Web security

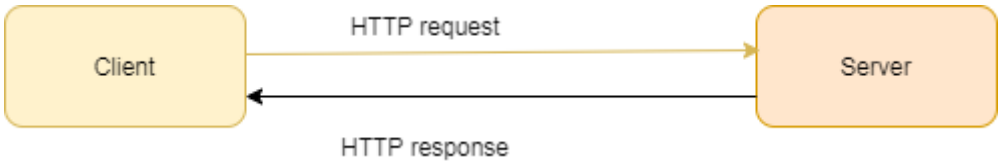
In this chapter we will focus on web security. Developers have to make efforts to ensure that the web application you use is secure. Lack of security in your code may result in private information being stolen.

Let's have a quick overview on how web applications work.

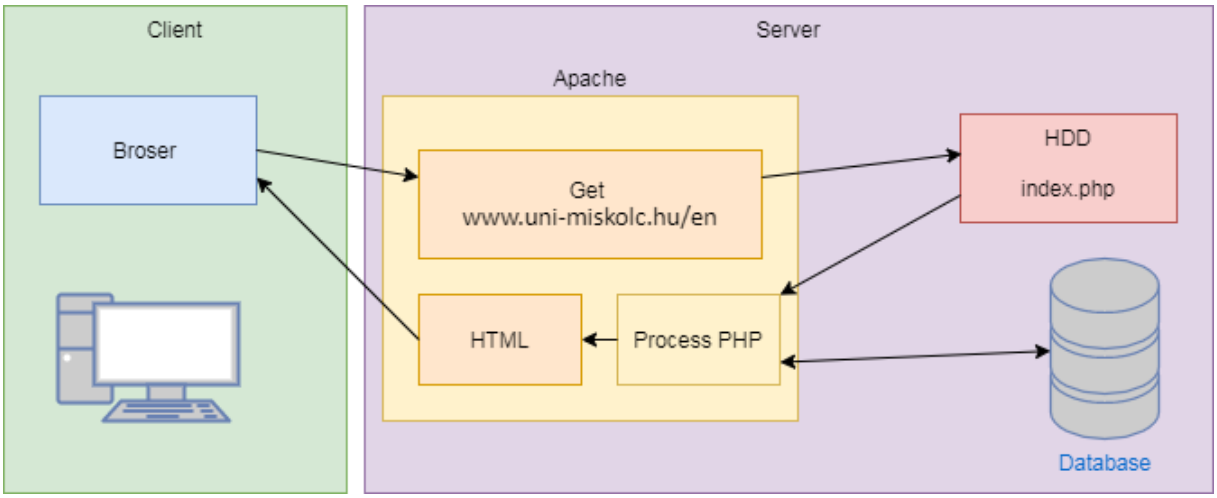
Communication

You use browsers for your communication

When you open your browser as a client, it sends a request to the server. The request uses Hypertext Transfer Protocol (HTTP). The server handles the request and sends back the response. The response consists of HTMM, CSS, JavaScript code that your browser understands and displays.



This type of communication is stateless, the server does not know what the browser is doing. A hacker may attack the client side or the server side.

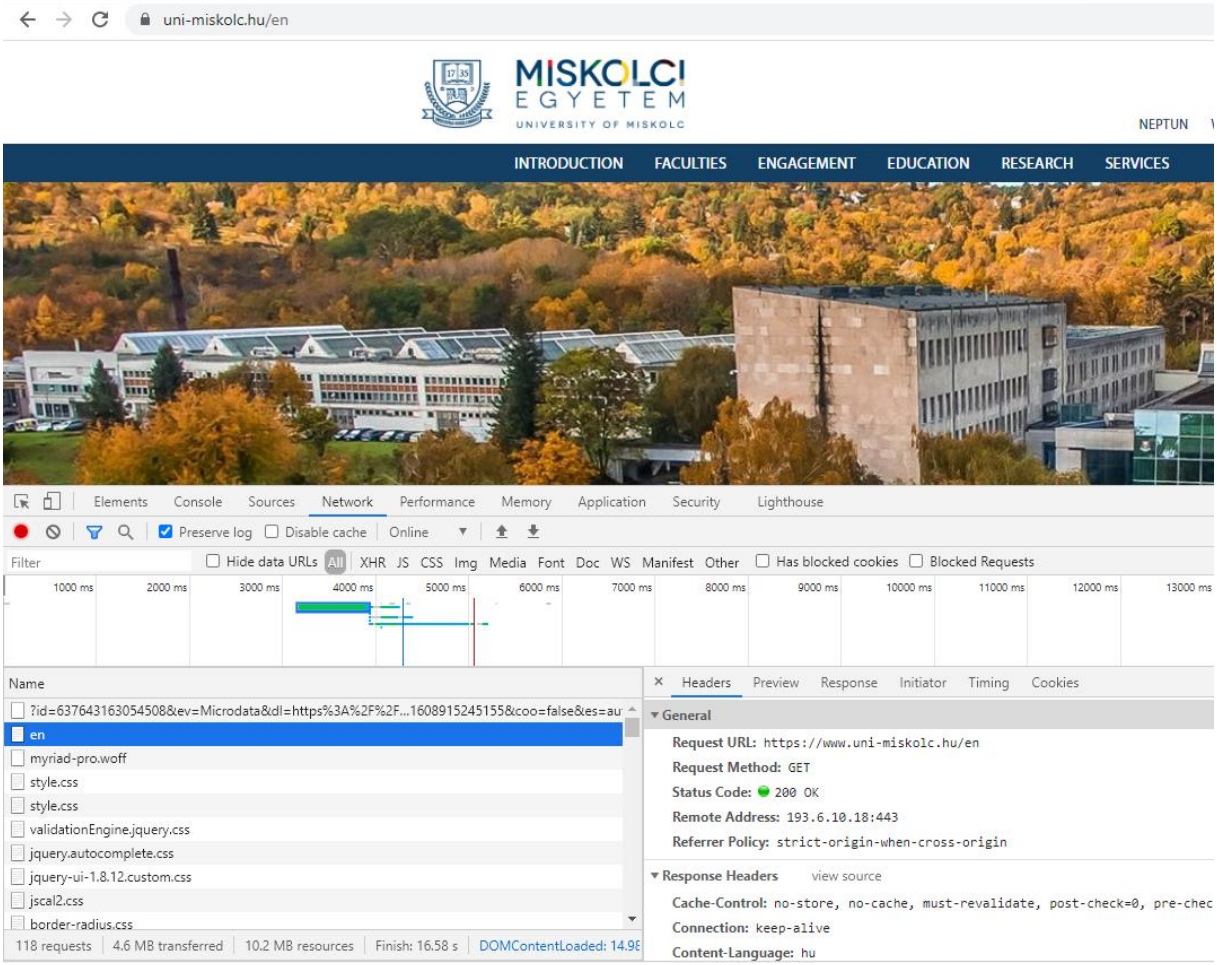


In the following examples the Apache web server will be used. The sample programs will be written in HTML, PHP language and a MySQL database will be also used. The workflow is as follows:

1. Your browser opens a php file
2. Apache looks for the file on the hard drive and processes the code
3. The PHP code opens a database to show data

4. An HTML file is rendered
5. which can display the data.

When you open a web page like www.uni-miskolc.hu/en then you will perform a GET request to the web server If you open the developer tools in your browser and go to the network ab you can see the actual header of the request



The status code 200 means that the processing went fine. The following status codes can be received:

1×× Informational

100 Continue

101 Switching Protocols

102 Processing

2×× Success

200 OK

201 Created

202 Accepted

203 Non-authoritative Information

204 No Content

205 Reset Content

206 Partial Content

207 Multi-Status

208 Already Reported

226 IM Used

3×× Redirection

300 Multiple Choices

301 Moved Permanently

302 Found

303 See Other

304 Not Modified

305 Use Proxy

307 Temporary Redirect

308 Permanent Redirect

4×× Client Error

400 Bad Request

401 Unauthorized

402 Payment Required

403 Forbidden

404 Not Found

405 Method Not Allowed

406 Not Acceptable

407 Proxy Authentication Required

408 Request Timeout

409 Conflict

410 Gone

411 Length Required

412 Precondition Failed

413 Payload Too Large

414 Request-URI Too Long

415 Unsupported Media Type

416 Requested Range Not Satisfiable

417 Expectation Failed

418 I'm a teapot

421 Misdirected Request

422 Unprocessable Entity

423 Locked

424 Failed Dependency

426 Upgrade Required

428 Precondition Required

429 Too Many Requests

431 Request Header Fields Too Large

444 Connection Closed Without Response

451 Unavailable For Legal Reasons

499 Client Closed Request

5×× Server Error

500 Internal Server Error

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

504 Gateway Timeout

505 HTTP Version Not Supported

506 Variant Also Negotiates

507 Insufficient Storage

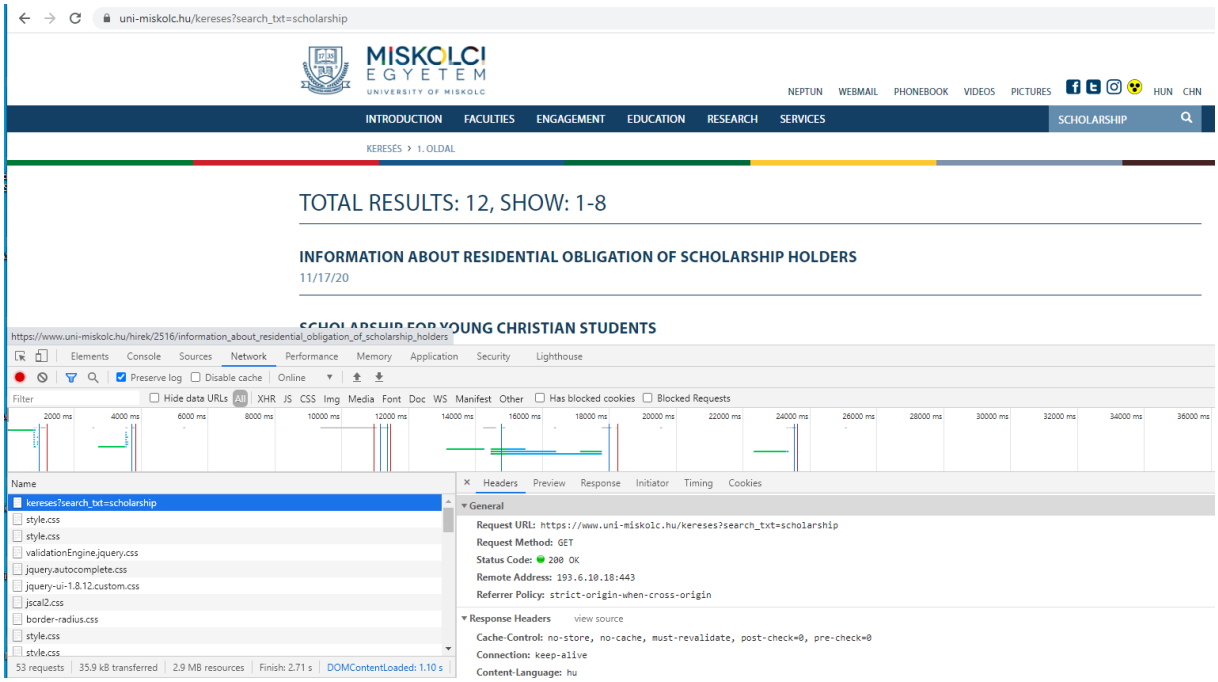
508 Loop Detected

510 Not Extended

511 Network Authentication Required

599 Network Connect Timeout Error

Enter ‘Scholarship’ search button to the search field and click search button.



