Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

ANALYSIS AND PREVENTION OF OWASP WEB APPLICATION SECURITY RISKS

PROJECT REPORT FOR:

INFORMATION SECURITY ANALYSIS AND AUDIT
CSE3501 J-COMPONENT
REVIEW 2

PRESENTED BY:

Varun Tiwari	19BCE2030
Ankita Mandal	19BCE0438
Argho Konar	19BCE0960
Devjyoti Karan	19BCE0690
Vibha Garg	19BCE0350

UNDER THE GUIDANCE OF

Dr. K Vimala Devi

ABSTRACT

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 mitigating these risks afterward.

PROBLEM STATEMENT AND OBJECTIVES

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 improve the code and test it again in order to get an in-depth knowledge of how dangerous weak web apps can be.

LITERATURE REVIEW

1. INJECTION ATTACKS:

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9319139

(i) Title: Effective Filter for Common Injection Attacks in Online Web Applications
Author: Santiago Ibarra-fiallos , Javier Bermejo Higuera , Monserrate Intriago-pazmiño , Juan Ramón Bermejo Higuera , Juan Antonio Sicilia Montalvo and Javier Cubo
Date of Publication: January 11, 2021

(ii) Problem and Objectives: According to the OWASP Top Ten Web Application Security Risks, the injection vulnerability is located at the beginning of the list, and it could be executed by internal and external users to an organization. This establishes a point of reference and an important warning to software developers. While developing a web application programmers should change the tendency to be more concerned with functional validation and give less value to secure programming. The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba . also known that there are web applications that are vulnerable to the most common injection attacks. These applications can not be rewritten. However, to stay online, these applications should be protected. 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.

(iii) Their Proposed methodology : 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. Consequently, it can be invoked by all the web applications deployed on the 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. Therefore, this article focuses on the analysis of vulnerabilities that can be mitigated by the validation and sanitization of input parameters in web applications. The procedure of deploying the proposed filter is detailed, specifying the sections and contents of the configuration file. In addition, the infrastructure for running the tests is described, including the setting of an attack tool, and the implementation of a controller. The attack tool is used as a security scanner for common injection attacks, and the controller is developed for routing the requests in two steps; first a request is addressed to the filter, and if it is valid, it will redirect to the web application itself. The proposal filter has been tested on three public as well as on a real private web application. An accuracy of 98.4% and an average processing time of 50 ms are achieved, based on which it is possible to conclude the proposed filter is highly reliable and does not require additional computational resources.

(iv) Limitations: The advance of the Web applications, as now it includes URL parameters, and with the time it is necessary to complement the development of the filter to protect this type of parameters under the methodology raised in the present work to efficiently prevent common injection attacks.

2. XSS ATTACKS:

https://ieeexplore.ieee.org/abstract/document/7877529

(i) Title: XSS Vulnerability Assessment and Prevention in Web Application
Author: Ankit Shrivastava, Santosh Choudhary & Ashish Kumar
Date of publication: 12 July, 2019

(ii) Problem and Objectives: Technique Implemented: XSS.
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. Once the vulnerability is oppressed, an intruder advances intended access of the authenticated user's web browser and may perform session hijacking, cookie-stealing, malicious redirection and malware spreading. As prevention against such attacks, it is essential to implement security measures that certainly block third party intrusion. Recently the most dangerous attacks are reflected and DOM based cross-site scripting attacks because in both cases attacker attacks using server side scripting and do forgery over the network,it is hard to detect and therefore it must be prevented. Vulnerabilities of websites are exploited over the network through web requests using GET and POST methods.

(iii) Their Proposed methodology : 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 researchers 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.

(iv) Limitations: In the existing approach normally developers use javascript validation or user input filter to prevent client side malicious inputs, for the output they use escape method and sanitation method. Some more secure web applications use application level firewalls to filter user requests. But still, we are not able to stop XSS attacks.

3. SECURITY MISCONFIGURATION ATTACKS:

https://link.springer.com/chapter/10.1007/978-981-15-7990-5_8

(i) Title: Automatic Detection of Security Misconfigurations in Web Applications
Authors: Sandra Kumi, ChaeHo Lim, Sang-Gon Lee
Date of Publication: 28 November 2020

(ii) Problem and Objectives: Improper configuration of web applications or servers can lead to various security flaws. Security misconfiguration is one of the top 10 OWASP Web Application Security Risks. It is a critical risk in web applications that web developers need to focus on. 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 exploitation of security misconfiguration vulnerabilities can lead attackers to exploit more critical vulnerabilities and also ultimately compromise an application. This paper presents a tool that automatically scans web applications to detect security misconfigurations before deployment. The tool uses black-box testing (Dynamic Application Security Testing) to detect misconfiguration vulnerabilities in web applications. DAST technologies are designed to detect conditions indicative of a security vulnerability in an application in its running state. They run on operating code to detect issues with interfaces, requests, responses, scripting, sessions, data injection and authentication.

(iii) Their Proposed methodology: 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. The tool is implemented in Perl and deployed on a PC and a server. It is able to detect security misconfiguration vulnerabilities like Internal server error, Exception error, Source code disclosure, PHP information leaked, POST method allowed and Script error. A comparative analysis was carried out with other tools which operate in a similar way.

(iv) Limitations: Security misconfigurations vulnerabilities like cookies without secure flag set, directory listing, weak password, heap buffer overflow, stack trace disclosure and unicode transformation issues are not detected by the tool. There is a possibility of detecting a wider range of vulnerabilities. It took 102 seconds to scan a web application with 61 URLs. A faster approach might be possible.

4. BROKEN AUTHENTICATION:

https://www.researchgate.net/publication/325961962_Broken_Authentication_and_Session_Management_Vulnerability_A_Case_Study_Of_Web_Application

i) Title: Broken Authentication and Session Management Vulnerability: A Case Study Of Web Application
Authors: Md Maruf Hassan, Shamima Sultana Nipa, Marjan Akter, Rafita Haque
Date of Publication: April 2018

ii) Problems and Objectives: 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 preventive guidelines were also provided through a solution of those attacks. An examination was performed to detect the existence of various web vulnerabilities. The survey was performed on 110 websites and revealed the cause of application-layer vulnerabilities. Three main factors were also identified for the above reason that includes lack of experience, lack of knowledge in web security programming, and neglect of using the encryption methods. A study performed on root cause analysis to detect the Session Management and Broken Authentication vulnerabilities and prescribed solutions have been given to reduce the recurring attack of the web application. The process of identifying the Broken Authentication vulnerability, attack procedure, and prescribed guidelines was discussed to protect the web-based system from the intruder. A technique Nemesis, used for preventing access control vulnerabilities and Exploiting Authentication problems on web applications is presented in the paper. The author implemented Nemesis through a tool by which the developer can control the given vulnerabilities in a small amount of time. The study explaining the types of Broken Exploiting Authentication problem and Session
Management attacks of web applications.

iii) Their Proposed Methodology: We can prevent BROKEN AUTHENTICATION AND SESSION MANAGEMENT VULNERABILITY using the following:
Session ID Life Cycle
Session Reset
Session Expiration
Cookies
Session Attacks Detection
Client-Side Defenses 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. A secure banking portal, for example, must automatically log out the user after a few minutes to avoid any risks of hijacked session IDs
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.

iv) Limitations: It also becomes necessary that users are adequately trained and educated about the potential risks of broken authentication through phishing attacks or weak passwords. Organizations must employ strong Cybersecurity measures in line with the continually evolving global standards. They must ensure the prevention of broken authentication by all means possible. Without adequate security controls, hackers rely on known mechanisms to access crucial system accounts, leading to social security fraud, money laundering, and identity theft, depending on the application's domain.

5. SENSITIVE DATA EXPOSURE:

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.800.1832&rep=rep1&type=pdf

i) Title: Sensitive Data Exposure Prevention using Dynamic Database Security Policy
Authors: Jignesh Doshi (LJ Institute of Management Studies, Ahmedabad,Gujarat, India) and Bhushan Trivedi (GLS Institute of Computer Technology, Ahmedabad,Gujarat, India)
Date of Publication: 15, November 2014

ii) Problems and Objectives: Attackers use different paths for executing their tasks. A wide range of technologies is used for web application development as a result, it is very easy to attack on applications. One attack can have several impacts on business. Application software with security leakages is dejecting our financial, defense, energy, healthcare and other critical infrastructure. Our digital infrastructure gets increasingly complex and interconnected. As a result, it is most critical to secure applications. It is observed that there are many leakages in the security of web applications. Five attacks out of top ten attacks are done using Structured Query Language (SQL). Database attacks mostly affect Data Theft, Data manipulation and bypassing user authentication. Our study focus is to prevent sensitive data exposure. The authors have proposed dynamic database security policy to prevent sensitive data exposure using Oracle database. Key Problems with existing Sensitive data exposure risk solutions are summarized as : Need complex logic to store and retrieve plain text, Works same for all users, Requires more space to store small data, Performance is a major concern, Requires more efforts and skill to deal with techniques. In most places no encryption is done or if done the key generated is weak. The research problem is defined as: Which concepts are to be implemented to prevent sensitive data theft even at column level? The Objective is to Prevent Sensitive Data loss without performance loss.

iii) Their Proposed Methodology: 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 catalog, creating custom security functions/policies and configuring custom policies using fine grained access control mechanism of oracle database. SDF provides a highly configurable environment for application development. Users can use it to create even for multiple applications. Such a setup does not require any additional infrastructure and can be used along with existing built-in security techniques (like MAC, DAC and RBAC) in application development to prevent database security.

iv) Limitations: Whenever any query is executed, the fine grain policy will check whether any security policy is set on objects used in the query, If yes, it will modify the query and run or else run the query as it is . If no security policy based objects were found in the query, it will execute the query as it is. Thus this filter prevents sensitive data exposure only by filtering the queries entered.

