Fixed Text

1. SQL Injection - fixed

The site was vulnerable to sql injection as seen on report 1.

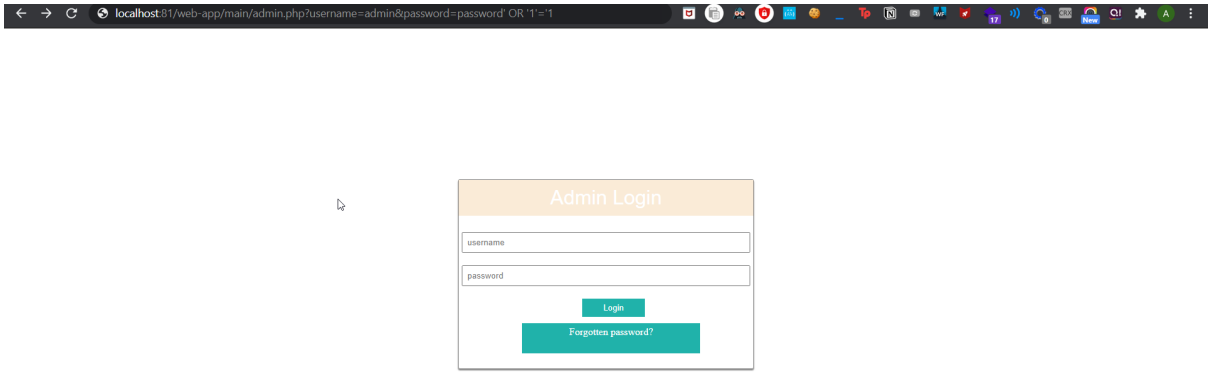
The login of the eCommerce site code was developed with SQL injection vulnerablity

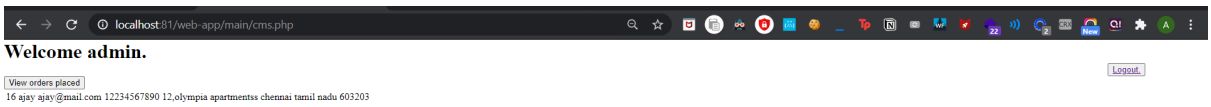
In login.php → the code vulnerable is

```
//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





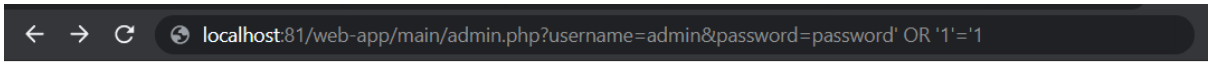
---

so we are santitzing the input and making sure that the ' ' are removed from the GET parameter preventing the sql injection from happening

We can also use PDOs which I'll implement in future versions which is more secure