As you will see the actual URL changes to

https://www.uni-miskolc.hu/kereses?search_txt=scholarship

As you could see a query string is sent in the URL of a GET request, the query string starts with a question mark, the pairs are separated by & and they and look like

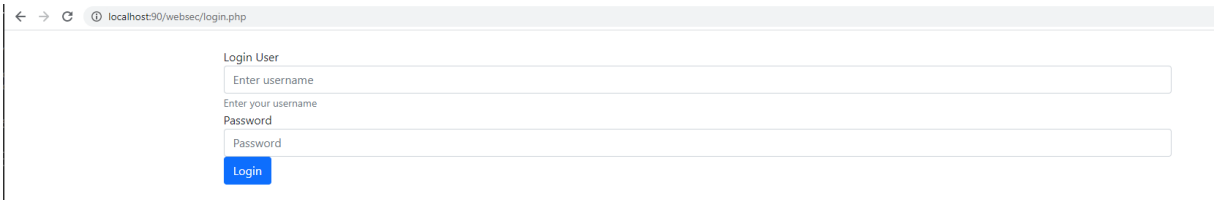
?name1=value1&name2=value2

Lets’s create a simple login form

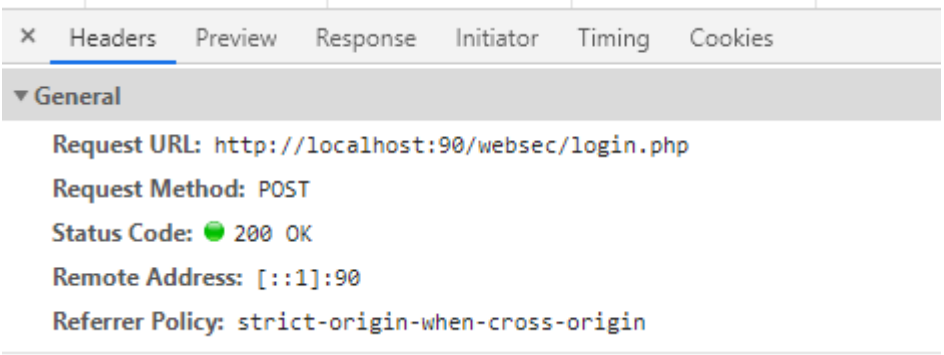
```
<!doctype html>
<html lang="en">
<head>
    <title>Login sample</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"
rel="stylesheet" crossorigin="anonymous">
    <!-- Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta1/dist/js/bootstrap.bundle.min.js" crossorigin="anonymous"></script>
</head>

<body>
</br>
<div class="container">
    <form id='login' action="login.php" method='post' accept-charset='UTF-8'>
    <div class="form-group">
        <label for="username">Login User</label>
```

```
        <input type="text" name='userName' class="form-control" id="username" aria-
describedby="userNameHelp" placeholder="Enter username" maxlength="50"  required />
        <small id="userNameHelp" class="form-text text-muted">Enter your username</small>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" name='password' class="form-control" id="password"
placeholder="Password" maxlength="50" required />
    </div>
    <input type="submit" class="btn btn-primary" name='Submit' value='Login' />
    </form>
</div>
</body>
</html>
```

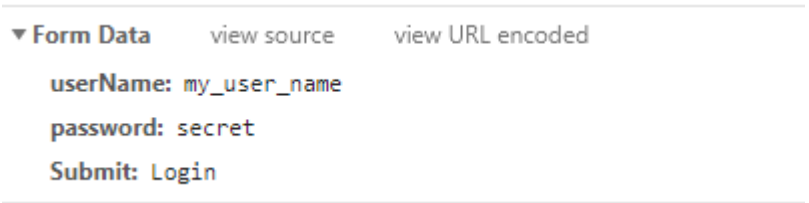
The page looks like this



When you enter your credentials and investigate the header you will see that this is a POST request,



The form data is sent to the browser



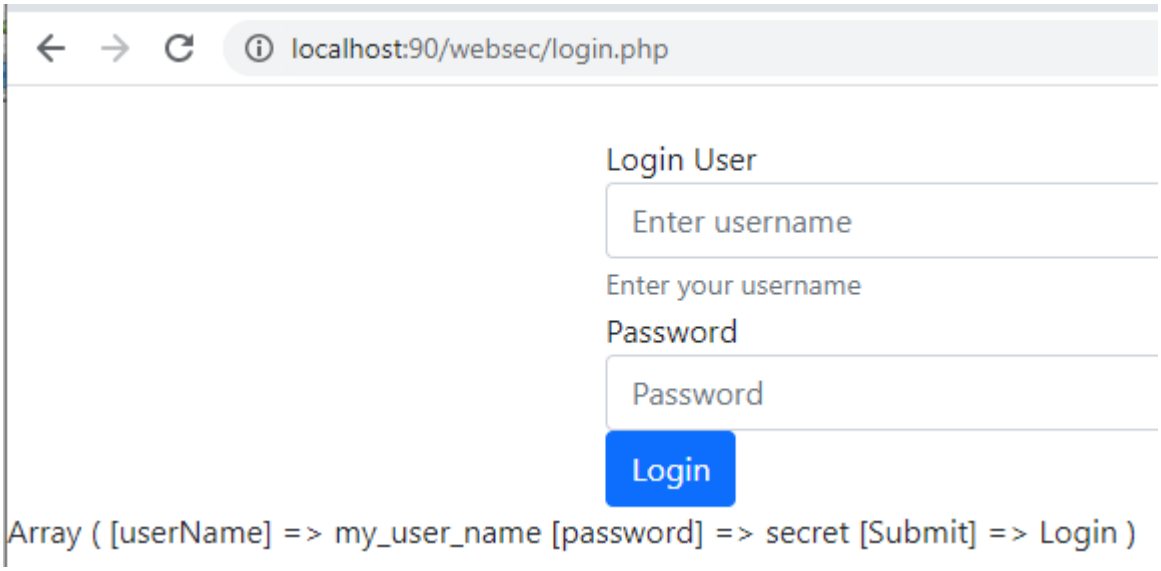
The properties of the two request types are compared in the following table:

GET requests	POST requests
can be cached	never cached
remain in the browser history	do not remain in the browser history
can be bookmarked	cannot be bookmarked

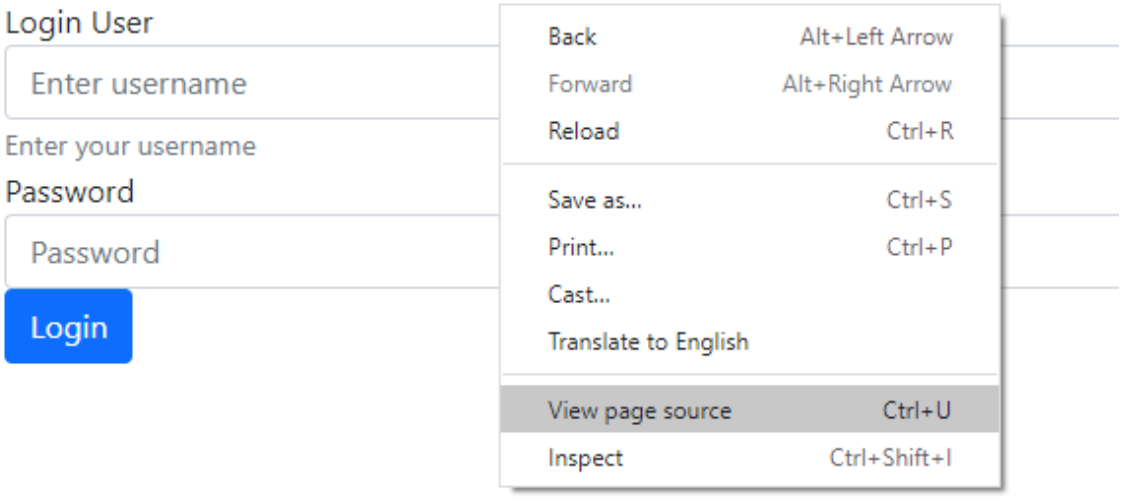
have length restrictions	have no restrictions on data length
should never be used when dealing with sensitive data	
only used to request data (not modify)	
only ASCII characters allowed	binary data is also allowed
data is visible to everyone in the URL	Data is not displayed in the URL

You can add php code to your HTML between <? php …?> tags. If you want to print the array of POST parameters add the following code before the </body> tag

```
<?php
print_r ($_POST);
?>
```