OWASP TOP 10 VULNERABILITIES:

1. **Injection:**
   Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
   Preventing injection requires keeping data separate from commands and queries.
   The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).

2. **Broken Authentication:**
   Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
   One of the prevention methods is to implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential reuse attacks.

3. **Sensitive data exposure:**
   Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

4. **XML External Entities:**
   Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

5. **Broken Access Control:**
   Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or

data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

6. **Security Misconfiguration:**
   Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

7. **Cross-Site Scripting XSS:**
   XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

8. **Insecure Deserialization:**
   Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

9. **Using Components with Known Vulnerabilities:**
   Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

10. **Insufficient Logging & Monitoring:**
    Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

OUR PROPOSED METHODOLOGY

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.
For our second review, we will deploy a web app on the target VM, and using the attacker machine we will demonstrate all the OWASP Top 10 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.

MODULES IDENTIFIED

- ❖ Cross Site Scripting
- ❖ Remote Code Execution
- ❖ Sensitive Data Exposure
- ❖ Security Misconfiguration
- ❖ Broken Authentication

CODE

CROSS SITE SCRIPTING (XSS)

reflected.php

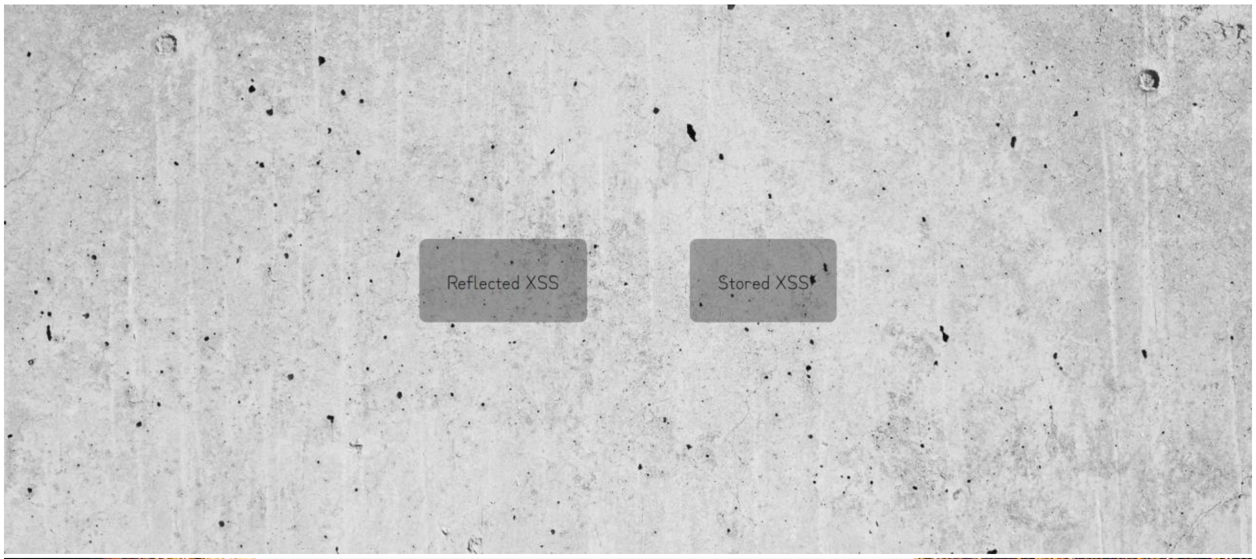
```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link
href="https://fonts.googleapis.com/css2?family=Zen+Kurenaido&display=swap"
rel="stylesheet">
        <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 $_GET['m']; } ?>
            </div>
        </div>
    </body>
</html>
```

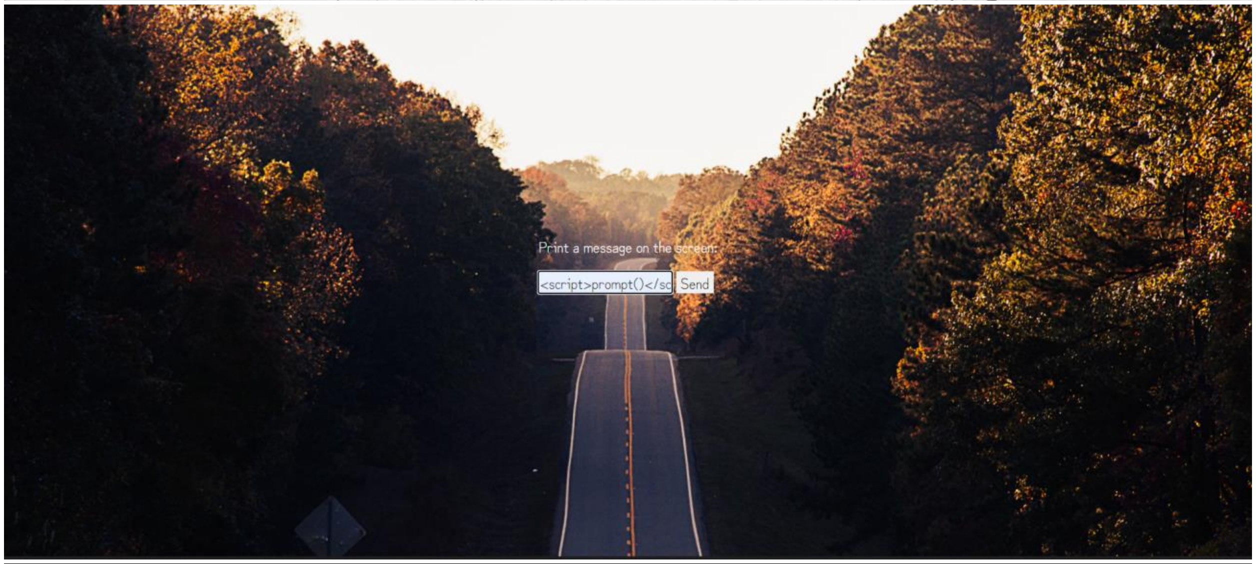
stored.php

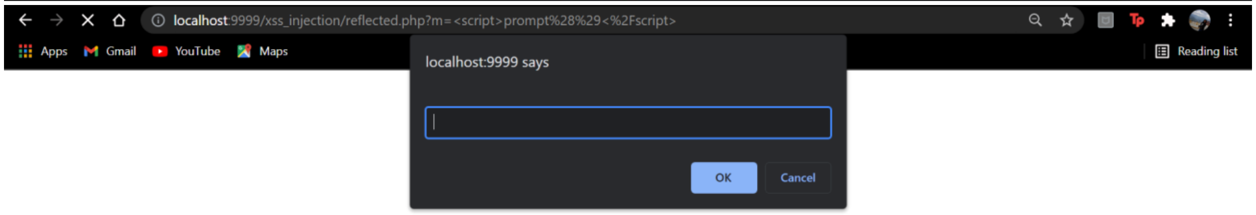
```
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>
        <link
href="https://fonts.googleapis.com/css2?family=Zen+Kurenaido&display=swap"
rel="stylesheet">
        <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 = $res->fetchArray(SQLITE3_ASSOC)) {
                            echo "<tr><td>:</td><td
class=\"com\">".$row['comment']."</td></tr>";
                        }
                        $conn->close();
                    ?>
                </tbody>
            </table>
        </div>
    </body>
</html>
```

