ASSESSMENT 1B

TESTING REPORT

(FILE UPLOAD VULNERABILITIES)

Kieu Q T Nguyen 339142 0435293839

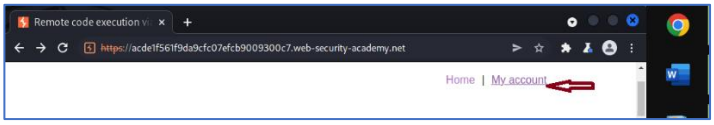
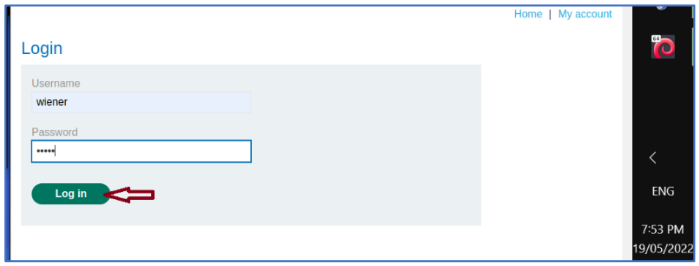
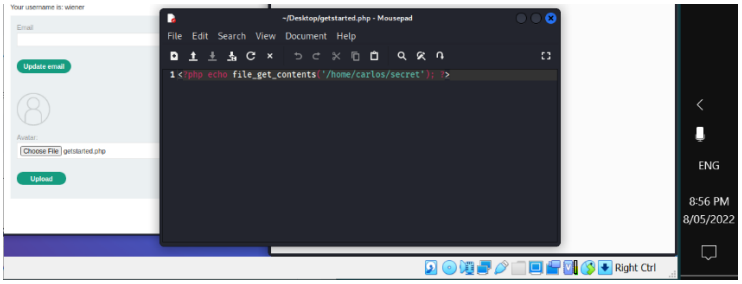
CHARLES DARWIN UNIVERSITY

COLLEGE OF ENGINEERING, INFORMATION TECHNOLOGY AND ENVIRONMENT

27 April 2021

Table of Contents

1. Remote code execution via web shell upload

Remote code execution via web shell upload	
Problem 1	This lab contains a vulnerable image upload function. It doesn't perform any validation on the files users upload before storing them on the server's filesystem. To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner. You can log in to your own account using the following credentials: **wiener:peter**
Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login.*  *Figure 1: Access login page by tapping on "My account"*  *Figure 2: Login using username: wiener and password: peter* *Step 2: Create and upload a malicious php file* After, create a malicious php file with content: `<?php echo file_get_contents('/home/carlos/secret'); ?>` The php command echo help print out the content of default webpage (can be index.php) in repository ('/home/carlos/secret"). Then We upload it in "My Account" page  *Figure 3: Upload malicious php file, getstarted.php* *Step 3: After uploading the malicious file (getstarted.php) and get notified uploading successfully.*

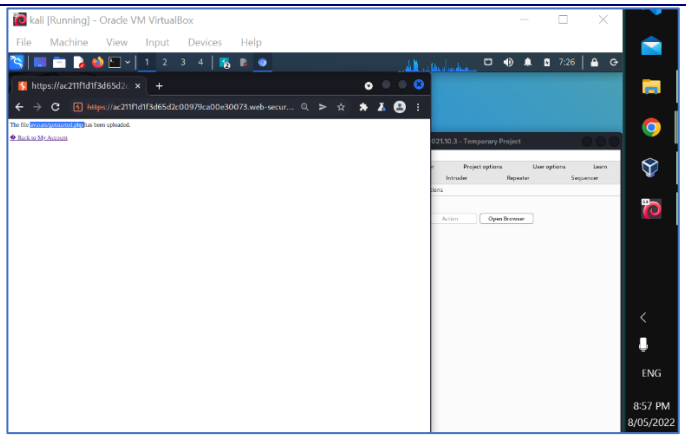


Figure 4: The file getstarted.php is uploaded

Step 4: we get back the "My account" page and refresh. Then looking for the file getstarted.php in HTTP history, we see the content of the php file with the code we need to solve this task and submit it.

The content of the file is the result of the "echo" command after executed by php server.