If you right click on the page you can view page source

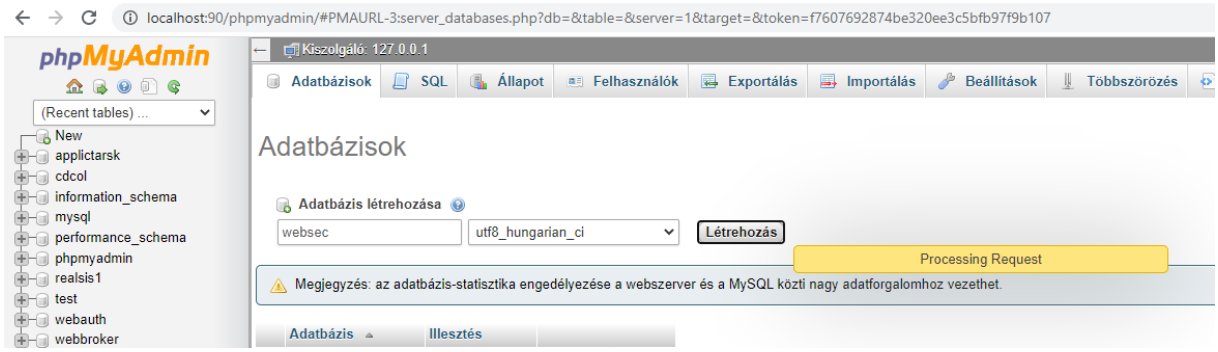


Developer may leave some test code left

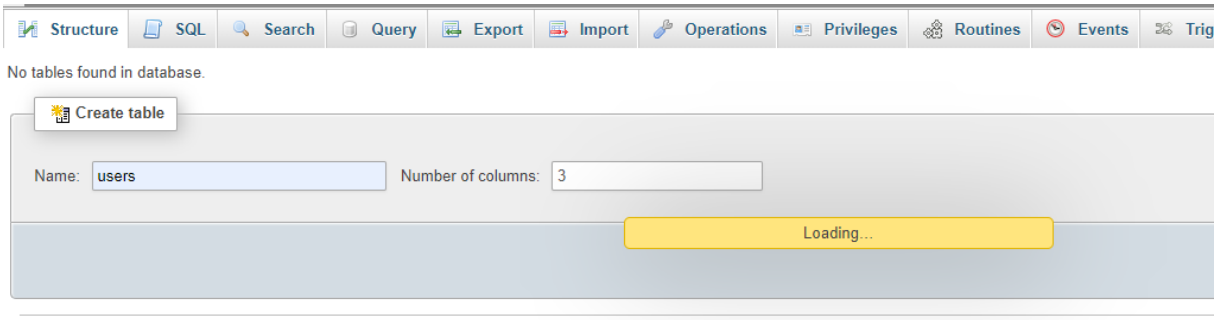
If you have to document your code then add it as a php comment, so that it won't be visible in browsers source code

```
<?php // Test data: user=admin, password=admin ?>
```

Create a new database called websec



Create a table with 3 columns

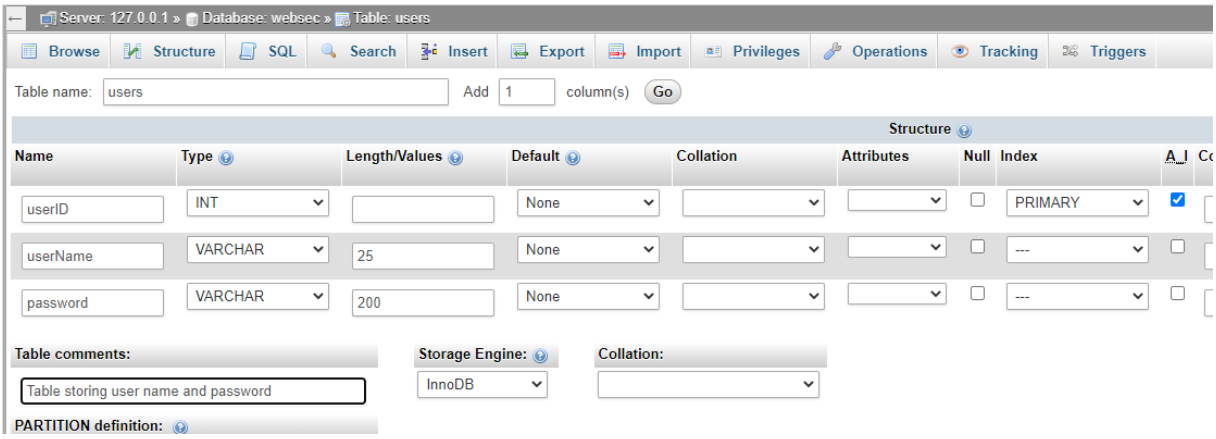


Add 3 users using the following SQL

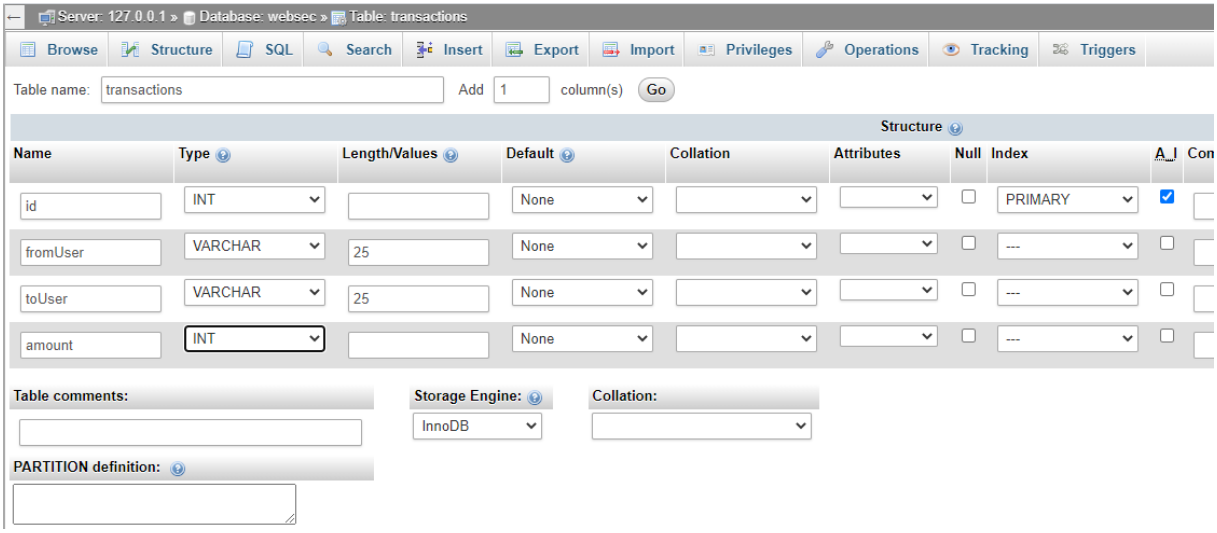
INSERT INTO `websec`.`users` (`userID`, `userName`, `password`) VALUES (NULL, 'admin', 'admin');

INSERT INTO `websec`.`users` (`userID`, `userName`, `password`) VALUES (NULL, 'alice', '1234');

INSERT INTO `websec`.`users` (`userID`, `userName`, `password`) VALUES (NULL, 'bob', '5678');



Then create a transactions table with 4 columns



INSERT INTO `websec`.`transactions` (`id`, `fromUser`, `toUser`, `amount`) VALUES (NULL, 'alice', 'bob', '111');

INSERT INTO `websec`.`transactions` (`id`, `fromUser`, `toUser`, `amount`) VALUES (NULL, 'bob', 'admin', '222');

Let's write a piece of code that

```
<body>
<div class="container">
    <h1> List of users</h1>
    <table class="table table-bordered table-striped">
        <thead class="thead-dark">
            <tr>
                <th> UserID </td>
                <th> UserName</td>
                <th> Password</td>
            </tr>
        </thead>
        <tbody>
            <?php

                //database authentication
                $hostDB="127.0.0.1"; $userDB ="root"; $passwordDB=""; $databaseDB
="websec";

                //connect to database
                $connect = mysqli_connect($hostDB,
$userDB,$passwordDB,$databaseDB);
                if (mysqli_connect_errno()){
                    die(" cannot connect to database ". mysqli_connect_error());
                }

                $query ="select * from users " ;

                $result= mysqli_query($connect,$query);
                if (!$result){
                    die(' error while running query');
                }

                $userInfo=array();
                $loginInUser=null;
                while ($row= mysqli_fetch_assoc($result)){
                    echo " <tr>";
                    echo " <td>". $row["userID"] ." </td>";
                    echo "  <td>". $row["userName"]."</td>";
                    echo " <td>". $row["password"]."</td>";
                    echo " </tr>";
                    }

                mysqli_free_result($result);
                mysqli_close($connect);
            ?>
        </tbody>
    </table>
</div>
</body>
```

Let's move the database authentication code to a separate file

Now add some code to login page so that on successful login it shows a Transactions button

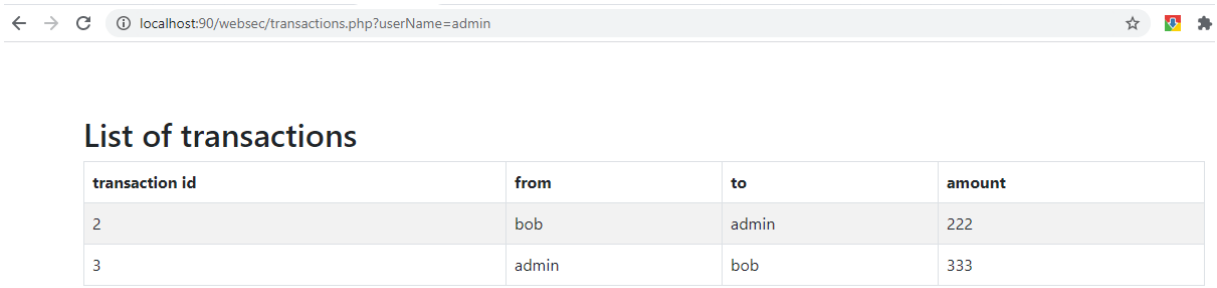
```php
<?php
require("dbAuth.inc");

$userName = isset($_POST['userName']) ? $_POST['userName'] : '';
$password = isset($_POST['password']) ? $_POST['password'] : '';

if(!empty($userName) and !empty($password)){
    //connect to database
    $connect = mysqli_connect($hostDB, $userDB,$passwordDB,$databaseDB);
    if(mysqli_connect_errno()){
        die(" cannot connect to database ". mysqli_connect_error());
    }

    $query ="select * from users  where userName='" .
        $userName ."' and password='" . $password ."'" ;

    $result= mysqli_query($connect,$query);
    if (!$result){
        die(' error while running query');
    }

    $loginInUser=null;
    while ($row= mysqli_fetch_assoc($result)){
        $loginInUser = $row["userName"];
        break;
    }

    mysqli_free_result($result);
    mysqli_close($connect);

    echo "<pre>";
    echo "Server data:</br>";
    echo "username: " . $userName. "</br>";
    echo "password: " . $password. "</br>";
    echo "query: " . $query . "</br>";
    if (! empty($loginInUser)){
        echo "</br><div class=\"alert alert-success\">Login successful for
(". $loginInUser .")";
        echo "</br></br><a class=\"btn btn-success\"
href='transactions.php?userName=" . $loginInUser ."'> Transactions</a>";
        echo "</div>";

    }else{
        echo "</br><div class=\"alert alert-danger\">Database login
failed</div>";
    }
    echo "</pre>";
}
?>
```

