

# LAB REPORT

## **5822UE Exercises: Security Insider Lab II - System and Application Security (Software-Sicherheit) - SS 2022**

### **Part 1 Web Application Vulnerabilities - 1**

#### **Group 2**

Pratik Baishnav - 90760 (baish01@ads.uni-passau.de)

Date : 4th May, 2022 - 11th May, 2022

Time: Wednesday (14:00 - 20:00)

Location: ITZ SR 002

Organiser : Farnaz Mohammadi (Farnaz.Mohammadi@uni-passau.de)

## Exercise 1 : Setup

1. **On your Linux machine install apache2, PHP, MySQL, and PHPMyAdmin. Hint: Make sure the PHP version is compatible with the web application in step 2.**

On my Linux machine(Ubuntu 20.04) I installed **XAMPP** environment. It is one of the most popular development environments for PHP developers. Since the project requires a suitable PHP version which is below 7. I installed old XAMPP version **5.6.35** which comes up with **Apache 2.4.33**, **PHP 5.6.35**, **MySQL 5.0.11**, and **PhpMyAdmin 4.8.0** and other modules such as Tomcat, Perl, etc.

We can download the environment from:

<https://sourceforge.net/projects/xampp/files/XAMPP%20Linux/5.6.35/>

After downloading, to run the file we have to make that executable by using the command: **sudo chmod +x xampp-linux-x64-5.6.35-0-installer.run** then type **./xampp-linux-x64-5.6.35-0-installer.run** . After successful installation, you will get the XAMPP control panel or in my case used command **/opt/lampp/lampp start** to run XAMPP. This command runs Apache, PHP, MySQL, ProFTPD.

```
lsick@sick-core:~$ sudo /opt/lampp/lampp start
Starting XAMPP for Linux 5.6.35-0...
XAMPP: Starting Apache...ok.
XAMPP: Starting MySQL...ok.
XAMPP: Starting ProFTPD...ok.
```

2. **Download the zipped file of the Vulnerable Bank web application and extract it to our web server.**

After downloading **vBank.zip** we have to extract this file and there were 3 folders and one SQL file. Now, following the steps :

- ❖ Move files of htdocs to /opt/lampp/htdocs.
- ❖ Move files of etc to /opt/lampp/etc.
- ❖ Move folder pages to /opt/lampp/

### 3. The configuration is located under “./etc/config.php”

**config.php** contains all the configurations for the database, hosting, and other related files so we have to change the path in the config.php .

```
<?php
/*
** FILE: config.php
** PURPOSE: contains all configuration information
** AUTHOR(S): Daniel Schreckling
*/

global $htbconf;

$xorValue = 0x0BADCODE;
// paths
$htbconf['paths/prefix'] = '/opt/lampp/';
$htbconf['paths/pages'] = 'pages/';

// url of this bank
// $htbconf['web/baseurl'] = 'http://www.unsecurito.it/';
$htbconf['web/baseurl'] = 'http://127.0.0.1/';
```

### 4. From phpmyadmin import the file “vbank.sql” to setup the database and tables in MySQL.

Firstly, we have to create a new Database whose name I set as ‘vbank’ and then import the database (vbank.sql).

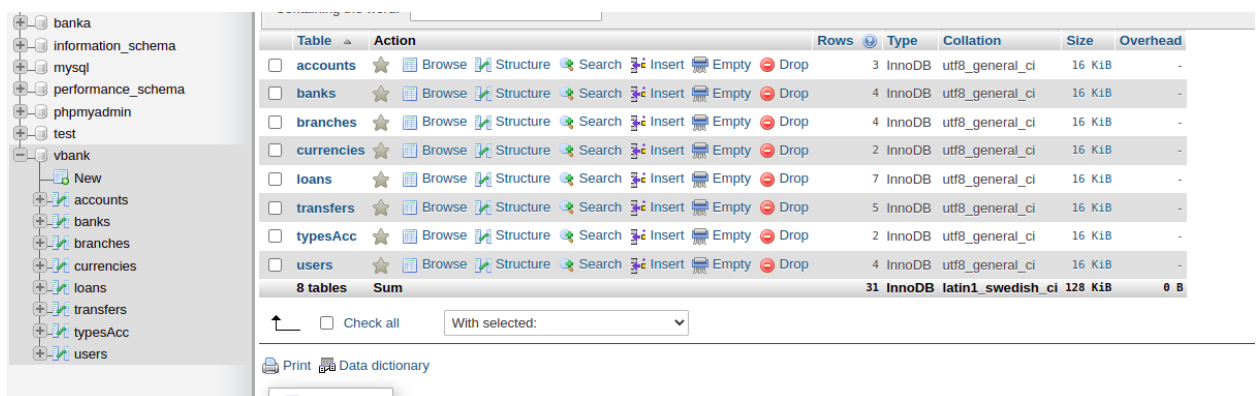


Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> accounts	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> banks	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> branches	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> currencies	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> loans	★ Browse Structure Search Insert Empty Drop	7	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> transfers	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> typesAcc	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8_general_ci	16 KiB	-
<b>8 tables</b>	<b>Sum</b>	<b>31</b>	<b>InnoDB</b>	<b>latin1_swedish_ci</b>	<b>128 KiB</b>	<b>0 B</b>