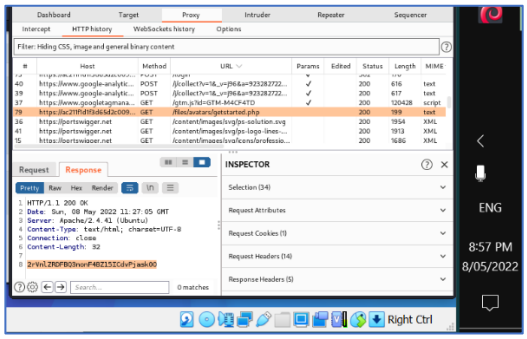


Figure 5: Content of getstarted.php with the code after executed by server.

2. Web shell upload via Content-Type restriction bypass

Web shell upload via Content-Type restriction bypass	
Problem 2	This lab contains a vulnerable image upload function. It attempts to prevent users from uploading unexpected file types,but relies on checking user-controllable input to verify this. To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner. You can log in to your own account using the following credentials: **wiener:peter**
Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login.* (Similar to first step of problem 1) *Step 2: Uploading the malicious php file (getstarted.php) and recognized that it is denied by php file.*

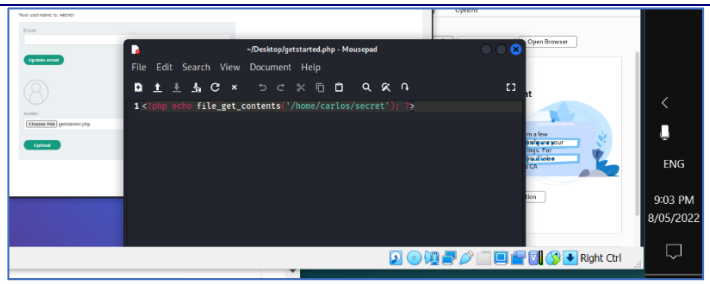


Figure 6: Uploading malicious php file

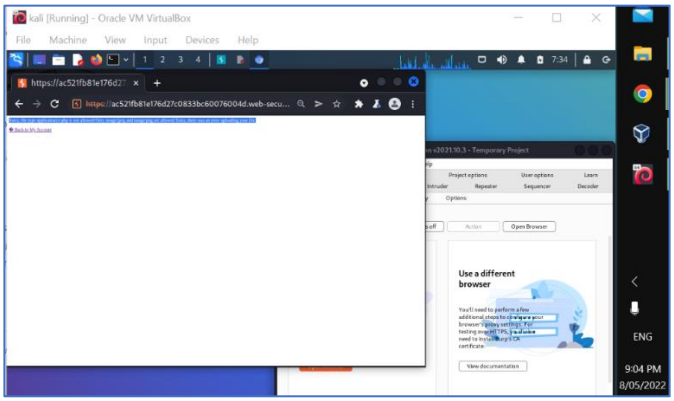


Figure 7: Notified the uploading file getstarted.php is denied because its type php.

Step 3: Upload the getstarted.php file again and using Burp Suit to edit the request upload. We change the Content-Type to image/jpeg and forward the request.

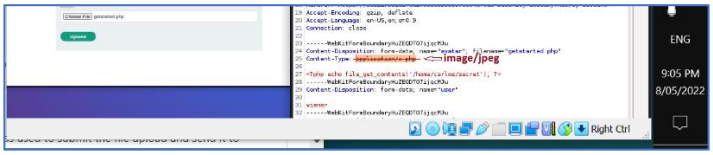


Figure 8: Using Burp Suit to edit Content-Type in the uploading request.

Step 3: After successfully forward the uploading request to server, we get back the "My account" page, refresh and could find the code in the content of file getstarted.php in HTTP History (can be seen as below). It helps solve the task.