```
$user = $_GET["username"];
$user = preg_replace('/[^a-z-Z0-9]/i', '', $user);
$password = $_GET["password"];
$password = preg_replace('/[^a-z-Z0-9]/i', '', $password);
```

adding this to the code makes sure only numbers and alphabets are taken as input



we get failed messge therfore the vulnerablity has been corrected.

---

2. injection

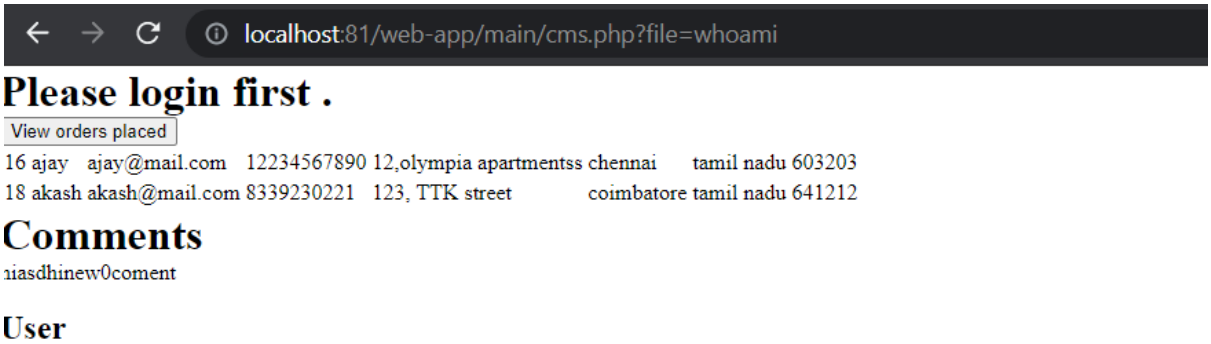
In the following example, the script passes an unvalidated/unsanitized HTTP request value directly to the include() PHP function. This means that the script will try to include whatever path/filename is passed as a parameter:

This vulnerability may be mitigated in different ways, depending on the specific case. However, the most common and generic way to do it is by using the `basename()` and `realpath()` functions.

The `basename()` function returns only the filename part of a given path/filename: `basename("../../../etc/passwd")` = *passwd*. The `realpath()` function returns the canonicalized absolute pathname but only if the file exists and if the running script has executable permissions on all directories in the hierarchy: `realpath("../../../etc/passwd")` = `/etc/passwd`

The new code

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



whoami command is not executed. if you comapre it with first reports output we can notice the difference

---

3. XSS

in xss-form.php

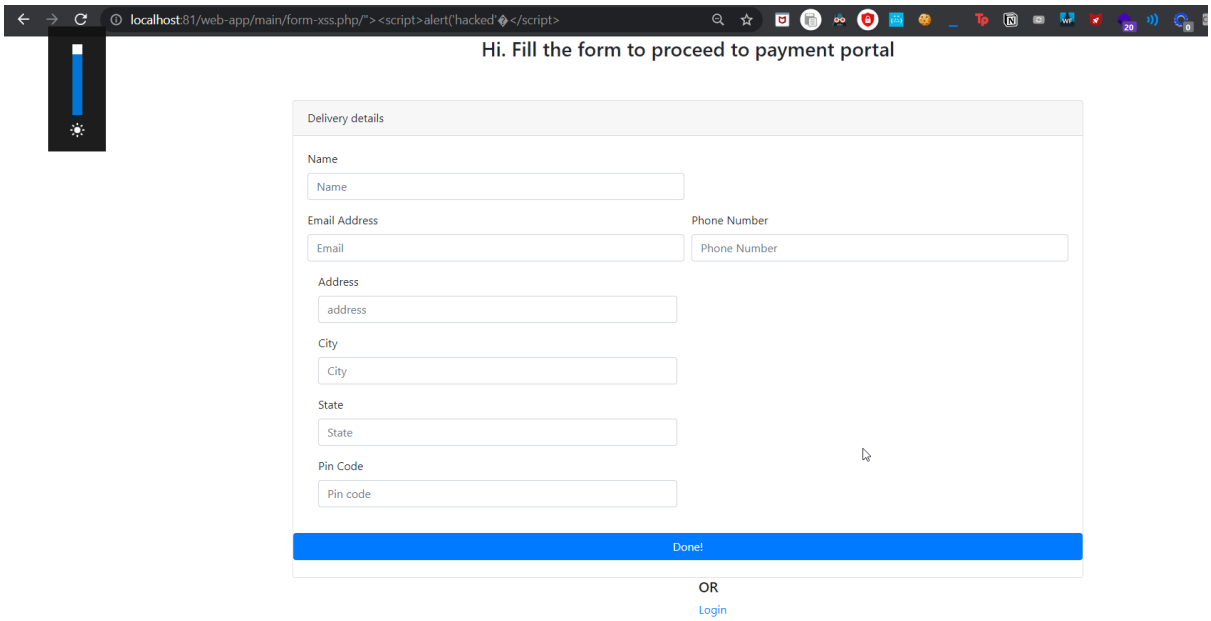
```
    </div>
    <div class="card-body">
        <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">