**5. Test if the application works from your web browser.**

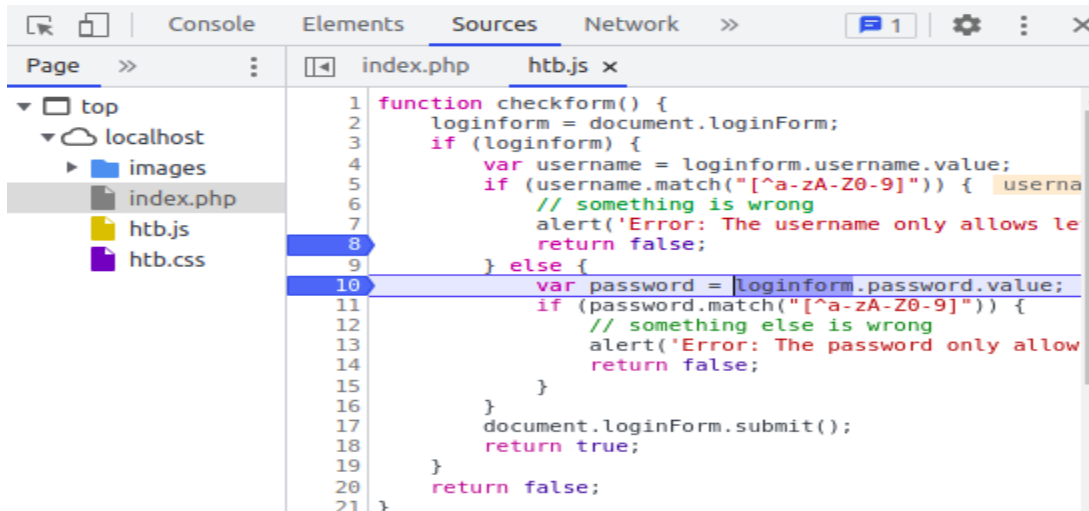


It works ;) .

## Exercise 2 : Client/Server Side Scripting

- 1. Identify a mechanism which protects the login process (not on the server) and briefly describe the general security problem with this implementation.**

The mechanism which protects the login process except on the server is client-side scripting. Client-end scripts are embedded in a website's HTML markup code. Based on the experience I had, javascript must be used to validate this form so, while inspecting I found that it uses a javascript function called `checkform()`. This function takes the value of the username and password field and uses the regular expressions in the if-else condition.



Protection on the client-side is a safe approach for providing better security but on the server-side, if we are unable to use validation or protection mechanism then it is useless to do it only on the client-side. As it has several drawbacks :

- ❖ SQL Injection
- ❖ Cross-Site Scripting
- ❖ Cross-Site Request Forgery
- ❖ Security Misconfiguration and etc.

## 2. List the steps a client could use to circumvent this restriction. Test your approach. Make sure it works.

Clients can perform the following approaches to circumvent this restriction:

- ❖ To avoid the validation process, change the function while submitting the login form.
- ❖ By disabling the javascript in the browser.
- ❖ From the inspection, we can observe that it uses the GET method to request the server which means that we can perform SQL injection in the URL to bypass the form activity.

→ Using special characters while logging we get the error result:

<http://localhost/login.php?username=test&password=asdf@,%22>



Though the special character via URL was sent, it successfully processed and threw an error message which implies that there is no server-side validation. This is a huge issue in web development which can lead to various kinds of SQL attacks.

→ Now Using SQL injection we try to bypass via URL:  
<http://localhost/login.php?username=&password=' or '1=1'> which successfully logs in to the home page of Alex.



### 3. Propose a better solution by providing commented source code.

In order to protect a website, only client-side validation is not enough. The solutions to overcome such problems to secure a website are:

- ❖ Use server-side validation along with the client-side validation which will make the security robust.
- ❖ Use white-space validation.
- ❖ Use POST method instead of GET method while transferring the sensitive data.

Use the following code on the server-side while validating (which is on login.php)

```
/* if (!preg_match("[^a-zA-Z0-9]", $username, $password)) {
    $_SESSION['error'] .= "<p>Server Side Validation
worked!</p>";
    htb_redirect(htb_getbaseurl());
    exit();
}*/
```

Now if we try to input special characters via URL it will throw an error.



### Exercise 3 : SQL Injection

It is a time to find a "username" that allows you to enter the system without knowing the username or the password.

1. **Find a query to enter the system (without manipulating the data used by the web application, you should get access on behalf of another user). Show this query and briefly explain it using the source code at hand.**

From the above exercise, we know that via URL we can perform SQL injection attacks and from the URL <http://localhost/login.php?username=&password=> we can have a guess about the query structure which might be