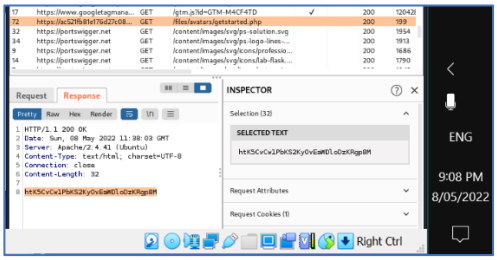
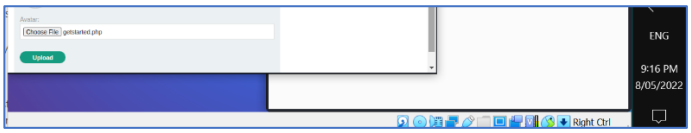
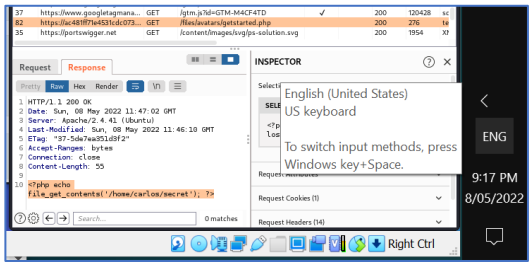
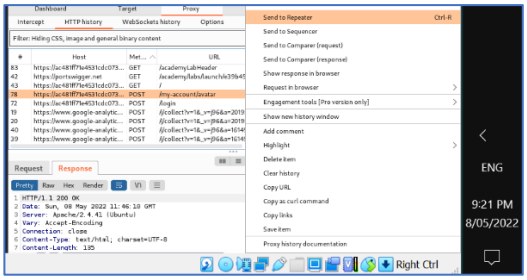


Figure 9: Content of getstarted.php with the code after executed by server.

3. Web shell upload via path traversal

Web shell upload via path traversal	
Problem	This lab contains a vulnerable image upload function. The server is configured to prevent execution of user-supplied files, but this restriction can be bypassed by exploiting a secondary vulnerability. To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner. You can log in to your own account using the following credentials: **wiener:peter**
Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login.* (Similar to first step of problem 1) *Step 2: Using the malicious php file (getstarted.php like in problem 1&2) to upload and get notification the upload succeeds.*  *Figure 10: Upload the malicious file getstarted.php* *Step 3: get back "My account" page and refresh. Then looking the request for file getstarted.php in HTTP history, we found that the content of file (php command) is not executed by server.*  *Figure 11: Content of getstarted.php is not executed by server.* *Step 4: Looking for the request of avatar (php file) and send it to repeater.*  *Figure 12: Content of getstarted.php is not executed by server.* *Step 5: We try to add path traversal to parent repository while uploading the file. To do this, in the Content-Disposition header, change the filename into:* *Content-Disposition: form-data; name="avatar"; filename="../exploit.php"* *After that, we send the request and be notified the file is uploaded in response (seen as below).*

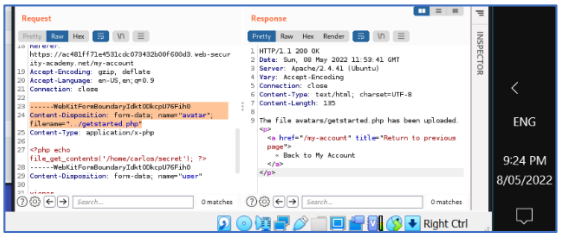


Figure 13: Adding traversal path, send the request and get response that file is uploaded.

Step 6: After getting back "My account" page and refresh, looking for the file getstarted.php in HTTP history, we get the code for solving the tasks in its content after it is executed.

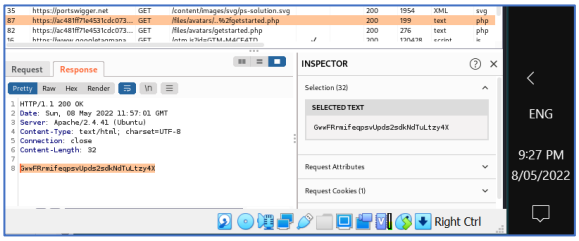
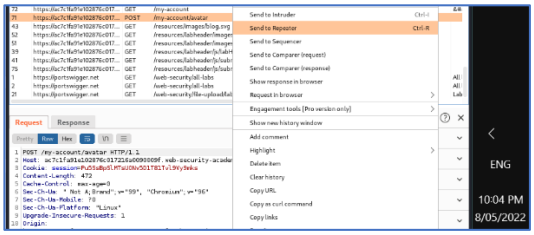


Figure 14: Content of uploaded php file after adding traversal path.