            <div class="form-row">
                <div class="form-group col-md-6">
                    <label for="inputName">Name</label>
                    <input type="text" class="form-control" name="name"
                    placeholder="Name" id="inputName">
                </div>
```

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

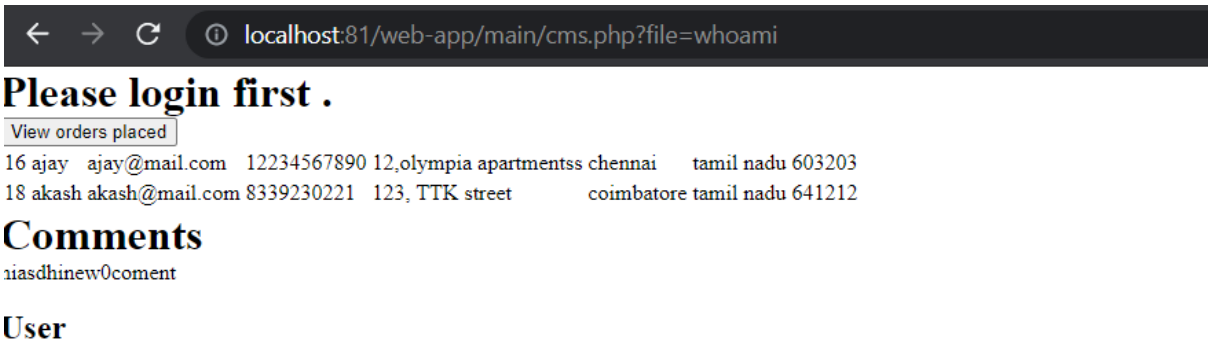
If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The htmlspecialchars() function converts special characters to HTML entities.



hence xss cannot be performed



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

---

5. Open url redirection

How To Prevent Open Redirects

The safest way to prevent open redirection vulnerabilities is not to use any redirections in your web applications. If this is not possible, you can attempt the following approaches:

- Use a list of fixed destination pages. Store their full URLs in a database table and call them using identifiers as request parameters, not the URLs themselves. For example, store *http://example2.com* in the database table with the identifier 42 and then use the following call to redirect to example2.com: *https://example.com/redirect.php?redir_id=42*.
- If you cannot use a fixed list of redirection targets, filter untrusted input (if you can, using a whitelist, not a blacklist). Make sure to check for partial strings, for example, *http://example.com.evil.com* is a valid URL. Additionally, disallow all protocols except HTTP and HTTPS. Also note, that despite your best efforts it is possible that attackers may find a way around your filters.

https://stackoverflow.com/questions/129677/how-can-i-sanitize-user-input-with-php

---

6.Reflected XSS

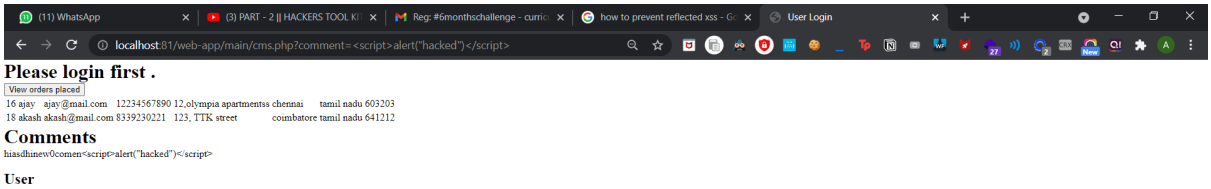
here we use same htmlspecialchar() function that was used in above xss

we can see it is modfied in the code and the comment that gets saved gets all special chars removed in below screenshots

cms.php | comments.txt