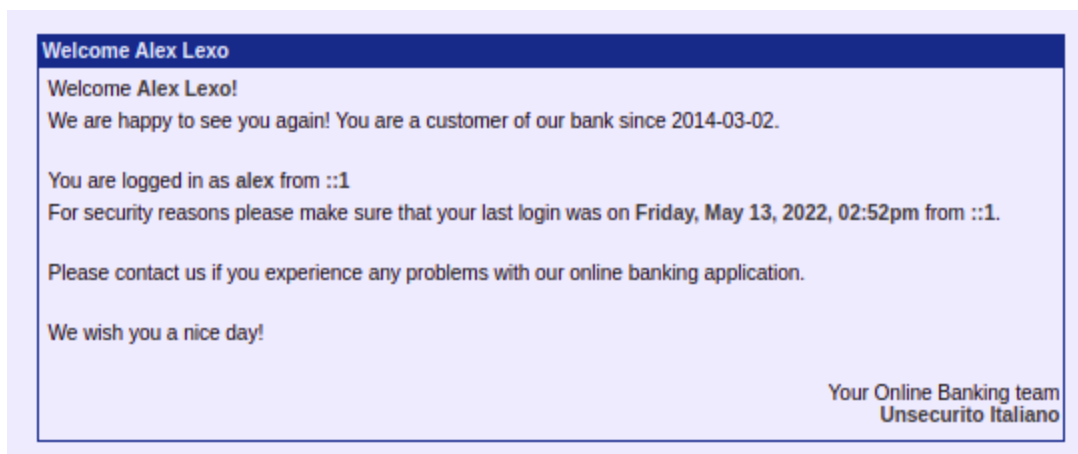
Select username, password From "some\_table\_name" where username = 'username\_data' and password = 'user's\_password\_data';

So from the observation and guesses of Exercise 2, we get the URL and we were able to bypass javascript validation easily. Now we can try the error method using SQL injection techniques to obtain a real query.

**2. Fire your attack...!!! Why is your attack successful? & Which checks and mechanisms can prevent this failure (mention at least two mechanisms).**

As mentioned in the previous answer now we simply try to modify the query and try to log in without a username and password.

<http://localhost/login.php?username=&password=' or '1=1'>



Boom!!! As shown in the above screenshot, I was able to get access without a username and password because our access of query is right which was modified as '**or '1=1'**'.

This condition is always true as the username and password are left blank which gives a false result but after the addition of '**or '1=1'**' condition with a true statement which makes the whole statement true i.e. **True** and **True** is always **True**.

To prevent such attacks or such failures in the login system following mechanisms should be implemented:

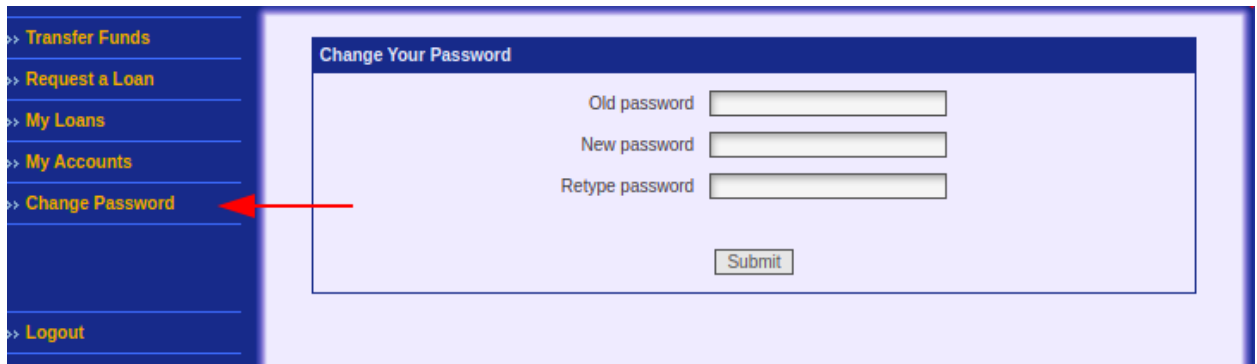
- ❖ First of all, validate user input and request string properly on both the client and the server side.
- ❖ Use of stored procedures which can encapsulate the SQL statements and treat all input as parameters.
- ❖ Use of prepared statements which works by creating the SQL statements first and then considers all submitted user data as parameters in such a way that there is no



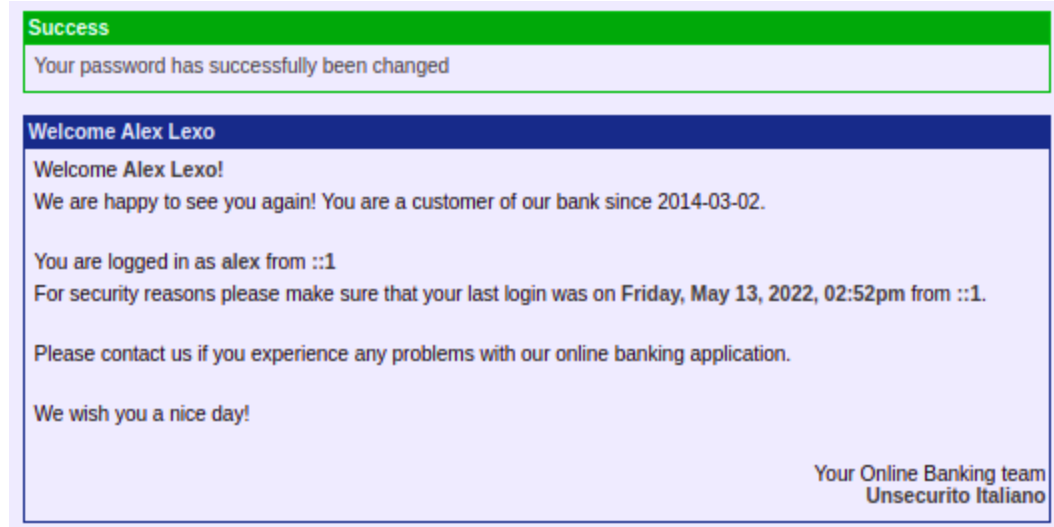
effect on the SQL statement.