Create a new transactions.php file that lists all the transactions where from or to user is the logged in user:

```
        <br /><br /><br />
```

```
        <h2> List of transactions</h2>
        <table class="table table-bordered table-striped">
            <thead>
            <tr>
                <th> transaction id </td>
                <th> from </td>
                <th> to</td>
                <th> amount</td>
            </tr>
            </thead>
            <tbody>
<?php
//Database Authentication
require("dbAuth.inc");

$userName = $_GET['userName'];

//connect to database
$connect = mysqli_connect($hostDB, $userDB,$passwordDB,$databaseDB);
if(mysqli_connect_errno()){
    die(" cannot connect to database ". mysqli_connect_error());
}

// get user transactions
if( !empty($userName)) {
    $query ="select * from transactions  where fromUser='". $userName ."' or
toUser='". $userName ."'" ;
    $result= mysqli_query($connect,$query);
    if (!$result){
        die(' Error cannot run query');
    }

    $userInfo=array();
    $loginInUser=null;
    while ($row= mysqli_fetch_assoc($result)) {
        $rowColor ="class='success'";
        if($row["fromUser"]==$userName){
            $rowColor ="class='danger'";
        }
        echo " <tr ". $rowColor .">";
        echo " <td>". $row["id"] ." </td>";
        echo " <td>". $row["fromUser"] ." </td>";
        echo "  <td>". $row["toUser"]."</td>";
        echo " <td>". $row["amount"]."</td>";
        echo " </tr>";
    }
    mysqli_free_result($result);
}
mysqli_close($connect);
?>
            </tbody>
        </table>
```

The page looks like



Avoid validation rules

Let's create an addUser.php page which is very similar to the login form. It adds a new user to the database. The form has a javascript code the checks if the password you enter has at least 5 lowercase letters. The form is as follows

```
    <form id='adduser' action="addUser.php" method='post' accept-charset='UTF-8'>
    <div class="form-group">
        <label for="username">User</label>
        <input type="text" name='userName' class="form-control" id="username" aria-
describedby="userNameHelp" placeholder="Enter username" maxlength="50" />
        <small id="userNameHelp" class="form-text text-muted">Enter your
username</small>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" name='password' class="form-control" id="password"
aria-describedby="passwordHelp" placeholder="Password" maxlength="50"/>
        <small id="passwordHelp" class="form-text text-muted">Enter password to
enable Add button</small>
    </div>
    <input type="submit" class="btn btn-primary" id="submit" name='Submit'
value='Add' disabled/>
    </form>
```

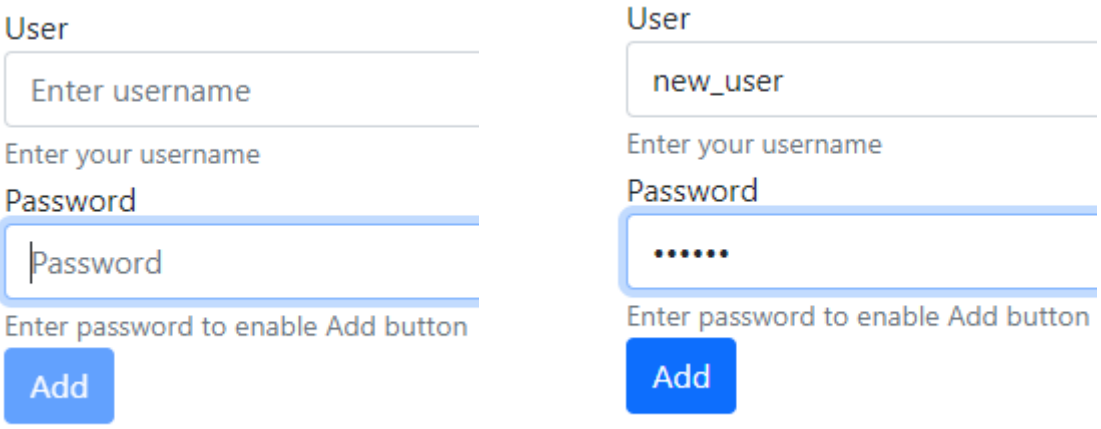
The insert SQL is constructed as:

```
    $query ="Insert into login(userName,password) VALUES ('" .
        $userName ."','" . $password ."')" ;
```

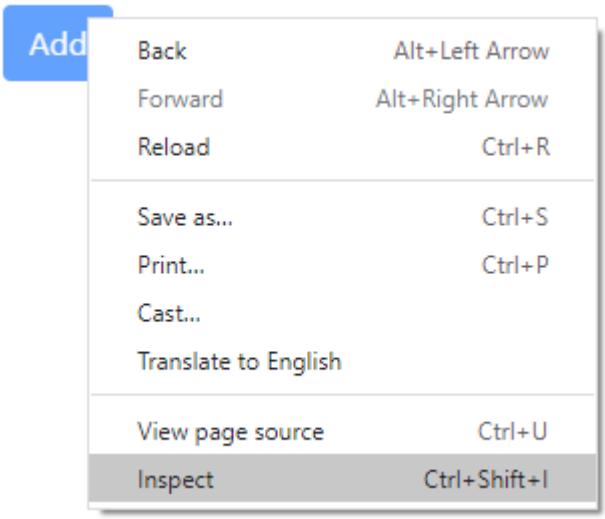
As you can see Add button is disabled by default. There is a JavaScript on the page that enables the button when more than 5 lowercase letters are entered to the password field:

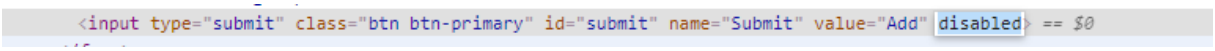
```
<script>
    var passwordField = document.getElementById("password");
    var submitBtn = document.getElementById("submit");
    submitBtn.disabled = true;
    // When the user types something into the password field
    passwordField.onkeyup = function() {
        submitBtn.disabled = true;
        // Validate lowercase letters
        var lowerCaseLetters = /[a-z]/g;
        if (!passwordField.value.match(lowerCaseLetters)) {
         return;
        }
        if(passwordField.value.length > 5) {
```

```
            submitBtn.disabled = false;
        }
    }
</script>
```

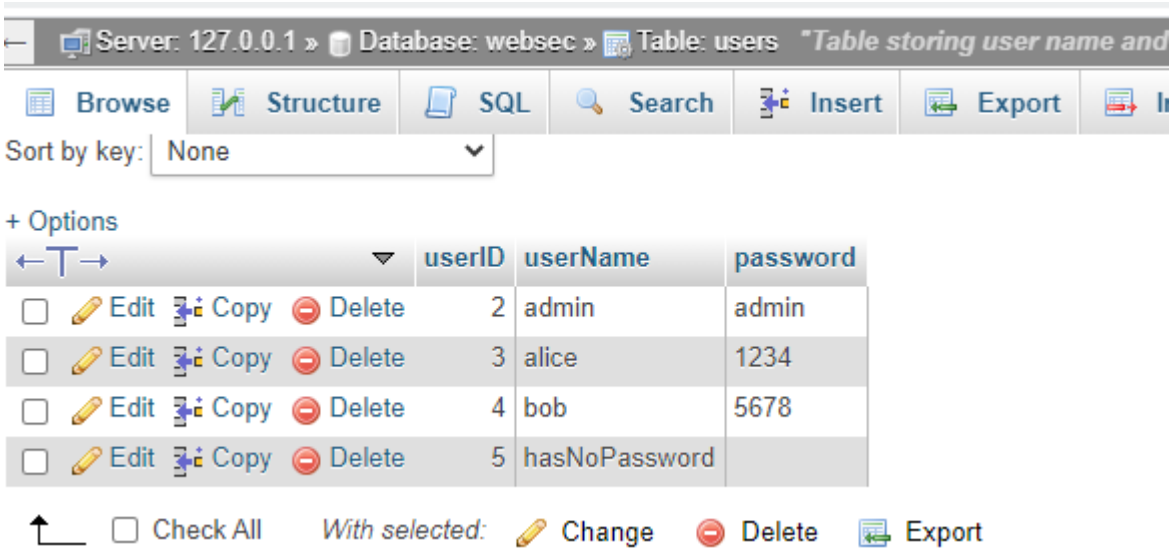


Now right click on the Add button and select Inspect menu:





Replace ‘disabled’ to ‘enabled’ Now you can skip validation and submit the form without the password validation. If you open your database table, you will see a user entry without password field.