4. Web shell upload via extension blacklist bypass

Web shell upload via extension blacklist bypass	
Problem	This lab contains a vulnerable image upload function. Certain file extensions are blacklisted, but this defense can be bypassed due to a fundamental flaw in the configuration of this blacklist. To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner. You can log in to your own account using the following credentials: **wiener:peter**
Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login.* (Similar to first step of problem 1) *Step 2: Using the malicious php file (getstarted.php like in previous problem) to upload and get notification the upload failed.* *Step 3: upload a normal image file successfully in "My account" page, then looking for the image request in "My account" page in HTTP History. We sent it to repeater.*  *Figure 15: Send request to Repeater* *Step 4: Edit the image request of image to upload htaccess file, which is to config Apache server and allow server to execute the php under the extension .l33t* *To do this:*

- Change the filename to .htaccess.
- Change the Content-Type header to text/plain.
- Change the contents of the file to: AddType application/x-httpd-php .l33t

After making changes and send request, we recognize successfully uploaded in the response.

Figure 16: Send request to upload .htaccess config file

Step 5: Looking for request uploading getstarted.php in HTTP History, then using Repeater to edit the filename: getstarted.php to getstarted.l33t .Then send the request to server and be notified that the file is uploaded in response.

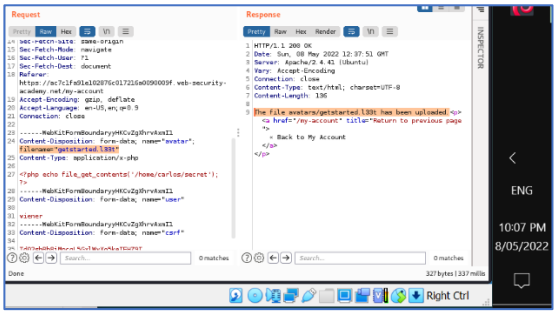


Figure 17: Request and response of uploading file getstarted.l33t

Step 6: Using repeater to edit the image request in "My account" page to get file: getstarted.l33t . Then we can get the content of the file after being executed by server. It also the code we need to solve this task.

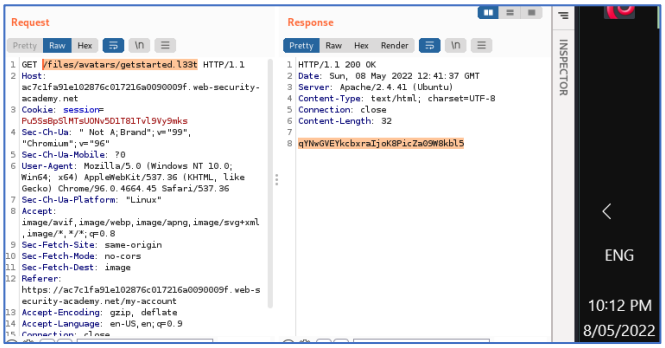
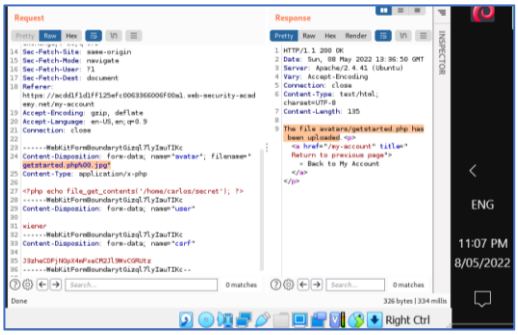
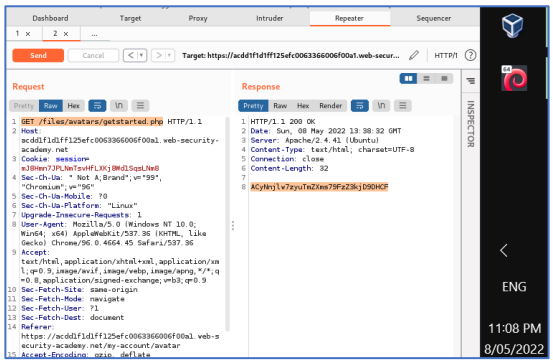


Figure 18: Content of file getstarted.l33t

5. Web shell upload via obfuscated file extension

Web shell upload via obfuscated file extension	
Problem	This lab contains a vulnerable image upload function. Certain file extensions are blacklisted, but this defense can be bypassed using a classic obfuscation technique. To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner.