**3. Change the password of the user you are logged in with. Briefly describe your actions and indicate the source code allowing for this attack.**

Now that we are successfully logged inside the system, we can see various options provided. There is one option to change the password of the user, click on that option a page regarding password change will open.



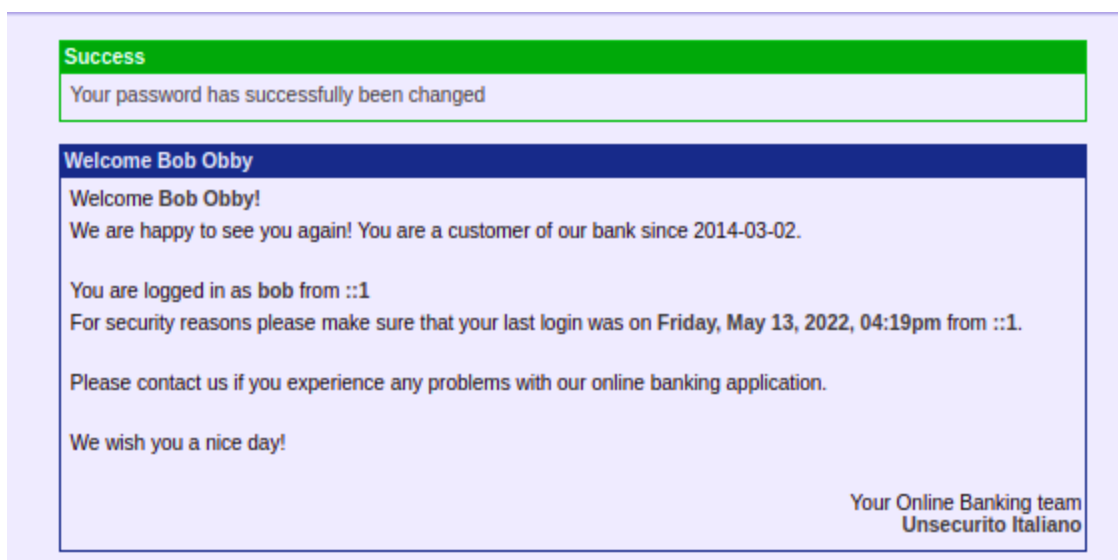
- Then, type ' or '1=1' as old password. It bypasses the old password by making the statement **True**.
- Now type the desired password and confirm it which will successfully change the password of the user.



#### 4. Change the password of a known user (you know the login name of the user)!

We now know that there is a user called Alex but what about others? There might be others too, so in order to confirm that I tried common names which are frequently used in the field of IT-Security which are Alice, Bob, Eve, Mallory, etc. Since we now have the username to try but still don't have any password therefore in order to bypass the password with the following URL: <http://localhost/login.php?username=bob'--'&password=>

Which basically checks whether there is a valid user in the database and if there is a valid user it neglects the password by commenting out the rest of the URL. I tried using other names too but with Bob, I was able to log in using the username **"Bob"** and bypass the password using **'--'**. After logging in as Bob we follow the same steps as question no. 3 to change the password.



### Exercise 4 : SQL Injection - continued

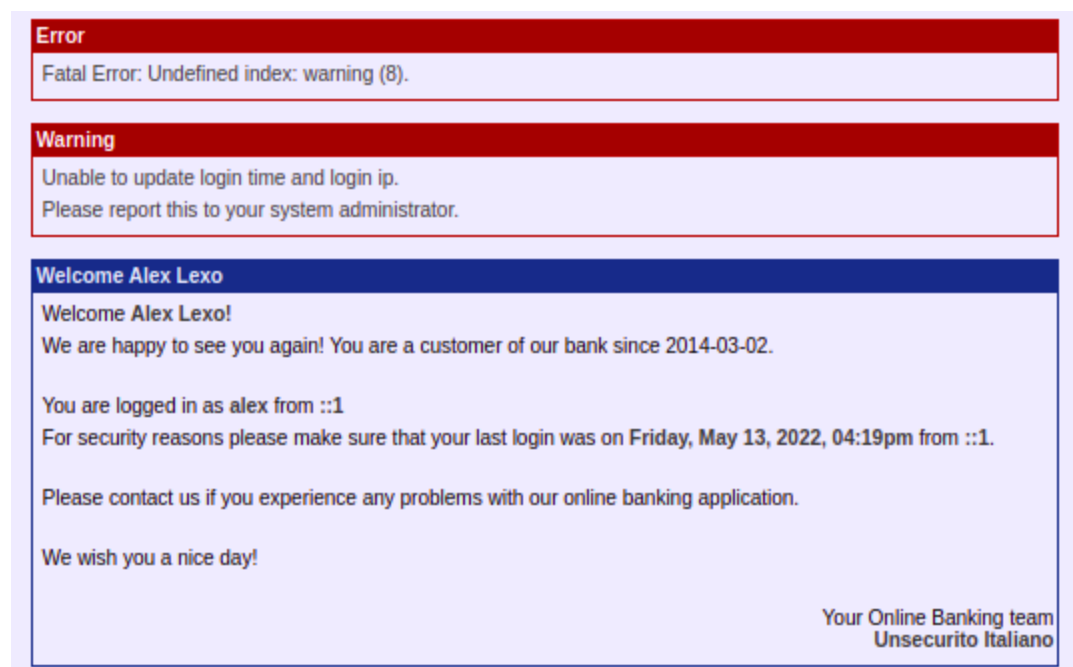
In the last exercise, you logged in without manipulating the attached database. Now you should try to enter the bank without using the login information of another person. Create your own account.