```
1 hiasdhinew0comen&lt;script&gt;alert(&quot;hacked&quot;)&lt;/script&gt;
```

```
64        echo "<h1>Comments</h1>";
65
66        file_put_contents("comments.txt",htmlspecialchars($_GET["comment"]
              ), FILE_APPEND);
67        echo file_get_contents("comments.txt");
68
69
```



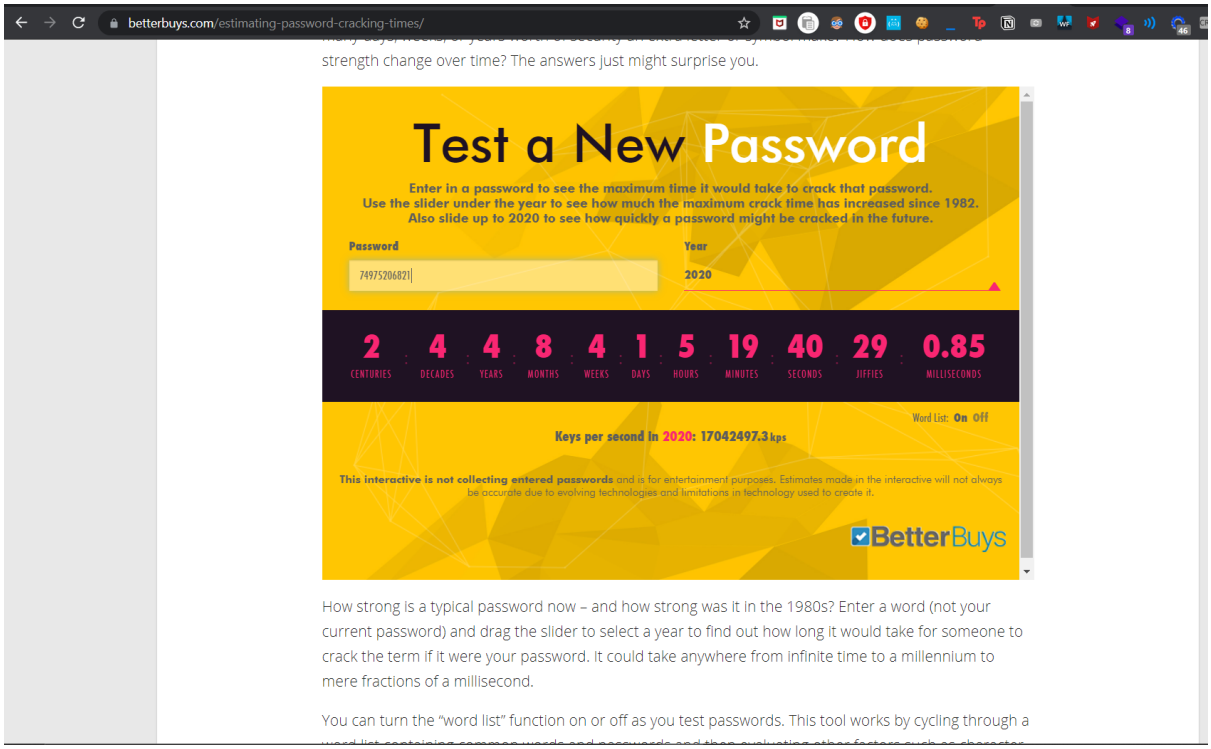
---

therefore a xss can not take place since we santitize the input that gets stored.

7.IDOR

IDOR occurs when the user is able to get the id or brute force it. so using a very random value which is highly impossible to bruteforce like using alpha numeric hashes to refer to variables stored in the database help preventing it.

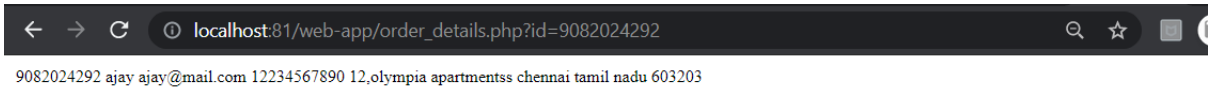
how long will it take to brute force a random 11 digit number?



hence we use a random 10 digit numbers for each ids and the ids of the users are known only to the admin and the developer team

we see we cant find it with a guessable number,

we can see that it can only be accessed with the random 11 digit number. no guessable numbers can allow us to see; Brute forcing will take a long time as seen from the link



https://www.betterbuys.com/estimating-password-cracking-times/

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

we disable directory listing from our mamp server

In Apache Virtual Host:

```
<Directory /var/www/public_html>
Options -Indexes
</Directory>
```

In .htaccess file:

```
Options -Indexes
```

While using the .htaccess, make sure that Apache server is enabled to use .htaccess files for that directory. In most cases, .htaccess is disabled by default.

Finally, reload the Apache service after doing changes in Virtual host to apply changes. The .htaccess changes will apply immediately without reloading service.