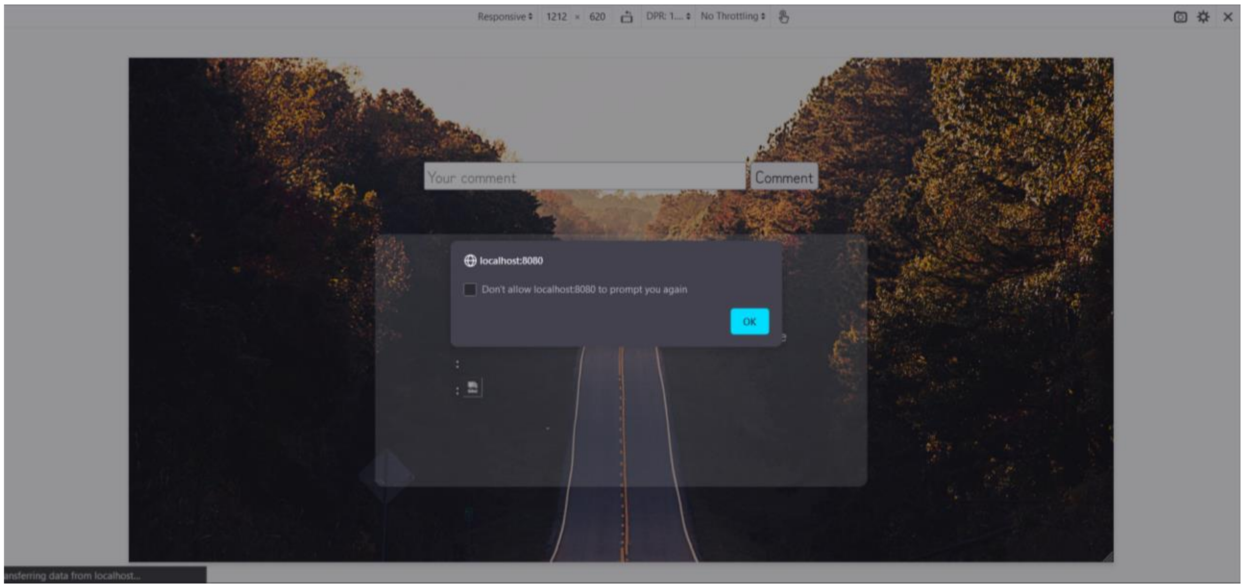
OUTPUT SCREENSHOT






localhost:9999/xss_injection/reflected.php?m=<script>prompt%28%29<%2Fscript>

localhost:9999 says



localhost:8080

Don't allow localhost:8080 to prompt you again

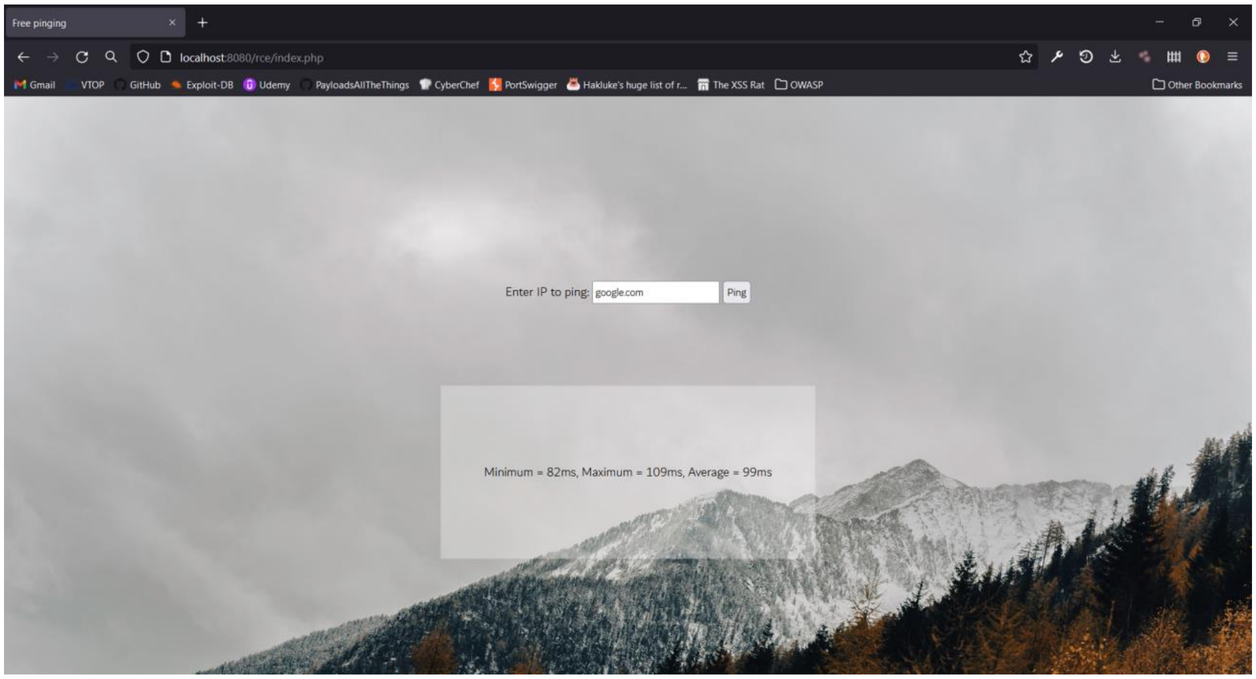
ansferring data from localhost...

CODE

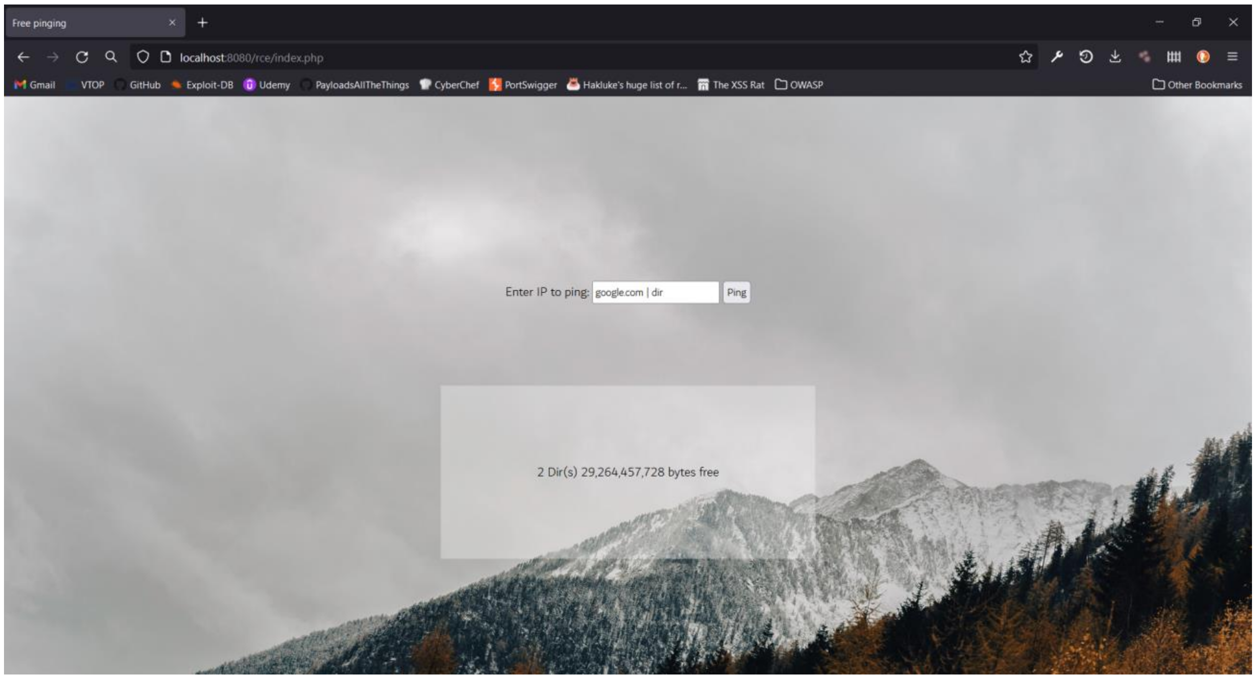
REMOTE CODE EXECUTION

```
<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'])) {
                    $res = exec("ping ".$_POST['command']);
                    echo $res;
                }
            ?>
        </div>
    </body>
</html>
```

OUTPUT SCREENSHOT



```
Minimum = 82ms, Maximum = 109ms, Average = 99ms
```


Free pinging
localhost:8080/rce/index.php
Enter IP to ping: google.com | dir
Ping

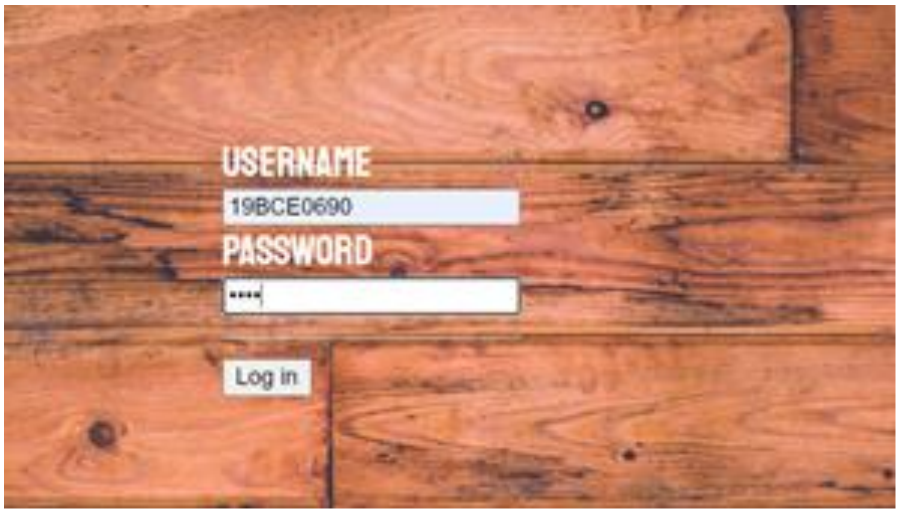
```
2 Dir(s) 29,264,457,728 bytes free
```