# 1. Briefly describe how did you retrieve the information you need to create a new user.

We have to create a new user in the system but to create a new user, first, we need to know all the information about the database, table schema, name of the tables, how many columns the table has, etc. We assume that it uses MySQL as it is the most popular database system for PHP. Now, we find the required necessary information about the table.

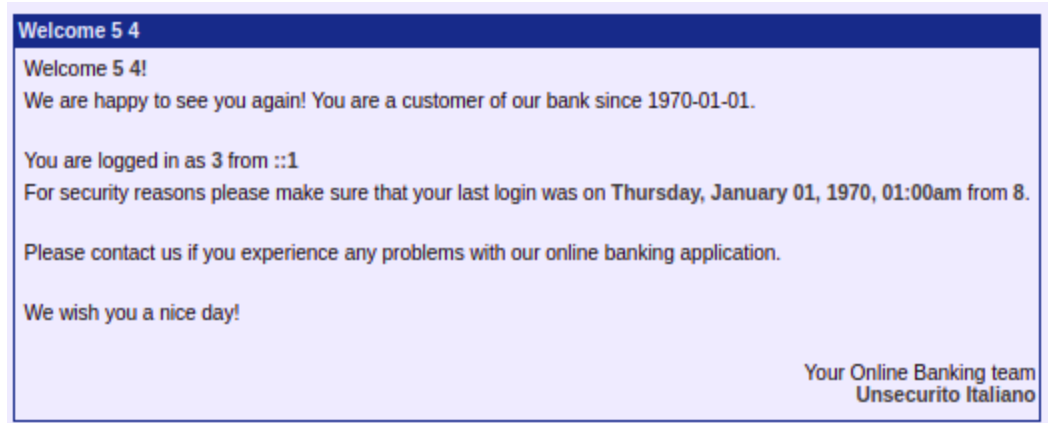
- First, we need to know how many columns are there to insert into the desired table. For this, we use the ‘**Order by**’ statement as this statement commands the database how to order the result. We will continue to increase the number until we can no longer log in.

<http://localhost/login.php?username=alex&password=' or '1=1' order by 1--+'>



- We could log in exactly 8 times but on the 9th time we could not log in and the system threw an error. Now we know that there are 8 columns in the table and we need to select all the columns in that table for which we use the ‘**Union Select**’ statement.

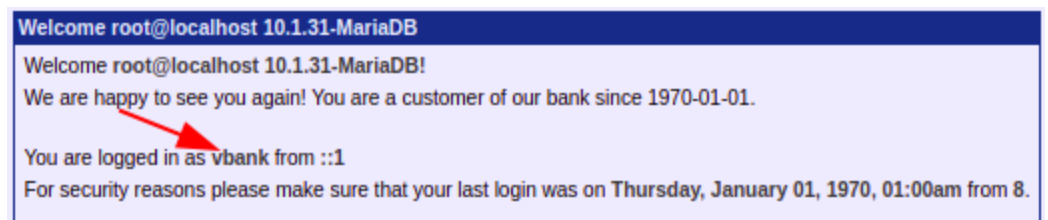
<http://localhost/login.php?username=alex&password=' Union Select 1,2,3,4,5,6,7,8 --+'>



→ Here, a virtual table is created using the 'Union Select' statement and from this table, we can observe the number "Welcome 5 4" and "You are logged in as 3". From these numbers we can extract the required information. So, from these numbers we can safely assume that no.5 represents **Name**, 4 represents **Lastname** and 3 represents **username**.

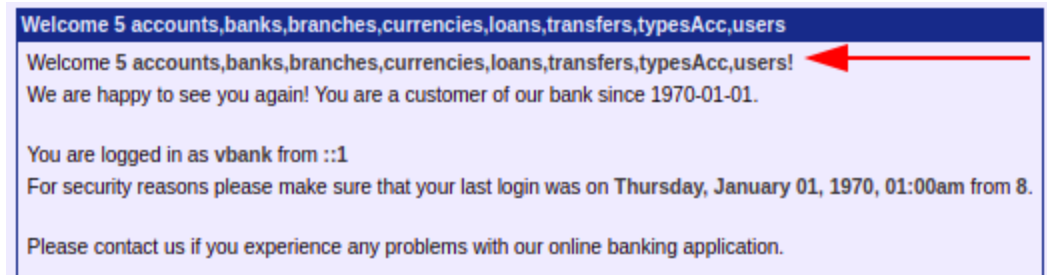
→ Now, we can pass the various in-built sql functions such as 'database()', 'version()', 'user()' etc. to retrieve the information. And from the database() function we get the name of the database which is 'vbank' which we can replace in the query instead of number 3, 4, and 5.