So, the solution is to add the validation code to the server side. Note the if statement that checks username and password not to be empty

```
<?php
require("dbAuth.inc");

$userName = isset($_POST['userName']) ? $_POST['userName'] : '';
$password = isset($_POST['password']) ? $_POST['password'] : '';

if(!empty($userName) and !empty($password)) {
    //connect to database
    $connect = mysqli_connect($hostDB, $userDB,$passwordDB,$databaseDB);
    if(mysqli_connect_errno()){
        die(" cannot connect to database ". mysqli_connect_error());
        }

    $query ="Insert into users(userName,password) VALUES ('" .
        $userName ."','" . $password ."')";

    $result = mysqli_query($connect,$query);
    if (!$result) {
        die(' error while running query');
        }
    mysqli_close($connect);
    }
?>
```

Avoid plain text get parameters

Let's go back to our login page. Log in as admin and click to see the transactions of admin. When you see the URL, it passes the username http://localhost:90/websec/transactions.php?userName=admin

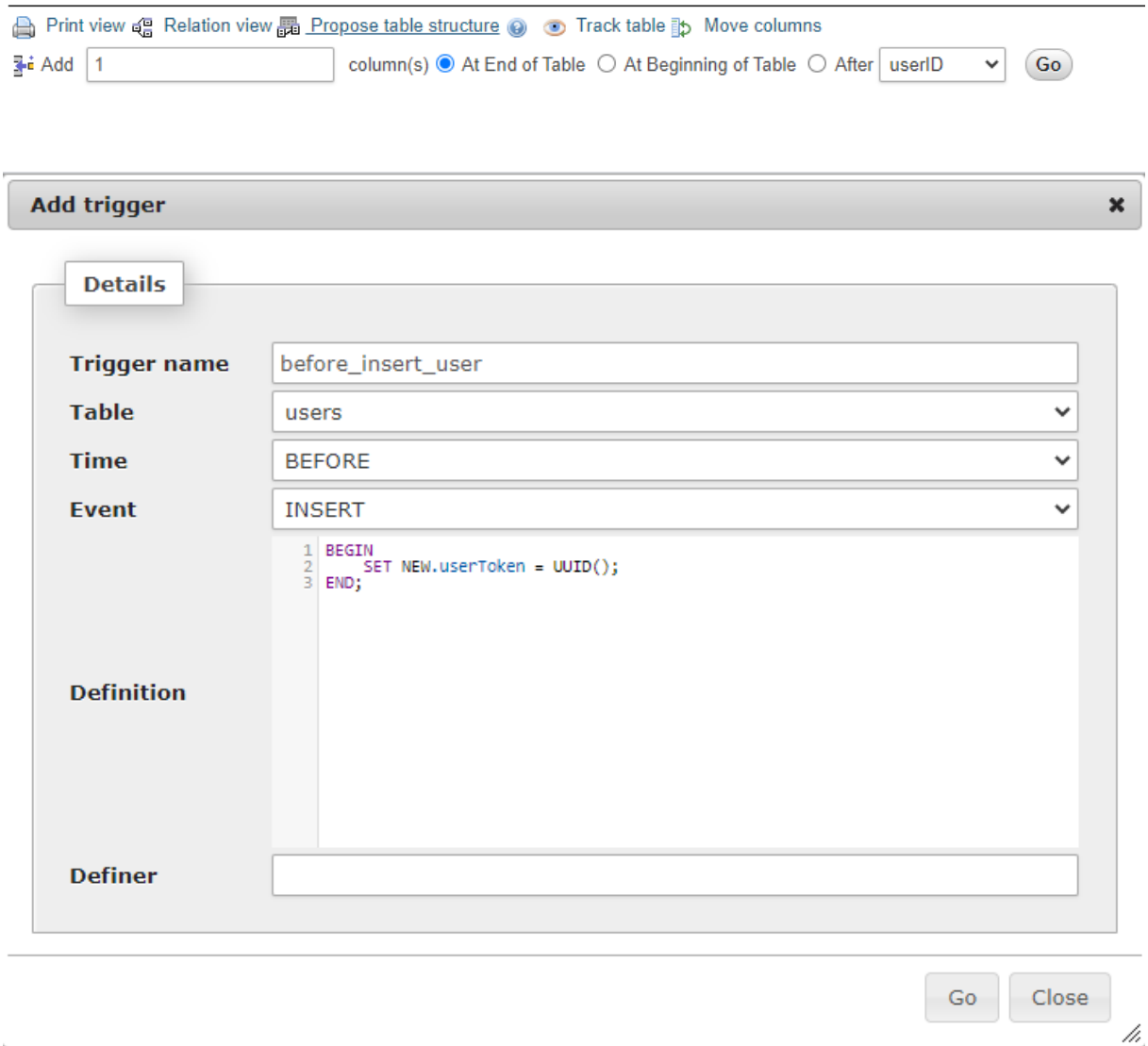
What happens if you change 'admin' to 'bob' ?

List of transactions

transaction id	from	to	amount
1	alice	bob	111
2	bob	admin	222
3	admin	bob	333

The page lists all transactions of Bob without Bob being logged in.

Do not use a get parameter that easy to guess



UPDATE `users` SET `userToken`= UUID();

And now in login.php page you can use the user token

```
$loginInUser = null;
$userToken = null;
while ($row = mysqli_fetch_assoc($result)) {
    $loginInUser = $row["userName"];
    $userToken = $row["userToken"];
```

```
        break;
    }
```

and the transactions.php parameter should be modified as

```
    if (! empty($loginInUser)) {
        echo "</br><div class=\"alert alert-success\">Login successful for
(". $loginInUser .")";
        echo "</br></br><a class=\"btn btn-success\"
href='transactions.php?userToken=" . $userToken ."'> Transactions</a>";
        echo "</div>";
    }
```

And in transactions.php you need to deduce user token from user name

```
//Database Authentication
require("dbAuth.inc");

//connect to database
$connect = mysqli_connect($hostDB, $userDB,$passwordDB,$databaseDB);
if(mysqli_connect_errno()) {
    die(" cannot connect to database ". mysqli_connect_error());
}

$userToken = isset($_GET['userToken']) ? $_GET['userToken'] : '';
// get user name from token
$query ="select * from users  where userToken='" . $userToken ."'";

$result= mysqli_query($connect,$query);
if (!$result) {
die(' error while running query');
}

$userName=null;
while ($row= mysqli_fetch_assoc($result)) {
    $userName= $row["userName"];
    break; // to be save
}
```

Hijacking cookies

A browser cookie is a text file. When you browse along sites some piece of information can be written to the cookie file to personalize your browsing and enable quick access to the website.

Cookies typically stores the number of visits of your computer, searches and purchases on the site.

Session Computer Browser Cookies

A session browser cookie can be stored in the web browser. It is deleted when the user exits the browser. Session cookies are used in shopping carts to record the content of your basket. These browser cookies let the user move from page to page without the need to repeatedly sign in.

Permanent Computer Browser Cookies

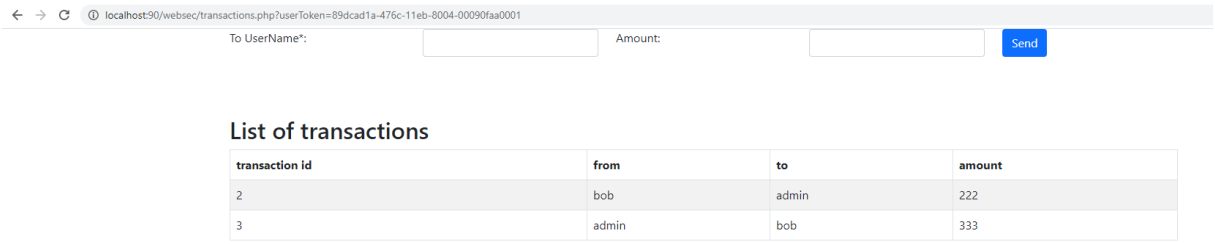
Permanent browser cookies are saved to the hard drive until they expire. they are not deleted when the browser is shut down. They are used to profile the browsing behavior of the user on the website

These browser cookies can only be read by the server that places the cookies on the computer.

Third Party Computer Browser Cookies

Banner ads and ads targeted to a user's location and interest are the result of third-party cookies. Web sites give permission to companies to put these browser cookies on computers and that third-party company can determine user interest by tracking the web surfer through browser activity. [4]

Let's extend the transaction page with a from, where the logged in user can send some amount of money.

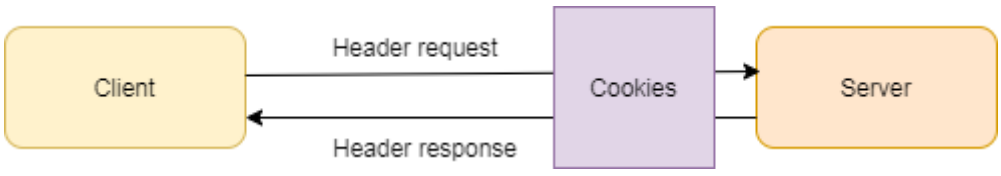


The SQL to run is quite straightforward:

```
//Add new transaction
if(!empty($_POST['fromUserName']) and !empty($_POST['ToUserName'])) {

    $query ="insert into transactions(fromUser, toUser, amount)
      values ('". $_POST['fromUserName'] ."','". $_POST['ToUserName']
."',". $_POST['Amount'] .")";

    $result= mysqli_query($connect,$query);
    if (!$result){
        die(' error while running query');
    }
}
```

The developer decides to use cookies to save the logged in user's username.

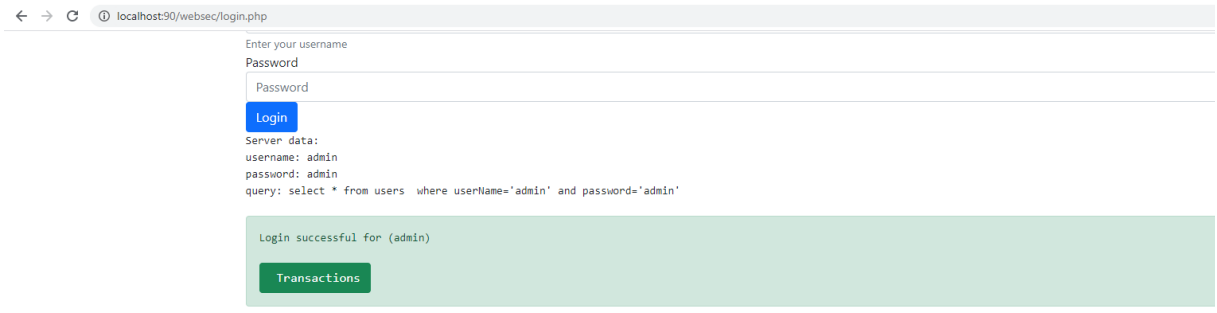
The browser can save such data for tracking your activity.

