
Elements
Console
Sources
Network
Application
Cookie Value
retr0
What's New
Issues
Highlights from the Chrome 100 update
View and edit @supports at-rules
The CSS @supports at-rules are now displayed and editable in the Styles pane.
Recorder panel improvements
DevTools

7. Results and Discussion

We deployed a web app on the target VM, and using the attacker machine we will demonstrate all the OWASP Top 5 vulnerabilities on the target. For this task we will use manual techniques for attacks like SQL injection and scanning weak points in the app. Then we will use tools for manual inspection like gobuster and Burpsuite for enumerating the website and exploiting vulnerabilities by modifying requests or responses. For our final task, the vulnerabilities found from the task above will be analyzed and corrected. The security patches will be kept in a record to compare the difference between the app before and after the assessment. We will then create a bug assessment to simplify the process of reviewing for our web app. The security fixes will also be demonstrated if needed. Thus we will present an analysis report of the above stated vulnerabilities and the process of preventing them in the developed web Application.

8. References:

Weblinks:

1. https://owasp.org/
2. https://www.rapid7.com/fundamentals/vulnerabilities-exploits-threats/
3. https://hdivsecurity.com/owasp-broken-authentication-and-session-management

Journals/Publications:

1. Kumi, S., Lim, C., Lee, S., Oktian, Y. and Witanto, E., 2022. Automatic Detection of Security Misconfigurations in Web Applications.
2. Ankit Shrivastava, Santosh Choudhary & Ashish Kumar. XSS Vulnerability Assessment and Prevention in Web Application.
3. Sandra Kumi, ChaeHo Lim, Sang-Gon Lee. Automatic Detection of Security Misconfigurations in Web Applications.
4. Md Maruf Hassan, Shamima Sultana Nipa, Marjan Akter, Rafita. Broken Authentication and Session Management Vulnerability: A Case Study of Web Application
5. Bhushan Trivedi, Jignesh Doshi. Sensitive Data Exposure Prevention using Dynamic Database Security Policy.