[http://localhost/login.php?username=alex&password=' Union Select 1,2,database\(\), 4,5,6,7,8 --+'](http://localhost/login.php?username=alex&password=' Union Select 1,2,database(), 4,5,6,7,8 --+')



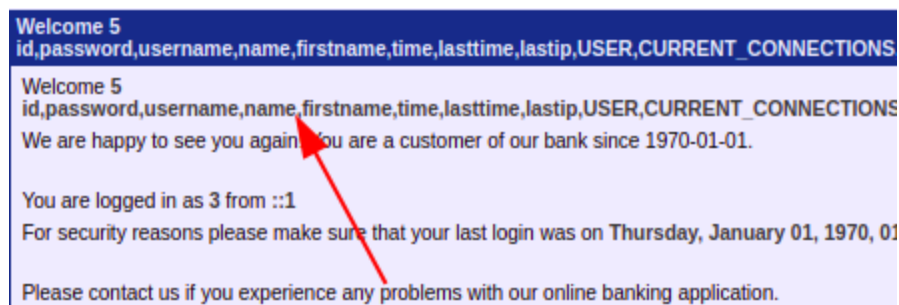
→ Now, we try to get all the table names from the database using the 'Information\_Schema.tables' statement.

[http://localhost/login.php?username=alex&password=' union select 1,2,database\(\).group\\_concat\(table\\_name\),5,6,7,8 from information\\_schema.tables where table\\_schema=database\(\)--+'](http://localhost/login.php?username=alex&password=' union select 1,2,database().group_concat(table_name),5,6,7,8 from information_schema.tables where table_schema=database()--+')



- After getting the list of table names, we need to get the column names of the table 'user' for which 'Information\_Schema.columns' statement is used.

[http://localhost/login.php?username=alex&password='union select 1,2,3,group\\_concat\(column\\_name\),5,6,7,8 information\\_schema.columns where table\\_name='users'--+'](http://localhost/login.php?username=alex&password='union select 1,2,3,group_concat(column_name),5,6,7,8 information_schema.columns where table_name='users'--+')



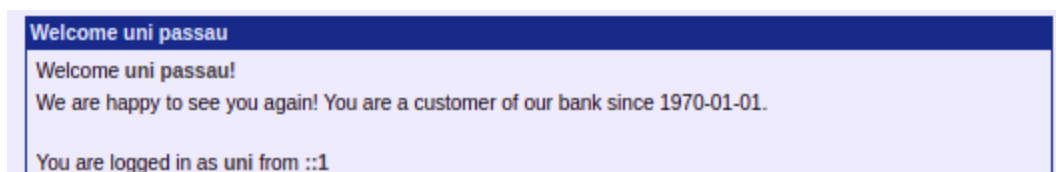
- Now, we have all the information needed to create a new user.

## 2. Briefly describe how did you retrieve the information you need to create a new user.

Since we know the column names of the table 'user' we simply use the 'Insert' statement to create a new user in the desired column field.

[http://localhost/login.php?username=alex&password=' ; Insert into users \(password, username, name, firstname\) values\('unipassau','uni','passau','uni'\)--+'](http://localhost/login.php?username=alex&password=' ; Insert into users (password, username, name, firstname) values('unipassau','uni','passau','uni')--+')

Hence, a new user 'Uni' is created. Here, ';' is a separate query that the MySQL interpreter interprets as an extra query and executes it.



**3. Give an additional precaution (apart from those you have given already) that may prevent this manipulation.**

The additional precautions which can be taken into consideration are:

- ❖ Use secure service like HTTPS.
- ❖ Update database system and server as well as avoid using deprecated libraries or methods.
- ❖ Data type validation.
- ❖ Least privilege to edit the database.

**4. Implement at least one protection mechanism. Show it in action preventing SQLinjection.**

As a protection mechanism I used '`mysqli_real_escape_string()`' function to remove special characters from both username and password variables along with **prepared statement** to prevent unauthorized login and unauthorized manipulation of the database.

```
$username = mysqli_real_escape_string($link, $username);  
$password = mysqli_real_escape_string($link, $password);  
  
$sql = ("SELECT * FROM " . $htbconf['db/users'] . " where " .  
$htbconf['db/users.username'] . "= ? and " .  
$htbconf['db/users.password'] . "= ? ");  
  
$stmt = $link->prepare($sql);  
  
$stmt->bind_param('ss', $username, $password);  
  
$stmt->execute();
```

Now, if we try to get into the system and create a new user or try SQL injection it won't be successful.

## Exercise 5 ; Request Manipulation

With the same type of SQL injection, you studied in the first few exercises, you would be able to create your own account with a lot of money. However, the database administrator took some precautions to prevent this. Nevertheless, we are able to log in as a legitimate user with some valid accounts and we can transfer money to other accounts ...

**1. 1. Install a tool that allows you to manipulate requests of your browser. Briefly discuss your choice.**

I used 'Burp Suite' tool because it is one of the easy-to-learn and famous tools available. It has so many functionalities which can be useful for future works as well.

**2. Briefly describe the steps to request a loan from your victim will benefit.**

After configuring the Burp Suite's settings we can intercept the victim's request and response from the server with the tool easily. We can totally overserve information like which pages are being requested, which HTTP methods are being used, the version of the system, values of the page, etc. So, using the tool we can do the following:

- ❖ The Loan request page from the vbank application contains 5 fields namely, Credit A/c no., Debit A/c no, Loan Amount, Loan Period, and Interest rate. From which only Loan Amount's can be set to any value of the user's choice being the text input field and the rest of the fields are select inputs.
- ❖ We enable the intercept mode on of the Burp Suite tool and after doing so every request that the user sends to the server is accessed by us in the middle first, then we forward it to the server. Again, the response is intercepted by us and we forward it to the user. By manipulating the HTTP requests we could change the values for all the inputs as per our choice.