	You can log in to your own account using the following credentials: **wiener:peter**
Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login* (like 1st step in problem 1) *Step 2: Using the malicious php file (getstarted.php like in previous problem) to upload and get notification the upload failed.* *Step 3: looking for the request of posting getstarted file (POST /my-account/avatar) in HTTP History. We sent it to repeater.* *Step 4: In repeater, we try editing filename: "getstarted.php" by "getstarted.php%00.jpg" . Then sending request and we got the uploading succeed in response.*  *Figure 19: Edit the request to upload the getstarted.php%00.jpg* *Step 5: After that, we send a request to get file uploaded file (GET /files/avatars/getstarted.php) in the repeater and we got the content with code to solve this task.*  *Figure 20: Response of request for getting content file getstarted.php*

6. Remote code execution via polyglot web shell upload

Remote code execution via polyglot web shell upload	
Problem	This lab contains a vulnerable image upload function. Although it checks the contents of the file to verify that it is a genuine image, it is still possible to upload and execute server-side code. To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner. You can log in to your own account using the following credentials: **wiener:peter**
Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login* (like 1st step in problem 1) *Step 2: Using the malicious php file (getstarted.php like in previous problem) to upload and get notification the upload failed.*

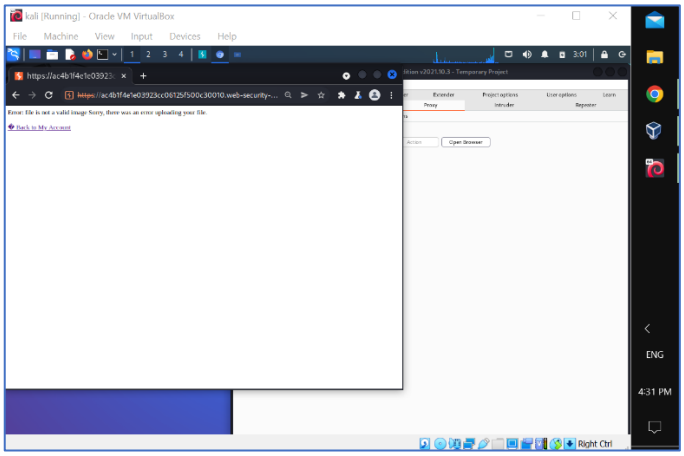


Figure 21: Failure to upload the getstarted.php

Step 3: Using the exiftool to create a polyglot PHP/JPG file which contains your PHP payload in its metadata.

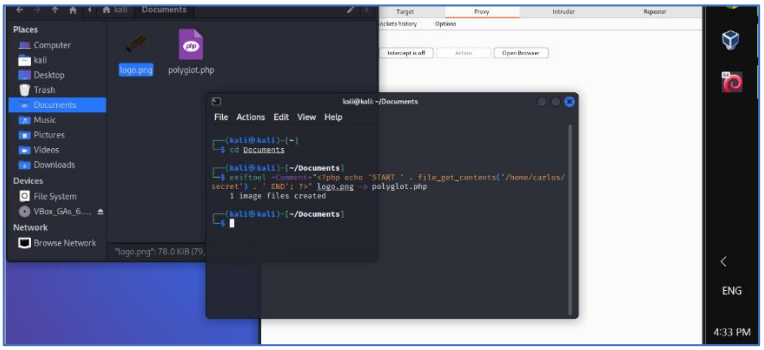


Figure 22: Create polyglot file with exiftool

Step 4: Upload the polyglot.php file in "My account" page and be notified the upload is succeeded.