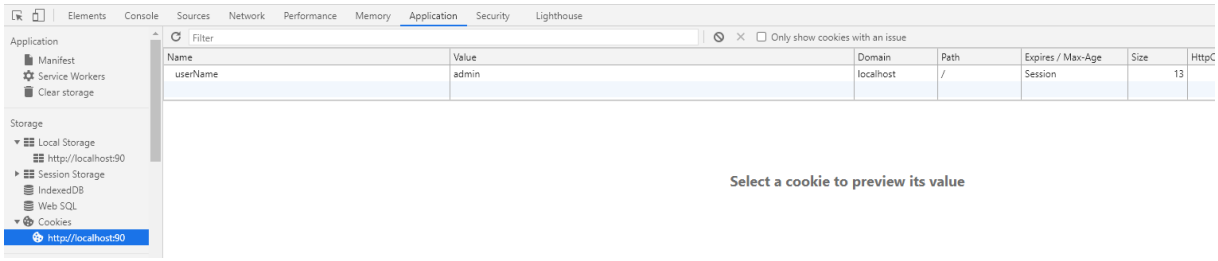


In login.php save the username into a cookie by

setcookie('userName', $loginInUser,false,"/",false);

If you go to the Applications tab you can see the cookies stored with the browser





In transactions.php add this form above the transactions list

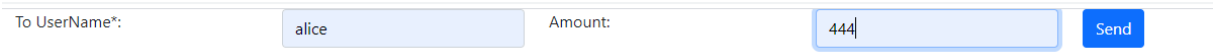
```
        <form id='login' action='transactions.php' method='post' accept-
charset='UTF-8'>
            <div class="row">
                <div class="col-sm">
                    <label for='ToUserName' >To UserName*:</label>
                </div>
                <div class="col-sm">
                    <input type='text' class="form-control mb-2 mr-sm-2"
name='ToUserName' id='ToUserName'  maxlength="50" required />
                </div>
                <div class="col-sm">
                    <label  for='Amount' >Amount:</label>
                </div>
                <div class="col-sm">
                    <input type='text'class="form-control mb-2 mr-sm-2"
name='Amount' id='Amount' maxlength="50" required />
                    <input type="hidden" name="fromUserName" value="<?=
$_COOKIE['userName'];?>" />
                </div>
                <div class="col-sm">
                    <input type='submit' class="btn btn-primary mb-2" name='Submit'
id='submit' value='Send' />
                </div>
            </div>
        </form>
```

Note that there is a hidden input called fromUserName whose value is coming from the cookie.

The code which deduced the user name from the token is replaced as :

```
$userName = $_COOKIE['userName'];
```

So, you can now enter as admin and then send some money to alice.

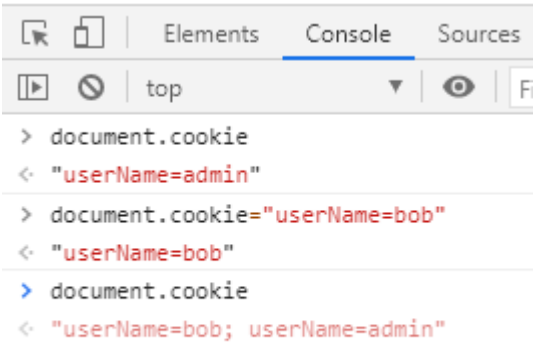


And your transaction will be displayed

List of transactions

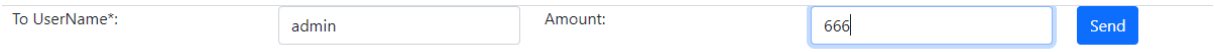
transaction id	from	to	amount
2	bob	admin	222
3	admin	bob	333
4	admin	alice	444

Hijack the cookie in the console window. Overwrite its value to bob.



Now send some money to admin. (Note, you logged in as admin, so you send the money to yourself).

Enter the value to send and press the button.



But if the cookie is hijacked, the money was sent from bob's account.

List of transactions

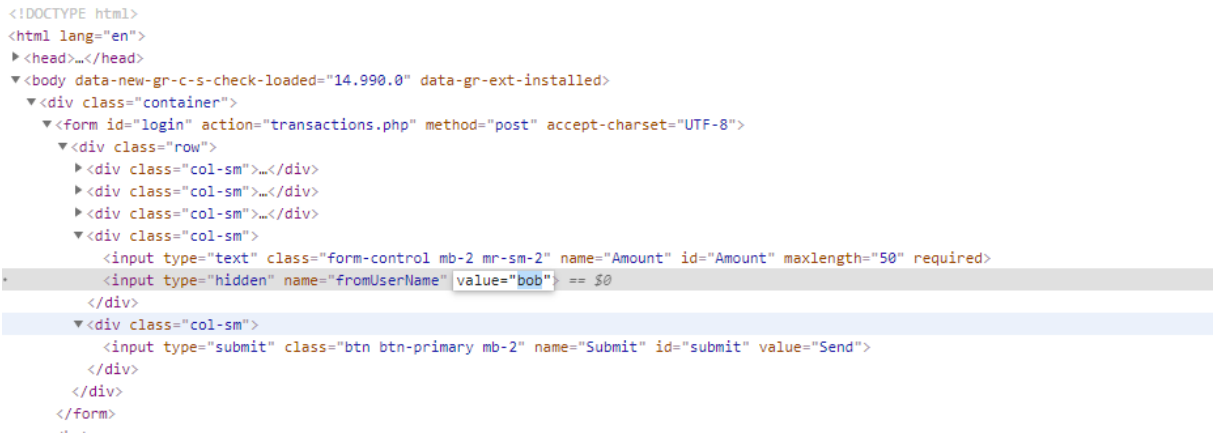
transaction id	from	to	amount
1	alice	bob	111
2	bob	admin	222
3	admin	bob	333
6	bob	admin	666

To prevent that you should check if the same browser from the same IP address is being served. You can see these in the network tab of your browser developer tools.

The solution is to use a token not the plain username field.

Hidden field attack

Now you can secure your cookies. But if you investigate the source of the transactions page that sends money, you will see that there is a hidden field in it



The hacker can change the name ‘bob’. The solution is not to pass sensitive data in the hidden fields, because they can be visible.

Jump URL attack

When you enter the following URL to your browser, you will see the add user form

http://localhost:90/websec/addUser.php

This means that everyone who knows this URL can add a new user to your system. You want to avoid this to allow only logged in users to add new users.

In the server there is a session file that stores information. That could be a good place to store your logged in account name. If that is empty, then addURL page should not be displayed.

```php
<?php
session_start();

if(isset($_SESSION["userName"])){
    echo "Serving user: ". $_SESSION["userName"];
}else{
    die("You have no permission to load the page");
    return;
}
?>
```

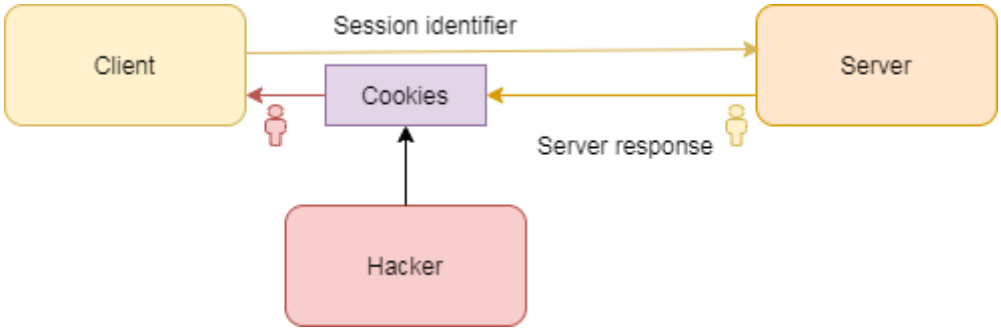
In the login page make sure you save the username into the session

```
    $loginInUser = null;
    $userToken = null;
    while ($row = mysqli_fetch_assoc($result)) {
        $loginInUser = $row["userName"];
        $userToken = $row["userToken"];
        $_SESSION["userName"] = $loginInUser;
        break;
    }
```

Note, you have to start the session first.

Session hijacking

The idea behind the session hijackings that the hackers stole the server session data.



In my computer the remote address is [::1]:90 and my browser is Chrome/87.0.4280.88

So, let's create a function that generates a unique hash of those information.

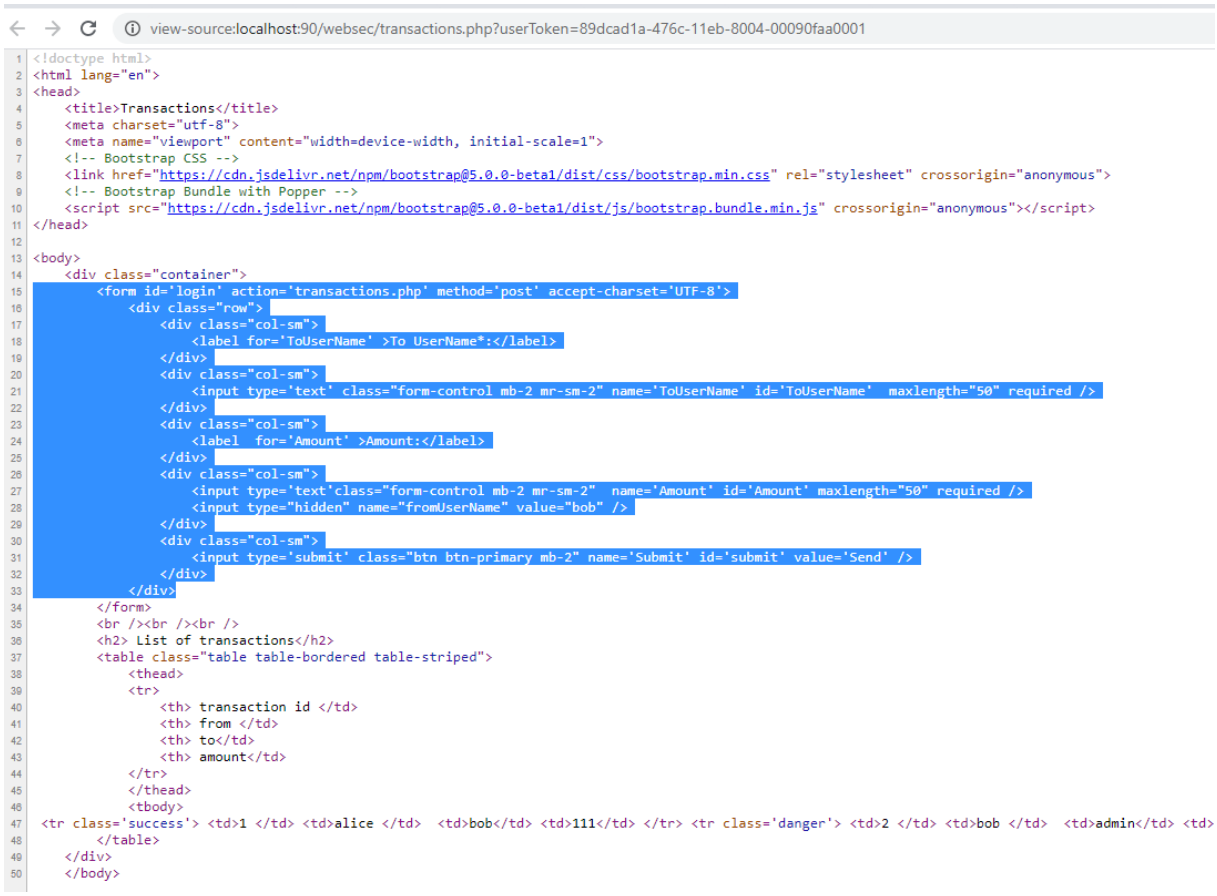
```
// generate a verification string from IP and user agent
function getuserAgentInfo(){
    $user_agent = $_SERVER['HTTP_USER_AGENT'];
    $ip = null;
    if (!empty($_SERVER['HTTP_CLIENT_IP'])) {
        $ip = $_SERVER['HTTP_CLIENT_IP'];
    } elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
        $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
    } else {
        $ip = $_SERVER['REMOTE_ADDR'];
    }
    return  $ip . ":" .$user_agent ;
}
```