CODE

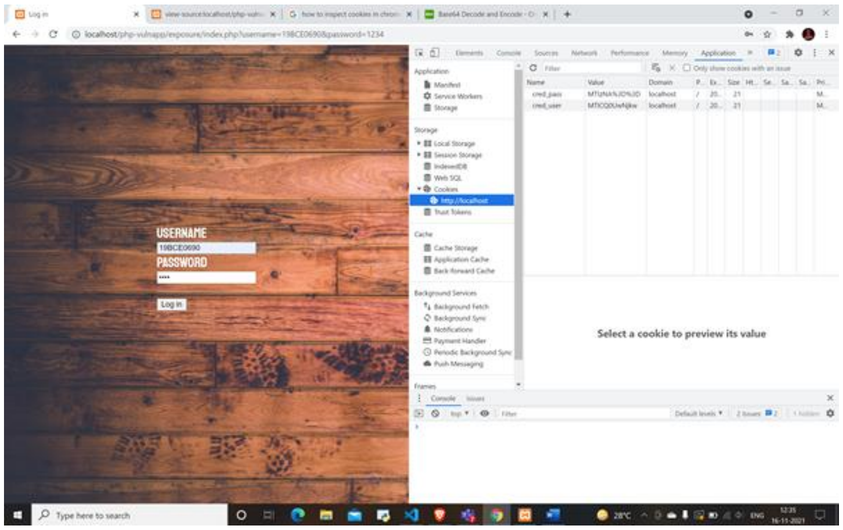
SENSITIVE DATA EXPOSURE

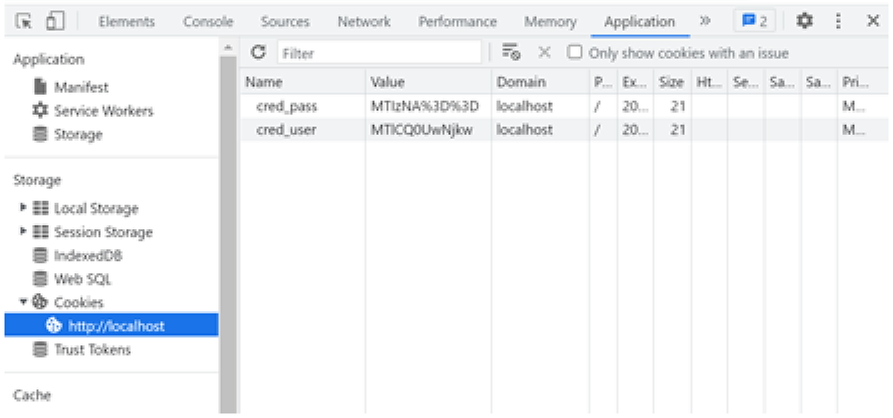
```
<?php
    if(isset($_COOKIE['cached_creds'])){
        if($_COOKIE['cached_creds'] == 'y'){
            $us = base64_decode($_COOKIE['cred_user']);
            $pw = base64_decode($_COOKIE['cred_pass']);
            header("Location: mainpage.php?us=".$us."&pw=".$pw);
        }
    } else if(isset($_GET['username']) && isset($_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>
```

OUTPUT SCREENSHOT

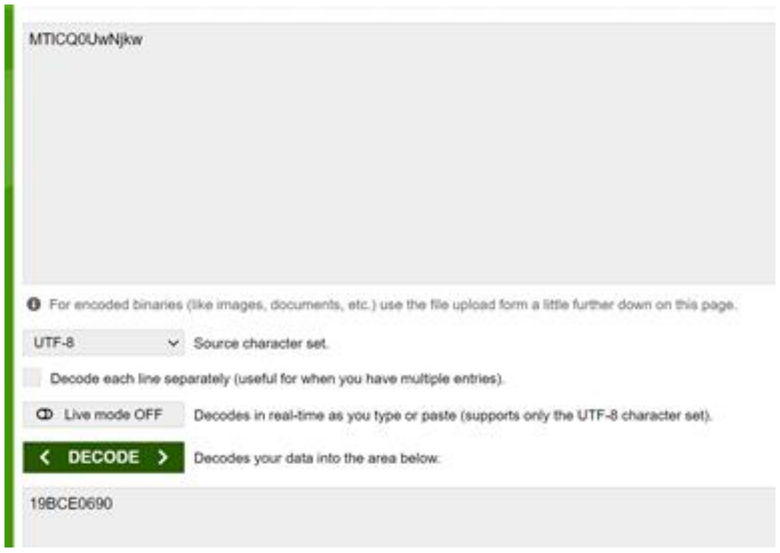


USERNAME
19BCE0690
PASSWORD
••••


Log in
localhost/php-vulnapp/exposure/index.php?username=19BCE0690&password=1234
USERNAME
19BCE0690
PASSWORD
Log in
Select a cookie to preview its value



Name	Value	Domain	P...	Ex...	Size	Ht...	Se...	Sa...	Sa...	Pri...
cred_pass	MTIzNA%3D%3D	localhost	/	20...	21					M...
cred_user	MTlCQ0UwNjkw	localhost	/	20...	21					M...


MTlCQ0UwNjkw

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

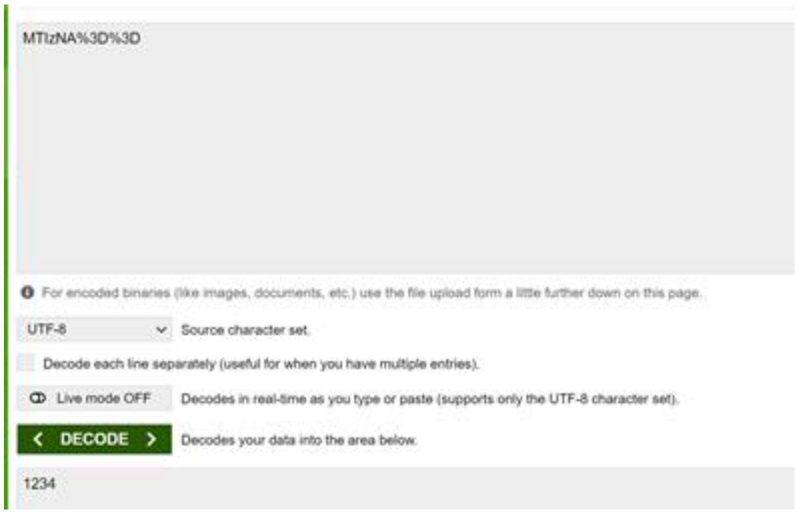
UTF-8 Source character set.

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

19BCE0690


MTIzNA%3D%3D

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

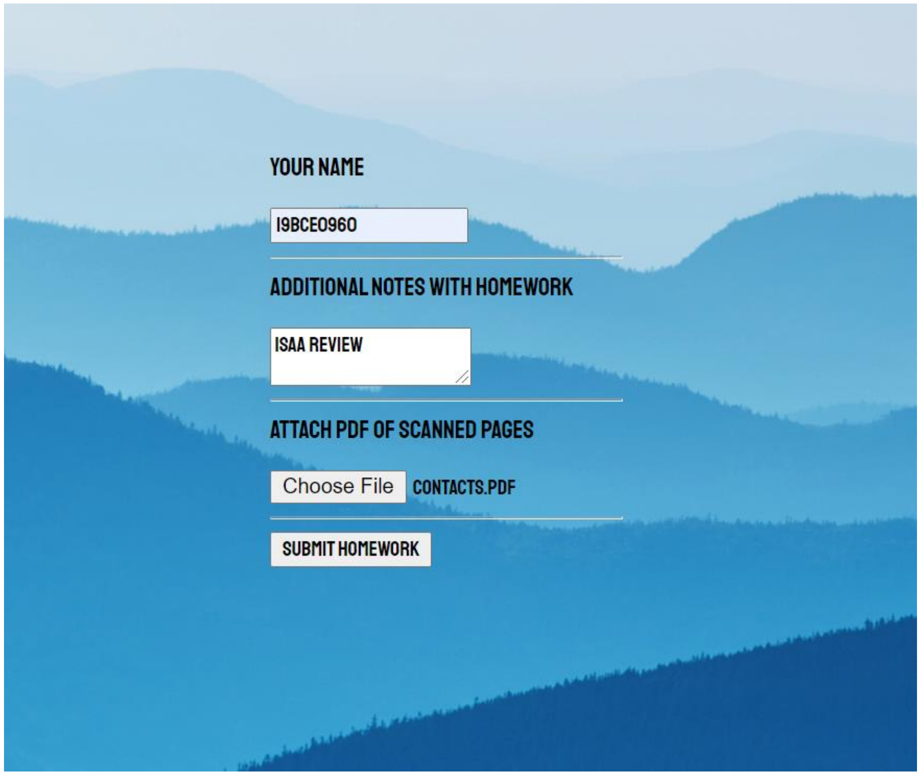
1234

CODE

SECURITY MISCONFIGURATION

```
<?php
    error_reporting(0);
    if(!empty($_POST)){
        $name = $_POST["name"];
        $notes = $_POST["feedback"];
        $filename =
"uploads/".rand(1111,9999).$_FILES["attachedFile"]["name"];
        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");
    }
?>
<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>
        Congrats! You did the homework<br><br>
        <button onclick="window.location.replace('index.html')">Go
back</button>
    </body>
</html>
```