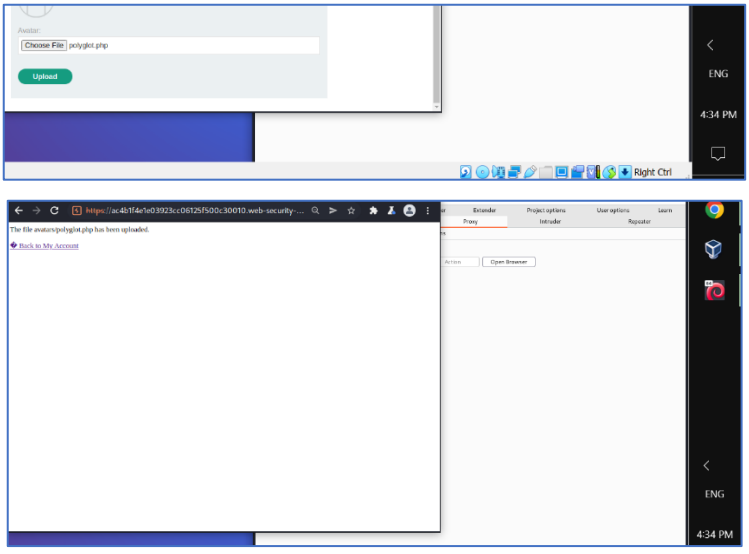


Figure 23: Succeed to upload the polyglot file polyglot.php

Step 4: After uploading successful, we open the file (seen as below)

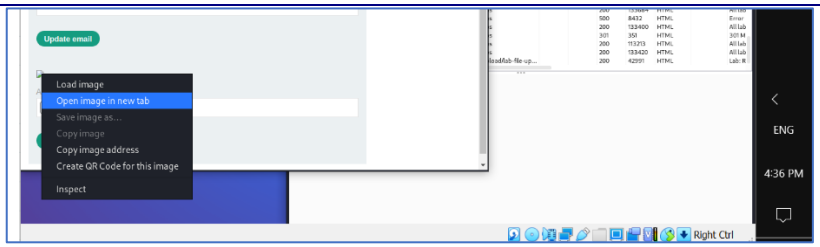


Figure 24: Open the polyglot file after upload

Step 5: The content of polyglot file, we count get the content file with the needed code between "START" and "END". The code is to solve this task.