Then add some the code that saves the hash of the string to the session file. Subsequent calls should verify if the IP address or the browser changed.

```
// first time login generates token
if(empty($_SESSION['UPCI'])) {
    $_SESSION['UPCI'] = md5(getUserPCInfo(), PASSWORD_DEFAULT);
} else {
    // ask user to re-open the browser
    if(!password_verify(  getUserPCInfo(),$_SESSION['UPCI'] ) ) {
        die("You are not using a valid Token, close the browser and open it
again");
    }
```

Cross site request forgery

In this type of attack that attacker investigates the source code of the form. For example, see

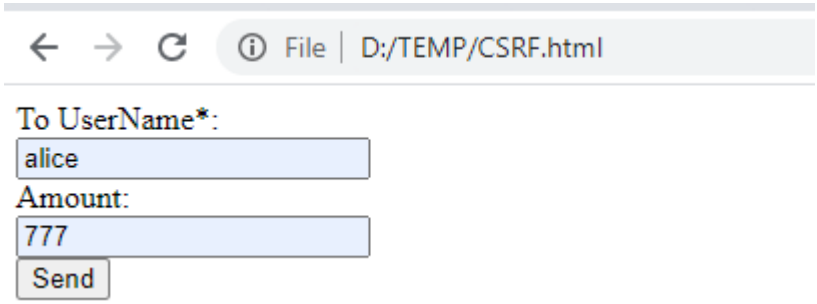


Copy the highlighted text and paste into a file in your local computer.

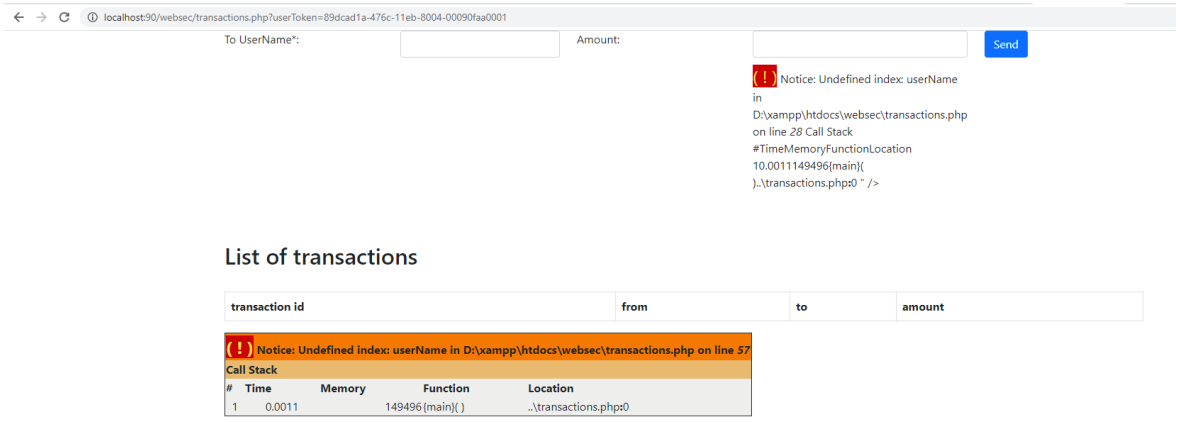
```
        <form id='login'
action='http://localhost:90/websec/transactions.php?userToken=89dcad1a-476c-11eb-
8004-00090faa0001' method='post' accept-charset='UTF-8'>
            <div class="row">
                <div class="col-sm">
                    <label for='ToUserName' >To UserName*:</label>
                </div>
                <div class="col-sm">
                    <input type='text' class="form-control mb-2 mr-sm-2"
name='ToUserName' id='ToUserName'  maxlength="50" required />
                </div>
                <div class="col-sm">
                    <label  for='Amount' >Amount:</label>
                </div>
```

```
                <div class="col-sm">
                    <input type='text'class="form-control mb-2 mr-sm-2"
name='Amount' id='Amount' maxlength="50" required />
                    <input type="hidden" name="fromUserName" value="bob" />
                </div>
                <div class="col-sm">
                    <input type='submit' class="btn btn-primary mb-2" name='Submit'
id='submit' value='Send' />
                </div>
            </div>
        </form>
```

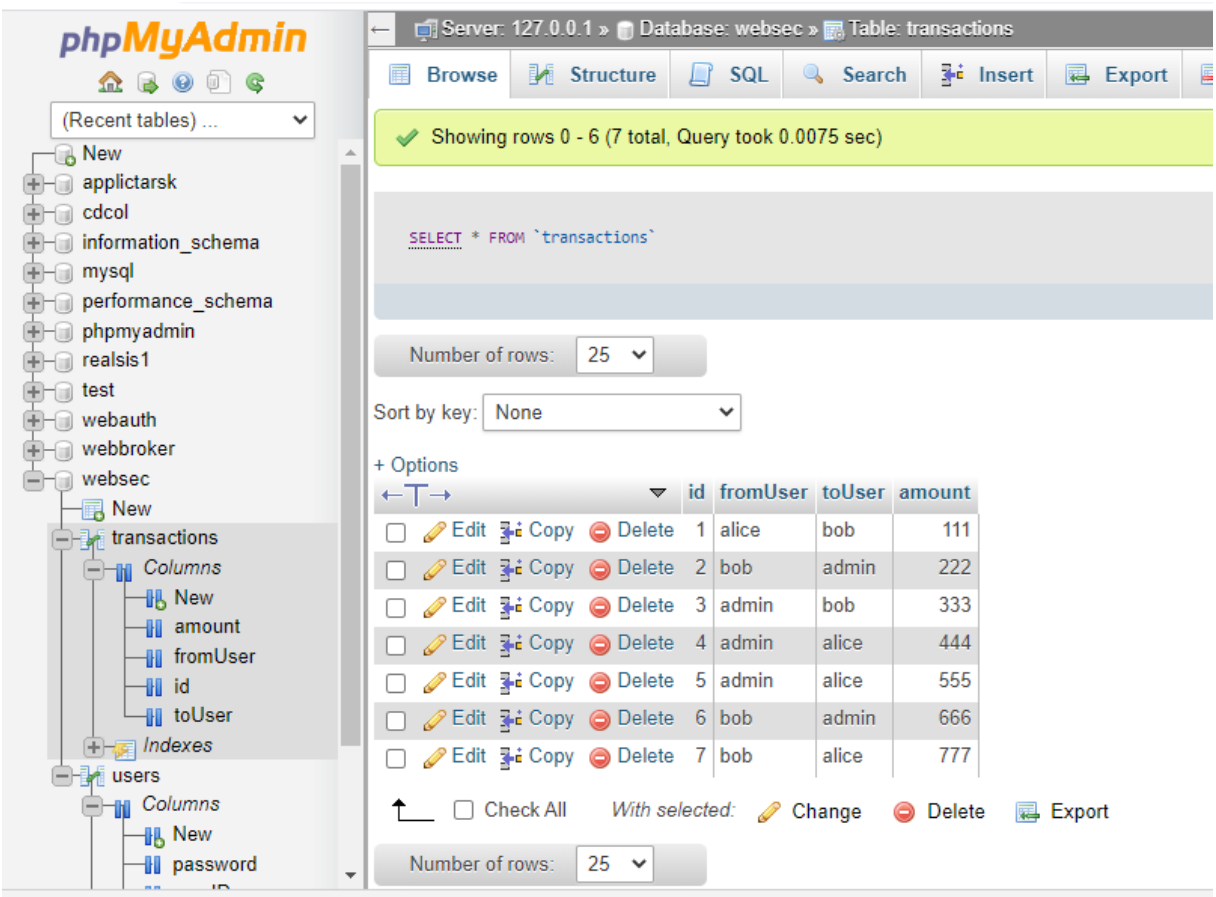
Modify the action parameter of the form tag to point to the server URL. Save the file to a local folder and open in a browser.



Enter the form details and send the money. Although the page displays some error messages like



Your database shows the new record you added.



The solution is to apply a random number to each transaction. We will use

uniqid($prefix, $moreEntopy = true) ,

which yields first 8 hex chars = Unixtime, last 5 hex chars = microseconds. We will apply a random number as prefix and create the md5 hash of the whole string

md5(uniqid(mt_rand(), true));

So, all you need is to add the following code before the form:

```
<?php
// Code to avoid CSRF attack
// make sure it is post process
session_start();
if($_POST) {
    // if token not vaild reject request
    if($_POST["csrf"] != $_SESSION["token"]) {
        echo " not valid request";
        return;
    }
}
// create new token for every new request
$_SESSION["token"] = md5(uniqid(mt_rand(),true));
?>
```

This ensures that the session is started. If the call is a POST call then it compared the post parameter csrf with the token saved to the session file. If validation fails then form quits. Otherwise it generates a random token and saves it to the session file.

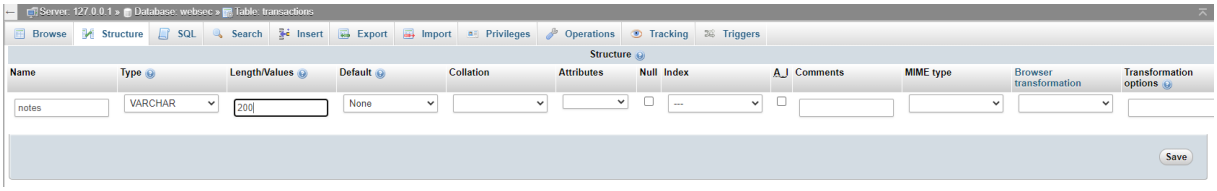
You need to add csrf as a hidden parameter to the form

```
<input type="hidden" name="csrf" value="<?= $_SESSION['token'];?>" />
```

As the result of this, if an attacker steals the whole form then the random token cannot be guessed so no transaction will be executed.