Loan Request - Confirmation

Credit Account No. 11111111

Debit Account No. 33333333

Loan Amount 5555 USD

Loan Period 100 year(s)

Interest rate 0 %

Confirm

Cancel

- ❖ It can be seen that the original data like Interest rate which should be 4.2% is changed to 0% and the year whose value should be 1year is changed to 100years.

My Loans						
Please click on the links to get more details on loan						
Credit Ac.	Debit Ac.	Issued	Amount	Period	Interest	
11111111	33333333	2014-03-27	1000 USD	1	4.2%	<a href="#">Amortisation</a>
11111111	33333333	2022-05-08	555 USD	1	10%	<a href="#">Amortisation</a>
11111111	33333333	2022-05-08	888 USD	1	44.2%	<a href="#">Amortisation</a>
11111111	33333333	2022-05-13	5555 USD	100	0%	<a href="#">Amortisation</a>

- ❖ The attack was successful which can be seen on the user's account loan page. Here, the victim can take multiple benefits as he can alter interest rate, change values for the loan period, change the account information, etc. according to his will.

### 3. What enables this type of attack? Identify the respective source code and give the vulnerability a name.

- ❖ The lack of enough server-side validation enabled this attack. Even though the VBank uses client-side validation it is not enough as the data can be tampered easily.
- ❖ The attack is called Web Parameter Tampering which leads attackers to tamper with data.

```

if(isset($http['submit']) && $http['submit'] == "Confirm") {
    $sql="update ".$htbconf['db/accounts']." set ".$htbconf['db/accounts.curbal']. "=".$htbconf['db/acc
    mysql_query($sql, $db_link);
    $sql="insert into ".$htbconf['db/loans']." (".$htbconf['db/loans.owner'].", ".$htbconf['db/loans.a
    mysql_query($sql, $db_link);
    var_dump($_SESSION);
    $_SESSION['success'] .= "<p>Loan granted</p>\n";
    htb_redirect(htb_getbaseurl(). 'index.php?page=htbaccounts');
    exit;
}

```



#### 4. Show how to fix this vulnerability. Implement your patch and briefly summarize your changes.

To fix this vulnerability, a strong server-side validation function must be implemented and also on numerous fields. Here, I validated the Interest value while requesting the Loan.

```
if($http['creditacc'] != 4.2) {
    $_SESSION['error'] .= "<p>t Interest rate tampered!</p><p>Please check your data and enter appropriate values!</p>";
    htb_redirect(htb_getbaseurl().'index.php?page=htbloanreq');
    exit;
}
```

If this validation matches, then only the user can proceed to take a loan from the bank. Similarly, like this other fields like Loan Periods, Amounts, and Account no. also should be validated strongly and should not be able to be tampered with.

## Exercise 6 : Cross Site Scripting - XSS

### 1. Briefly explain what is an XSS attack in your own words?

Cross-Site Scripting(XSS) is a type of injection where malicious codes are injected into the client-end script. In this attack, an attacker uses a web application server for storing and spreading harmful or malicious codes to the user's web pages. XSS attacks can be of different types, they are Stored XSS, Reflected XSS, and DOM Based XSS. For example, an attacker tricks a victim into opening a malicious link or message. When the victim user opens this link or message thinking it might be the part of the website but in reality, it is a trap set up by the attacker to steal cookies and many other sensitive information about/from the user.

### 2. Perform an XSS attack that opens a window with a nice message while another user uses his account. Briefly describe the required actions.

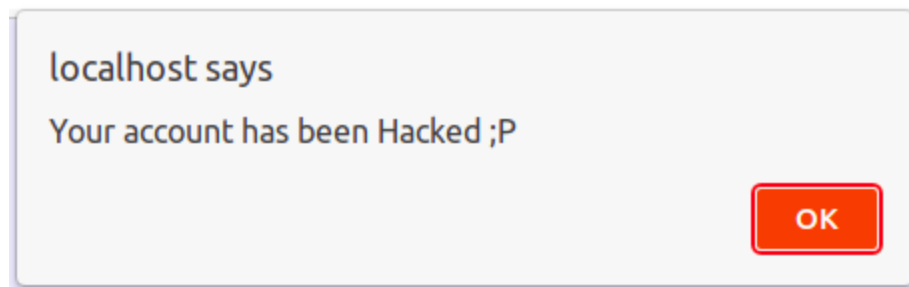
On the website, to perform an XSS attack we need to find a place to transfer our malicious code to another end user. In the **Transfer Fund** option there exists a **Remark** field/option from where we can send the malicious code.