OUTPUT SCREENSHOT



YOUR NAME

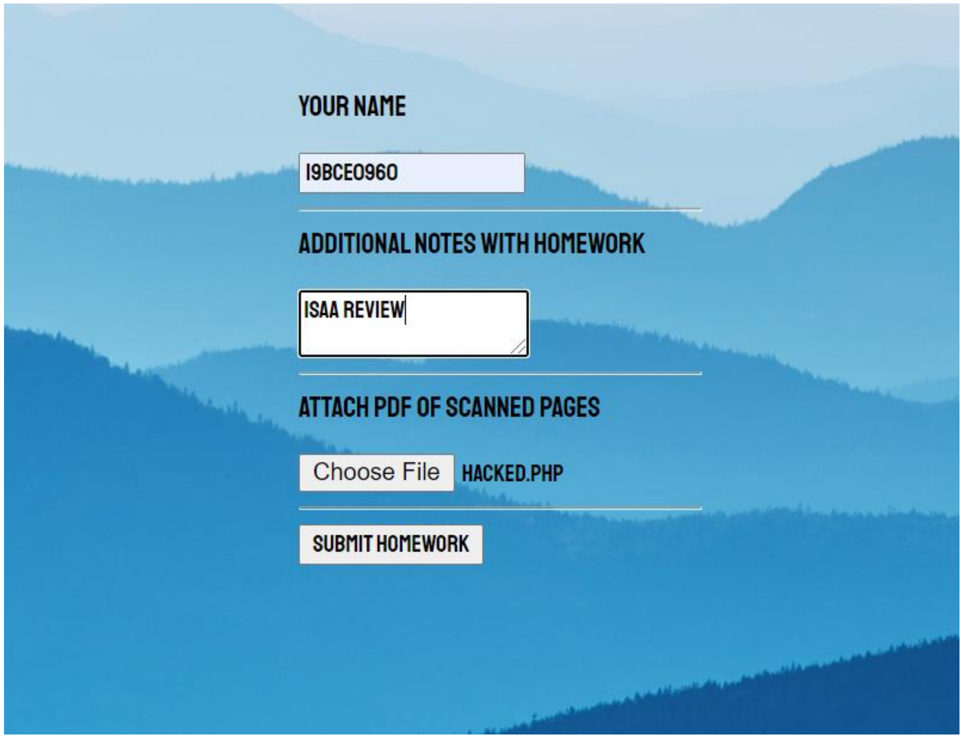
I9BCE0960

ADDITIONAL NOTES WITH HOMEWORK

ISAA REVIEW

ATTACH PDF OF SCANNED PAGES

CONTACTS.PDF



YOUR NAME

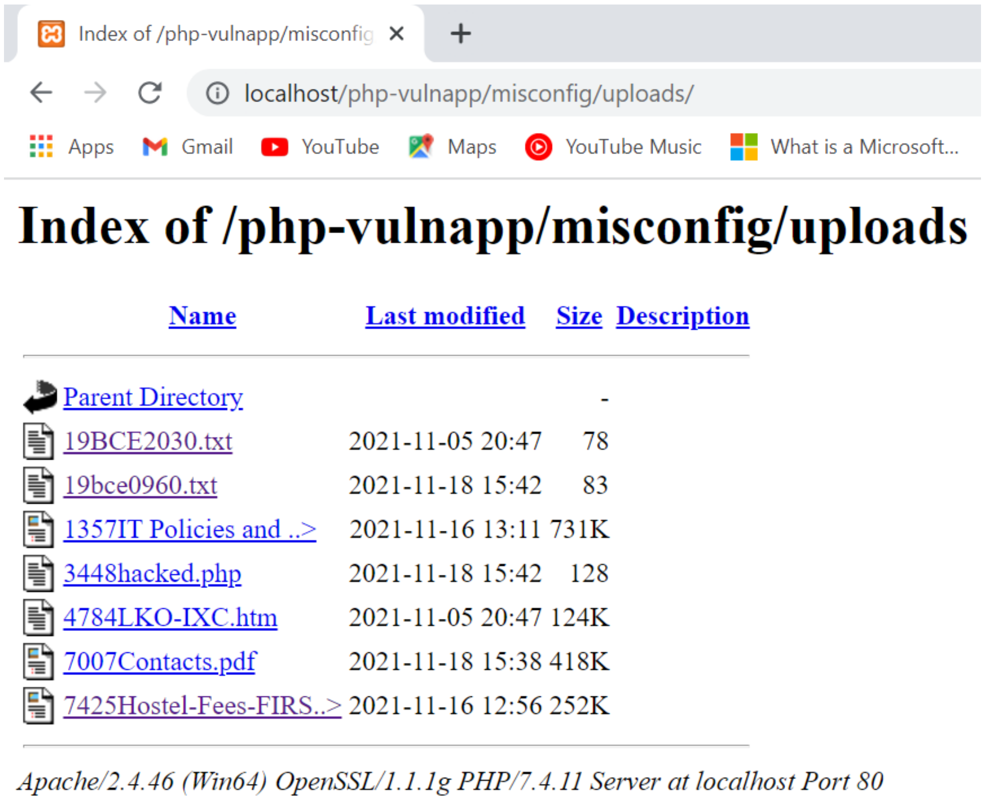
I9BCE0960

ADDITIONAL NOTES WITH HOMEWORK

ISAA REVIEW

ATTACH PDF OF SCANNED PAGES

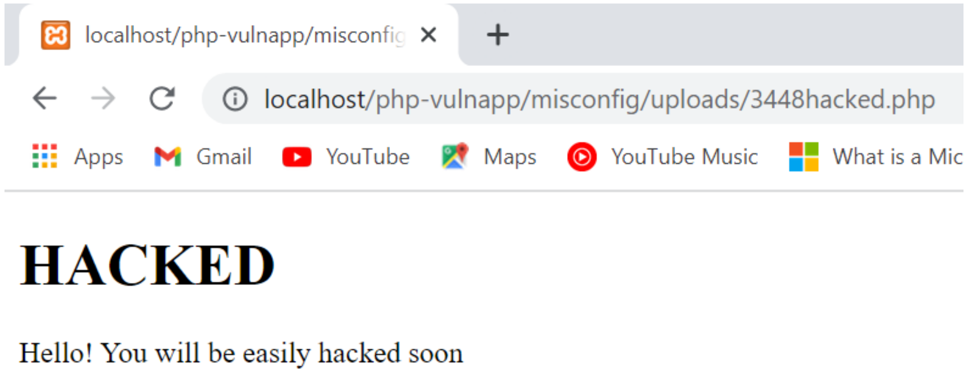
HACKED.PHP


Index of /php-vulnapp/misconfig
localhost/php-vulnapp/misconfig/uploads/
Apps
Gmail
YouTube
Maps
YouTube Music
What is a Microsoft...

Index of /php-vulnapp/misconfig/uploads

```
Name                          Last modified     Size  Description
Parent Directory                                   -
19BCE2030.txt                 2021-11-05 20:47    78
19bce0960.txt                 2021-11-18 15:42    83
1357IT Policies and ..>       2021-11-16 13:11  731K
3448hacked.php                2021-11-18 15:42   128
4784LKO-IXC.htm               2021-11-05 20:47  124K
7007Contacts.pdf              2021-11-18 15:38  418K
7425Hostel-Fees-FIRS..>       2021-11-16 12:56  252K
```

Apache/2.4.46 (Win64) OpenSSL/1.1.1g PHP/7.4.11 Server at localhost Port 80


localhost/php-vulnapp/misconfig
localhost/php-vulnapp/misconfig/uploads/3448hacked.php
Apps
Gmail
YouTube
Maps
YouTube Music
What is a Mic

HACKED

Hello! You will be easily hacked soon

CODE

BROKEN AUTHENTICATION:

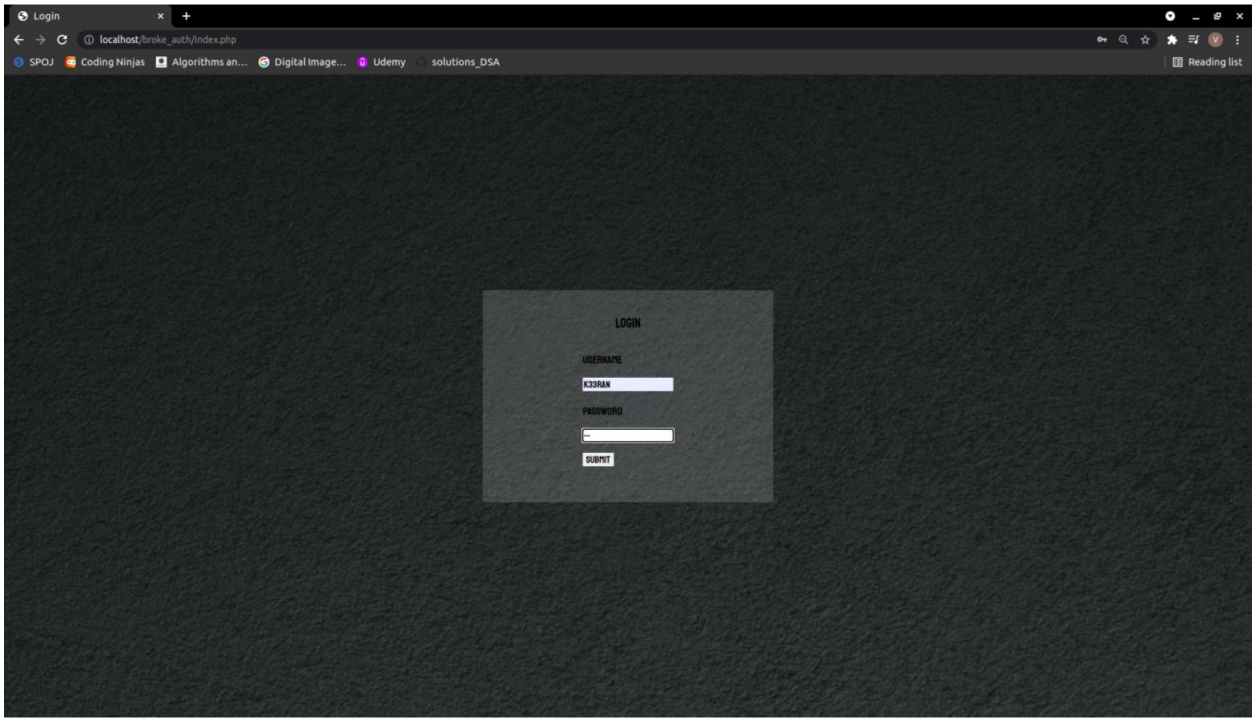
index.php

```
<?php
    if(isset($_COOKIE['user'])){
        header("Location: user.php");
        exit;
    }
?>
<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>Login</title>
        <style type="text/css">
            * { font-family: 'Staatliches', sans-serif; }
            body {
                margin: 0;
                display: flex;
                justify-content: center;
                align-items: center;
                background-image: url("bg1.jpg");
                background-position: cover;
                background-repeat: no-repeat;
                width: 100vw;
                height: 100vh;
            }
            #login {
                font-size: 20px;
                display: flex;
                justify-content: center;
                align-items: center;
                flex-flow: column nowrap;
                padding: 2vw 8vw 2vw 8vw;
                background-color: rgba(255, 255, 255, 0.15);
            }
            input {
                border: none;
                font-size: 18px;
            }
        </style>
    </head>
    <body>
        <div id="login">
            <div style="font-size: 24px;">Login</div><p></p>
            <form action="user.php" method="post">
                <label for="username">
                    Username<p></p>
                    <input name="username" type="text">
                </label><br>
                <br>
                <label for="password">
                    Password<p></p>
                    <input name="password" type="password">
                    <br>
                </label>
                <p></p>
                <input type="submit" value="Submit">
            </form>
        </div>
    </body>
</html>
```

user.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), "/");
        }
    }
?>
<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>
```

OUTPUT SCREENSHOT



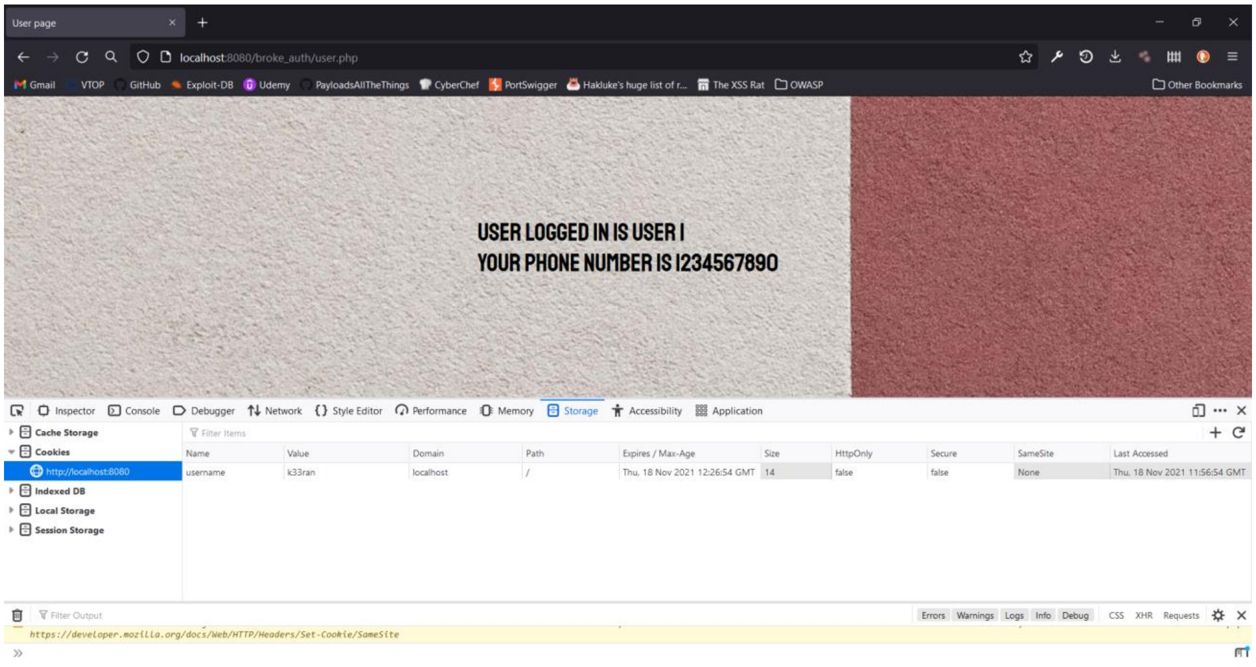
LOGIN

USERNAME

K33RAN

PASSWORD

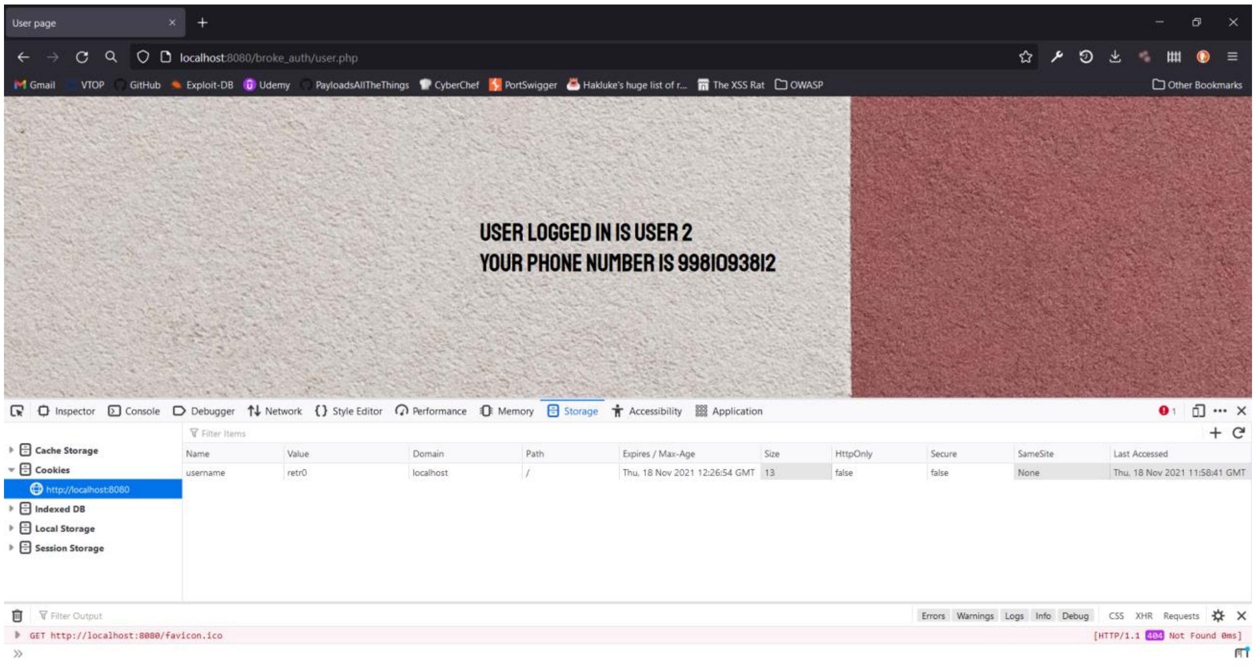
···



USER LOGGED IN IS USER 1
YOUR PHONE NUMBER IS 1234567890

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
username	k33ran	localhost	/	Thu, 18 Nov 2021 12:26:54 GMT	14	false	false	None	Thu, 18 Nov 2021 11:56:54 GMT