Cross site scripting

Let's add a notes field to the transaction table.



In transactions.php add the new field to the form

```
        <form id='login' action='transactions.php' method='post' accept-
charset='UTF-8'>
            <div class="row">
                <div class="col-sm">
                    <label for='ToUserName' >To UserName*:</label>
                </div>
                <div class="col-sm">
                    <input type='text' class="form-control mb-2 mr-sm-2"
name='ToUserName' id='ToUserName'  maxlength="50" required />
                </div>
                <div class="col-sm">
                    <label  for='Amount' >Amount:</label>
                </div>
                <div class="col-sm">
                    <input type='text'class="form-control mb-2 mr-sm-2"
name='Amount' id='Amount' maxlength="50" required />
                    <input type="hidden" name="fromUserName" value="<?=
$_COOKIE['userName'];?>" />
                    <input type="hidden" name="csrf" value="<?=
$_SESSION['token'];?>" />
                </div>
                <div class="col-sm">
                    <label for='Notes' >Notes:</label>
                </div>
                <div class="col-sm">
                    <input type='text'class="form-control mb-2 mr-sm-2"
name='Notes' id='Notes' />
                    </div>
                <div class="col-sm">
                    <input type='submit' class="btn btn-primary mb-2"
name='Submit' id='submit' value='Send' />
                </div>
            </div>
```

```
        </form>
```

In the list of transactions add the new field to the header

```
        <h2> List of transactions</h2>
        <table class="table table-bordered table-striped">
            <thead>
            <tr>
                <th> transaction id </td>
                <th> from </td>
                <th> to</td>
                <th> amount</td>
                <th> notes</td>
            </tr>
            </thead>
```

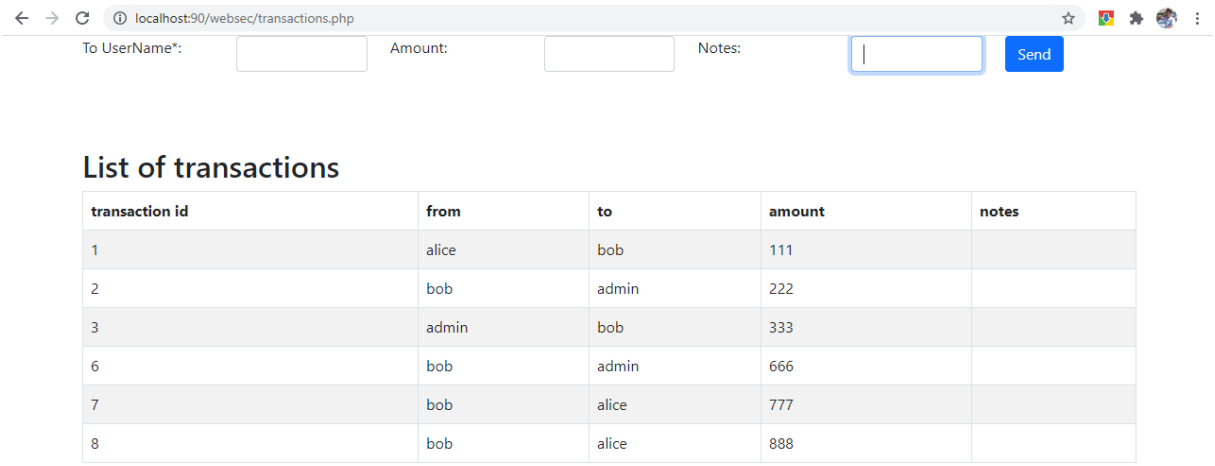
And to the table body as well

```
    while ($row= mysqli_fetch_assoc($result)) {
        $rowColor ="class='success'";
        if($row["fromUser"]==$userName) {
            $rowColor ="class='danger'";
        }
        echo " <tr ". $rowColor .">";
        echo " <td>". $row["id"] ." </td>";
        echo " <td>". $row["fromUser"] ." </td>";
        echo "  <td>". $row["toUser"]."</td>";
        echo " <td>". $row["amount"]."</td>";
        echo " <td>". $row["notes"]."</td>";
        echo " </tr>";
    }
```

Insert query needs modification, too

```
    $query ="insert into transactions(fromUser, toUser, amount, notes)
      values ('". $_POST['fromUserName'] ."','". $_POST['ToUserName']
."','". $_POST['Amount'] ."','". $_POST['Notes'] . ")";
```

Now the transactions page looks like



Enter the following javascript code to Notes field:

<script>alert('XSS testing')</script>

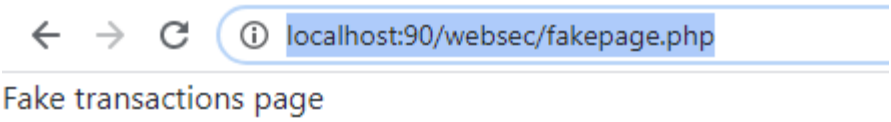
Now, whenever you list the transactions the javascript code will be executed.

You can execute any code you want. Create for example a fakepage.php file

And enter the script as

<script> document.location="http://localhost:90/websec/fakepage.php";</script>

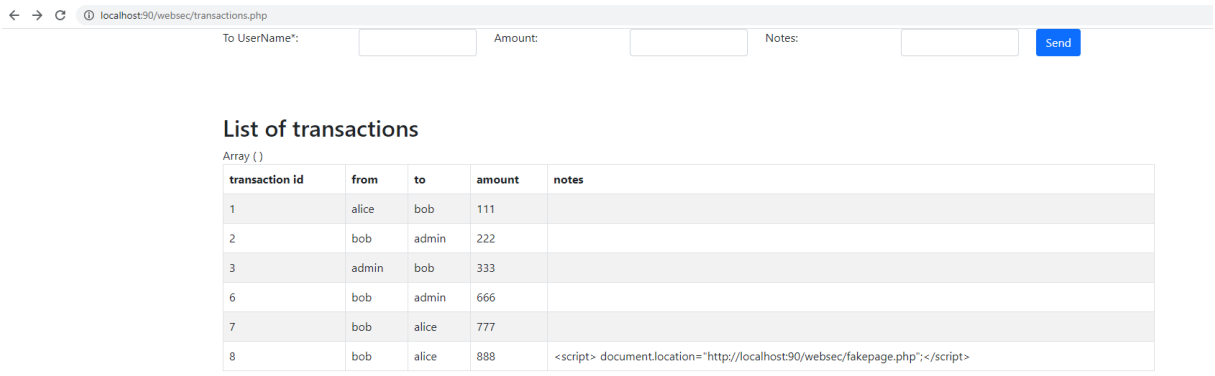
And your transaction file will be redirected.



The fix is quite straightforward: encode the Notes field before printing out the row

```
while ($row= mysqli_fetch_assoc($result)) {
    $rowColor ="class='success'";
    if($row["fromUser"]==$userName) {
        $rowColor ="class='danger'";
    }
    echo " <tr ". $rowColor .">";
    echo " <td>". $row["id"] ." </td>";
    echo " <td>". $row["fromUser"] ." </td>";
    echo "  <td>". $row["toUser"]."</td>";
    echo " <td>". $row["amount"]."</td>";
    echo " <td>". htmlentities($row["notes"])."</td>";
```

```
        echo " </tr>";
    }
```



SQL injection

Investigate the query in login page

```
select * from users  where userName='admin' and password='admin'
```

The first string constant comes from the username field, the second one from the password field. What if you enter *o' or '1' = '1* in the password field? The query will read as

```
select * from users  where userName='admin' and password='o' or '1' = '1'
```

This skips the password validation as the or clause will be always true. As result you will be able to log in as admin.

The solution is to use prepared statements.

The php manual on the prepared statements (https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php) says:

The MySQL database supports prepared statements. A prepared statement or a parameterized statement is used to execute the same statement repeatedly with high efficiency.

The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use.

The MySQL server supports using anonymous, positional placeholder with?

```
$query ="select userName from login  where userName=? and password=?" ;
$loginInUser=null;
/* create a prepared statement */
if ($stmt = $mysqli->prepare($query)) {

    /* bind parameters for markers */
    $stmt->bind_param("ss", $_POST['userName'],$_POST['password']);
    /* execute query */
    $stmt->execute();

    /* bind result variables */
    $stmt->bind_result($loginInUser);

    /* fetch value */
    $stmt->fetch();
```

```
    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
```

The first parameter of bind_parameter is a type description character:

Character	Description
i	corresponding variable has type integer
d	corresponding variable has type double
s	corresponding variable has type string
b	corresponding variable is a blob and will be sent in packets

In the sample code above there are two string type characters, hence the ‘ss’ type.

Directory traversal

In many operating systems there are special characters that represent traversing to certain directory. for example ../ (dot dot slash) means to climb up a directory level.

```
<?php
$fname= 'myfile.php';
if (isset($_COOKIE['FNAME'])) {
    $template = $_COOKIE['FNAME'];
}
include "/home/users/oliver/work/" . $fname;
```

in a Linux system set the cookie value as

Cookie: FNAME =../../../../../../../../../etc/passwd

That would output as

root:ge4jPT12ivkL7:0:1:System Operator:/:/bin/ksh

daemon:*:1:1::/tmp:

To prevent such attacks there are some possible solutions:

1. do not allow special characters like dot dot, slash, backlash, or their encode version:
   - %2e%2e%2f which translates to ../
   - %2e%2e/ which translates to ../
   - ..%2f which translates to ../
   - %2e%2e%5c which translates to ..\
   - %c1%1c
   - %c0%af
2. convert file names to absolute path and check if all starts with a common document root.

References

[1] https://owasp.org/www-community/attacks/

[2] https://www.php.net/manual/

[3] https://github.com/hussien89aa/swa

[4] Understanding Computer Browser Cookies: Internet Convenience, Information Security and Computer Privacy http://internet-security.suite101.com/article.cfm/understanding_computer_browser_cookies#ixzz0swFXNajh