- ❖ At first, fill out all the fields to whom we want to send the malicious code and attack.
- ❖ After filling all the fields, in the **Remark** field, we write a short test code and submit it, the codes will be saved in the database in the remark column.

```
<script>alert("Your account has been HACKED ;P")</script>
```

The screenshot shows a web application interface with a sidebar on the left containing links: >> Transfer Funds, >> Request a Loan, >> My Loans, >> My Accounts, >> Change Password, and >> Logout. The main content area is titled 'Transfer Funds' and contains the following fields: Source Account No. (dropdown menu with '11111111'), Destination Bank Code (dropdown menu with '41131337 (V-Bank)'), Destination Account No. (text input with '33333333'), Amount (text input with '5888' and a 'USD' label), and Remark (text input with '<script>alert('Your account has been HACKED ;P')</script>'). A 'Transfer' button is at the bottom. Red arrows highlight the 'Transfer Funds' link in the sidebar and the 'Remark' field.

- ❖ Now, when the victim user opens the page containing the Remark, even though the developers developed the page just to display the remark, the browser will compile our code which is written in JavaScript to pop up a message displaying the message “Your account has been HACKED ;P”



### 3. What obvious checks did the developers miss to apply?

The developers failed to sanitize the code. Sanitization of codes at the input point and the output point is crucially important to prevent these kinds of XSS attacks.

### 4. Identify the respective source code and eliminate the vulnerability(ies). Briefly summarise your changes.

We know that the code was injected in the **Remark** field when the transfer occurred. Therefore, we need to know and look at the source code which gets the value from the

Remark field which is available on the **htbdetails.page**. The source is:

```
$transfersStr .= "<td>".$row[5]."</td>\n";
```

I used the php function **filter\_var(Filter\_Sanitization\_String)** which performs sanitization removing all the bad code from the input data. My changed code is as follows:

```
$transfersStr .= "<td>".filter_var($row[5], Filter_Sanitization_String."</td>\n"; .
```

## REFERENCES

1. [https://www.w3schools.com/sql/sql\\_injection.asp](https://www.w3schools.com/sql/sql_injection.asp)
2. <https://www.hacksplaining.com/exercises/sql-injection#/hack-complete>
3. <https://portswigger.net/web-security/sql-injection/union-attacks>
4. <https://www.php.net/manual/en/security.database.sql-injection.php>
5. <https://www.w3resource.com/sql/sql-injection/sql-injection.phphttps://security.stackexchange.com/questions/196169/why-should-we-sometimes-use-instead-of-in-sql-injection-to-comment-the-re>
6. [https://www.google.com/search?q=how+union+query+works+sql+injection&sxsrf=ALiCzsZRSZMi9sf92dNJBj-jmENMbFwQRRQ%3A1651856865785&ei=4VV1Ysy6L6iX9u8P\\_6OS-AI&oq=how+union+query+works+sql+inje&gs\\_lcp=Cgdnd3Mtd2l6EAEYADIHCCEQChCgAToHCAAQRxCwAzoECCMQJzoFCCEQoAE6CAghEBYQHRAeSgQIQRgASgQIRhgAUKEGWNlqYPw1aARwAXgAgAGZAYgBvwuSAQM3LjeYAQCgAQHIAQjAAQE&sc\\_lent=gws-wiz](https://www.google.com/search?q=how+union+query+works+sql+injection&sxsrf=ALiCzsZRSZMi9sf92dNJBj-jmENMbFwQRRQ%3A1651856865785&ei=4VV1Ysy6L6iX9u8P_6OS-AI&oq=how+union+query+works+sql+inje&gs_lcp=Cgdnd3Mtd2l6EAEYADIHCCEQChCgAToHCAAQRxCwAzoECCMQJzoFCCEQoAE6CAghEBYQHRAeSgQIQRgASgQIRhgAUKEGWNlqYPw1aARwAXgAgAGZAYgBvwuSAQM3LjeYAQCgAQHIAQjAAQE&sc_lent=gws-wiz)
7. <https://www.hackedu.com/blog/how-to-prevent-sql-injection-vulnerabilities-how-prepared-statements-work>
8. [https://phpdelusions.net/pdo/sql\\_injection\\_example](https://phpdelusions.net/pdo/sql_injection_example)
9. <https://stackoverflow.com/questions/134099/are-pdo-prepared-statements-sufficient-to-prevent-sql-injection>
10. <https://websitebeaver.com/prepared-statements-in-php-mysqli-to-prevent-sql-injection>
11. [https://www.youtube.com/watch?v=l4JYwRljX6c&ab\\_channel=DaniKrossing](https://www.youtube.com/watch?v=l4JYwRljX6c&ab_channel=DaniKrossing)
12. [https://phpdelusions.net/mysqli\\_examples/prepared\\_select](https://phpdelusions.net/mysqli_examples/prepared_select)
13. [https://www.youtube.com/watch?v=5fnUt9fYQII&ab\\_channel=WebDevwithMatt](https://www.youtube.com/watch?v=5fnUt9fYQII&ab_channel=WebDevwithMatt)  
<https://portswigger.net/burp/documentation/desktop/tools/proxy/options>  
<https://medium.com/nerd-for-tech/how-to-manipulate-web-requests-ea5cf1c80a90>
14. [https://cheatsheetseries.owasp.org/cheatsheets/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html)
15. <https://www.wordfence.com/learn/how-to-prevent-cross-site-scripting-attacks/>  
[https://www.w3schools.com/php/filter\\_sanitize\\_string.asp](https://www.w3schools.com/php/filter_sanitize_string.asp)
16. <https://www.php.net/manual/en/filter.filters.validate.php>
17. [https://www.w3schools.com/php/php\\_form\\_validation.asp](https://www.w3schools.com/php/php_form_validation.asp)