```
https://developer.mozilla.org/docs/Web/HTTP/Headers/Set-Cookie/SameSite
```



USER LOGGED IN IS USER 2
YOUR PHONE NUMBER IS 9981093812

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
username	retr0	localhost	/	Thu, 18 Nov 2021 12:26:54 GMT	13	false	false	None	Thu, 18 Nov 2021 11:58:41 GMT

```
GET http://localhost:8080/favicon.ico                [HTTP/1.1 404 Not Found 0ms]
```

SECURITY ANALYSIS

The OWASP Top Ten represents a broad consensus about what are the most critical application security flaws. The following table summarizes the OWASP Top Ten Most Critical Application Security Vulnerabilities:

S.No.	VULNERABILITY & DESCRIPTION	IMPACT
1	**Injection Flaws** Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data.	Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.
2	**Broken Authentication and Session**	Such flaws may allow some or even

	Management Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys or authentication tokens to assume other users' identities.	all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted.
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	**XML External Entities (XXE)** Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.	These flaws can be used to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack, as well as execute other attacks. The business impact depends on the protection needs of all affected applications and data.
5	**Broken Access Control** Frequently, an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized The technical impact is attackers acting as users or administrators, or users using privileged functions, or Final View Level: Confidential Page 8 of 15 users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly. Also, when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization.	The technical impact is attackers acting as users or administrators, or users using privileged functions, orcreating, accessing, updating or deleting every record. The business impact depends on the protection needs of the application and data.

6	**Security Mis-Configuration** Security mis-configuration 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.	Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.
7	**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.
8	**Insecure Deserialization** Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.	The impact of deserialization flaws cannot be overstated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible. The business impact depends on the protection needs of the application and data.
9	**Using Known Vulnerable Components** Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.	The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could be minimal, up to complete host takeover and data compromise.
10	**Insufficient Logging and Monitoring** It is coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.	Most successful attacks start with vulnerability probing. Allowing such probes to continue can raise the likelihood of a successful exploit to nearly 100%.

IMPACT:

The actual impact of an XSS attack generally depends on the nature of the application, its functionality and data, and the status of the compromised user. For example:

- In a brochureware application, where all users are anonymous and all information is public, the impact will often be minimal.
- In an application holding sensitive data, such as banking transactions, emails, or healthcare records, the impact will usually be serious.
- If the compromised user has elevated privileges within the application, then the impact will generally be critical, allowing the attacker to take full control of the vulnerable application and compromise all users and their data.

TESTING XSS:

The vast majority of XSS vulnerabilities can be found quickly and reliably using Burp Suite's web vulnerability scanner.

Manually testing for reflected and stored XSS normally involves submitting some simple unique input (such as a short alphanumeric string) into every entry point in the application, identifying every location where the submitted input is returned in HTTP responses, and testing each location individually to determine whether suitably crafted input can be used to execute arbitrary JavaScript. In this way, you can determine the context in which the XSS occurs and select a suitable payload to exploit it.

PREVENTION:

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. In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content.

Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.

- Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

GENERIC TIPS:

Preventing Cross-site Scripting (XSS) is not easy. Specific prevention techniques depend on the subtype of XSS vulnerability, on user input usage context, and on the programming framework. However, there are certain general strategic principles that you should follow to keep your web application safe.

Step 1: Train and maintain awareness
To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with XSS vulnerabilities. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

Step 2: Don’t trust any user input
Treat all user input as untrusted. Any user input that is used as part of HTML output introduces a risk of an XSS. Treat input from authenticated and/or internal users the same way that you treat public input.

Step 3: Use escaping/encoding
Use an appropriate escaping/encoding technique depending on where user input is to be used: HTML escape, JavaScript escape, CSS escape, URL escape, etc. Use existing libraries for escaping, don’t write your own unless absolutely necessary.

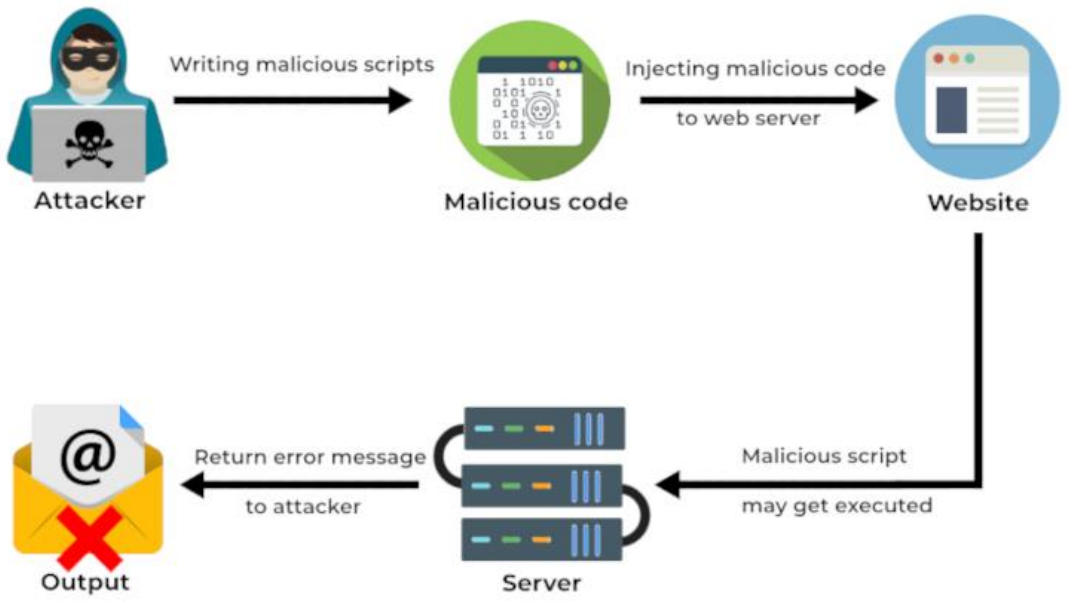
Step 4: Sanitize HTML
If the user input needs to contain HTML, you can’t escape/encode it because it would break valid tags. In such cases, use a trusted and verified library to parse and clean HTML. Choose the library depending on your development language, for example, HtmlSanitizer for .NET or SanitizeHelper for Ruby on Rails.

Step 5: Set the HttpOnly flag
To mitigate the consequences of a possible XSS vulnerability, set the HttpOnly flag for cookies. If you do, such cookies will not be accessible via client-side JavaScript.

Step 6: Use a Content Security Policy
To mitigate the consequences of a possible XSS vulnerability, also use a Content Security Policy (CSP). CSP is an HTTP response header that lets you declare the dynamic resources that are allowed to load depending on the request source.

Step 7: Scan regularly (with Acunetix)
XSS vulnerabilities may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically scan every build.

REMOTE CODE EXECUTION:


Attacker
Writing malicious scripts
Malicious code
Injecting malicious code to web server
Website
Malicious script may get executed
Server
Return error message to attacker
Output

After gaining access, the attacker might try to escalate privileges. This can completely compromise a vulnerable system.
Source code from third-parties applications, libraries, and plugins might be utilizing functions prone to RCE vulnerability. A recent case was with ImageMagick. ImageMagick is a well-known image processing library utilized by thousands of websites. Unfortunately, it had a RCE vulnerability, named ImageTragick.

IMPACT:

Remote code execution can leave the application and users at a high-risk, resulting in an impact on confidentiality, and integrity of data. An attacker who can execute commands with system or server privileges can:

- ❖ Add, read, modify, delete files
- ❖ Change access privileges
- ❖ Turn on and off configurations and services
- ❖ Communicate to other servers

PREVENTION:

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:

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

We don't know whether an attacker is targeting your systems because they're after cryptomining computing resources, or for a much more serious purpose. Either way, we must take the necessary preventative measures that ensure we're at minimal risk of a cyberattack – including an RCE attack. Here are a few key steps:

- Early alerts and monitoring. It may not be possible to always prevent an RCE attack, but early alert systems can tip you off to a security breach in situ – or to a situation where a successful attack has led to ongoing execution of illicit code. Similarly, a monitoring system can help you identify odd behavior that points to a compromised server.

- Firewalls and other security software. Deploy tools that can prevent common automated attacks: consider a website application firewall (WAF), for example. Likewise, deploy and run vulnerability scanning tools and penetration testing tools to help identify where a RCE attack may take place so that you can fix the vulnerability before it is too late.

- Build a response plan. It is challenging to avoid an RCE attack 100% of the time – even the tightest of security measures may be compromised. Plan for that event – construct a response plan that can help your organization to rapidly end an attack, and to quickly recover from the possible aftermath.

SENSITIVE DATA EXPOSURE

As from the implementation we have seen that information such as username,password and various other crucial data are getting exposed due to a very low end encryption technique used to validate the sensitive information. So to fix this vulnerability one solution can be to use some difficult decryption technique such as RSA,AES, or Triple DES.

With help of these techniques the attacker won't be able to decrypt our sensitive data so easily, and hence our data will be safe and secured.

For all sensitive data deserving encryption, all of the following needs to be done 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.

Here are 2 other ways, our data can be exposed:

1. **Intrusion**: Intruders can gain access to your data through a weakness in web applications. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. In September, Vodafone 2013 notified two million customers their personal and financial information had been breached. Generally, Organisations don’t publicly disclose breaches of internal information and trade secrets, as they are with regulated consumer data. Additionally, there’s no fraud algorithm to alert victims about illicit use of such data, leaving many cases of espionage undiscovered. Most of what we know publicly about this genre of the threat comes from incident responders, intelligence analysts, and malware researchers who compile and share their knowledge with the community.
2. **Phishing**: This is an attack on customers rather than on organizations. This is a clever method of extracting information from unsuspecting individuals. An e-mail, designed to look like it originated from a reputable company, usually a bank or online store, will tell you that there is a problem with your account. If links appear in this kind of e-mail message,

never click on them regardless of how "believable" they seem or who the source is. If you have an account with the organization, it's better to call your service representative and verify the authenticity of the e-mail. If it is legitimate, ask to complete the process by phone. This eliminates the need to send your sensitive data over unprotected networks.

How To Prevent '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.
6. Consider investing in DLP solutions or for Web Applications a WAF with custom rules (mask credit card numbers, SIN numbers) with targeted policies to prevent sensitive data exposure to clients

Threat Agents	**Attack Vectors**	**Security Weakness**	**Technical Impacts**	**Business Impacts**
Application Specific	Exploitability AVERAGE	Detectability AVERAGE	Impact SEVERE	Application / Business Specific
Consider who can gain access to your sensitive data and any backups of that	Attackers typically don't break crypto directly. They break something else,	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage are common,	Failure frequently compromises all data that should have been protected.	Consider the business value of the lost data and impact on your reputation.

data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats.	such as steal keys, do man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user's browser.	particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server-side flaws due to limited access and they are also usually hard to exploit.	Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc	What is your legal liability if this data is exposed? Also, consider the damage to your reputation.

SECURITY MISCONFIGURATION

Security misconfigurations arise when security settings are not defined, implemented, and default values are maintained. Usually, this means the configuration settings do not comply with the industry security standards (CIS benchmarks, OWASP Top 10 etc) which are critical to maintaining security and reduce business risk. Misconfiguration normally happens when a system or database administrator or developer does not properly configure the security framework of an application, website, desktop, or server leading to dangerous open pathways for hackers.
Misconfigurations are often seen as an easy target, as it can be easy to detect on misconfigured web servers, cloud and applications and then becomes exploitable, causing significant harm and leading to catastrophic data leakage issues.

Common Mistakes That Lead to Security Misconfiguration

1. **Unnecessary features are not removed or disabled**
   This can be things like:
   - Unnecessary ports being left open,
   - Unneeded services permitted to run,
   - Leftover pages still available to access, and
   - Unused accounts with certain privileges not being deleted.

   By not removing superfluous components, features, and code samples, you're leaving the application open to attacks. Say, for example, the application server includes demo programs that don't get deleted from the production server. If these demo programs have security vulnerabilities, then it's possible for hackers to use them to compromise the entire server.

2. **Default accounts and passwords are still being used**
   This is one of the simplest, yet most common items on the list of security configuration mistakes. All sorts of devices and programs come with a set of default credentials that allow the owner to initially access them out of the box. They're present in network devices, web applications, and pretty much anything that requires authentication.

   This isn't inherently a problem, except for the fact that they often don't get changed after the initial installation. By not having a policy requiring the changing of default credentials, you're leaving yourself exposed to an attack. It's easy for hackers to use lists of common default usernames and passwords to brute-force your system and gain unauthorized access.

3. **Error messages reveal too much information**
   Default server configurations can lead to overly informative error messages, containing information like detailed stack traces, being returned to users. This can result in sensitive information being disclosed and can give attackers helpful information like which component versions are being used so they can search for corresponding flaws to exploit.

4. **Old software versions or missed updates**
   Out-of-date software leaves your system exposed to known vulnerabilities that may already have patches (but of course, you can't stay fully protected unless you regularly perform updates).

5. **Upgraded systems aren't properly configured**
   Upgraded software can bring with it new security features, but said features aren't going to be terribly useful if they aren't enabled or configured correctly. You'll want to review every update to see what exactly is being changed/introduced/removed and adjust your configuration accordingly.

6. **Cloud systems aren't configured correctly**
   Cloud solution providers take a huge chunk of responsibility and workload off the laps of their clients. You can have your own cloud-based data center up and running in minutes, something that would've taken orders of magnitude longer previously. However, you're still responsible for the configuration of your cloud resources.
   AWS uses a "shared responsibility model," for example. AWS takes responsibility for protecting all the infrastructure that actually runs the AWS cloud services. The customer, on the other hand, must properly manage their cloud OS, associated software, and the AWS firewall.

7. **Debugging is left enabled**
   Most companies are set up in a way where they have two distinct environments, one for development and one for production. Debugging is enabled in the development environment because the greater amount of information that's returned is helpful in refining the software and cleaning up errors. It shouldn't be left on in the production

environment though because hackers can theoretically trigger lengthy error messages that expose sensitive code-related information that can ultimately be used against you.

8. **Unprotected files & directories are out in the open**
   Attackers can gain unauthorized access to sensitive files if developers neglect setting permissions on certain directories, dashboards, or admin consoles. Forced browsing attacks can be used to try and locate susceptible locations that can be accessed in the hope of finding restricted files.

   It's even easier for attackers if directory listing is enabled on the server. Then they can simply ask for a list of all the files and directories that are present. For example, a hacker could search for compiled Java classes and grab them off the server. Then they can decompile the code in the hopes of finding an exploitable flaw in the application.

Prevention of 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.

4. **Disable Directory Listing**
   Make sure that this feature is not enabled on any of your deployed applications and check that proper permissions have been set for files and folders. You'll also want to deny requests for particular file types. Restrict access to files that users shouldn't need to access, like .bak files for instance.

5. **Regularly patch and update software**
   This will help protect your applications and systems from malware and new vulnerabilities you may not be aware of yet.

6. **Remove unused features**

All they end up doing is increasing your application’s susceptibility to misconfiguration vulnerabilities. If no one should be using them, then there’s no point in keeping them around.

7. **Use automation to your advantage**
   Regularly run scans and perform audits to find things like missing patches, misconfigurations, the use of default accounts, unnecessary services, etc.

8. **Knowledge and Processes**
   Security Misconfiguration issues can result from both human error and a general lack of knowledge. By being aware of the most common mistakes and the easiest prevention measures, you’ll have a great foundation for keeping your systems safe from the vast majority of misconfiguration-focused attacks. It’s important to remember that these best practices must be a part of an organizational focus on security, with proper processes in place that keep your staff trained and your systems up to date.

BROKEN AUTHENTICATION

THREAT AGENTS	ATTACK VECTORS	SECURITY WEAKNESS		TECHNICAL IMPACTS	BUSINESS IMPACTS
Application Specific	Exploitability AVERAGE	Prevalence WIDESPREAD	Detectability AVERAGE	Impact SEVERE	Application / Business Specific
Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also, consider insiders wanting to disguise their actions.	An attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users.	Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique.		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.	Consider the business value of the affected data or application functions. Also, consider the business impact of public exposure to the vulnerabilities.

IMPACT:

Once your account is hijacked by exploiting a broken authentication vulnerability, the hacker can do anything that you have permission to do that can lead to serious consequences influencing your company's sustainability.
Even access to a single admin account is enough for the cybercriminals to comprise an entire web application. Based on the objective of the hacked application, the impact of session hijacking can range from data breaches, leakage of sensitive information, identity theft to administrative access.
The following points list the scenarios that can cause broken authentication.

- Weak usernames and passwords.
- Session fixation attacks.
- URL rewriting.
- Consumer identity details aren't protected when stored.

- Consumer identity details are transferred over unencrypted connections.

PREVENTION:

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.
4. **Disallow weak passwords:** Users must be required to set passwords of a specific length containing special characters, letters as well as numbers to prevent credential theft. Therefore, those passwords that do not meet the required complexity and length must be automatically rejected.
5. **Breached password protection:** Employ a breached password protection mechanism that locks the accounts of users whose passwords have been compromised until they verify and change the password to a new one. This will ensure that if passwords are stolen, the organization is notified.
6. **Strict credential recovery process:** The process to recover credentials must be strict, involving multiple verification checks to ensure that such recovery options are not misused by attackers.
7. **Secure password storage**: Passwords must be encrypted, hashed, and salted as it helps slow down brute-force attacks or other attempts to infiltrate password databases.
8. **Employ brute-force protection:** Applications should set a maximum limit for user-login attempts from a specific IP address, to prevent brute-force and credential stuffing attacks. Any user exceeding this limit must be disallowed from making any further attempts.

In addition to the above steps, it also becomes necessary to ensure that users are adequately trained and educated on the potential risks of broken authentication through phishing attacks or weak passwords. Organizations must employ strong Cybersecurity measures in line with the constantly evolving global standards and must ensure that they avoid broken authentication by all means possible.