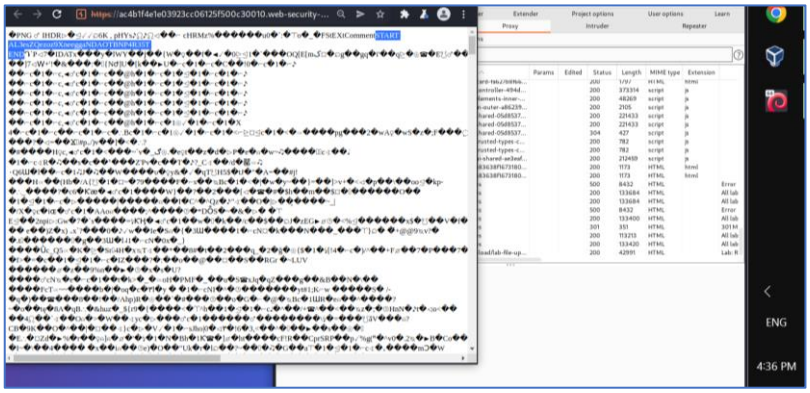
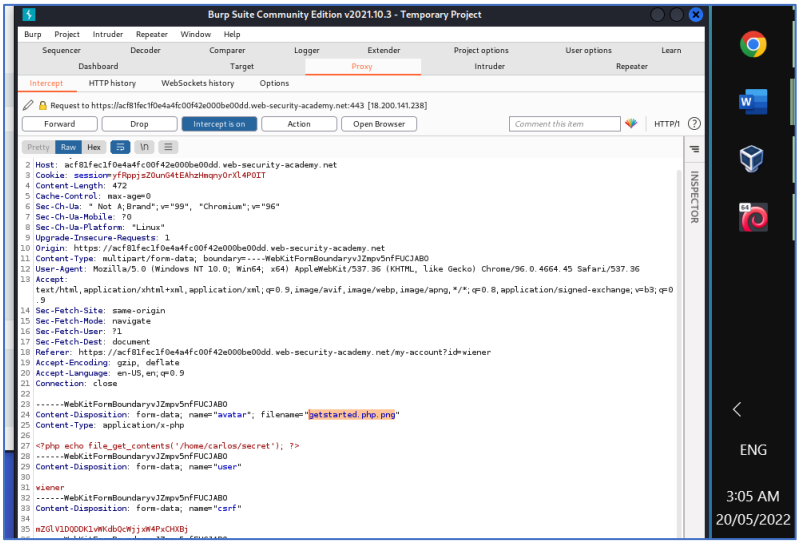
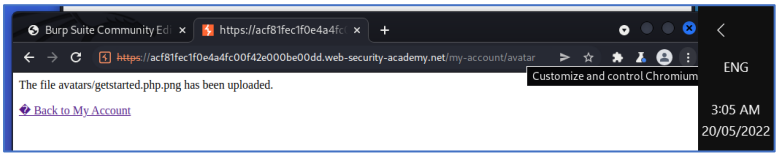
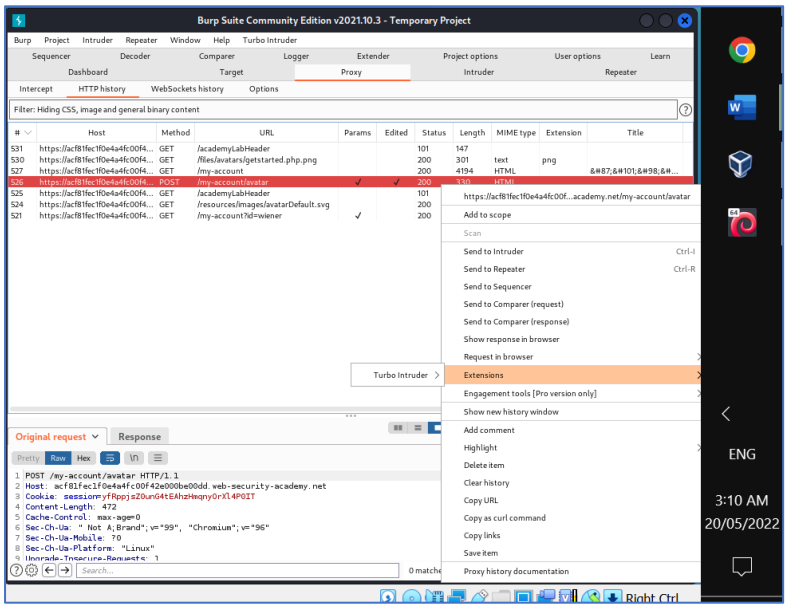


Figure 25: Content of polyglot.php after uploaded

7. Web shell upload via race condition

Web shell upload via race condition	
Problem	This lab contains a vulnerable image upload function. Although it performs robust validation on any files that are uploaded, it is possible to bypass this validation entirely by exploiting a race condition in the way it processes them. To solve the lab, upload a basic PHP web shell, then use it to exfiltrate the contents of the file **/home/carlos/secret**. Submit this secret using the button provided in the lab banner. You can log in to your own account using the following credentials: **wiener:peter**

Reproduce	*Step 1: Launch Burpsuit's browser and using credentials **wiener:peter** to login* (like 1st step in problem 1) *Step 2: Using the malicious php file (getstarted.php like in previous problem) to upload and get notification the upload failed.* *Step 3: Upload getstarted.php again and using the Burp Suit to edit the filename: getstarted.php to getstarted.php.png. Then we get notified that uploading file succeed (seen as below)*  *Figure 26: Edit the request to upload the getstarted.php*  *Figure 27: Upload the file succeed.* *Step 4: Send the request to post the file getstarted.php.png (POST /my-account/avatar) to turbo intruder*  *Figure 28: Send POST request to Turbo Intruder*

Step 5: In turbo Intruder, using the pattern below:

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint, concurrentConnections=10,)
    request1 = '''<YOUR-POST-REQUEST>'''
    request2 = '''<YOUR-GET-REQUEST>'''
    engine.queue(request1, gate='race1')
    for x in range(5):
        engine.queue(request2, gate='race1')
    engine.openGate('race1')
    engine.complete(timeout=60)
def handleResponse(req, interesting):
    table.add(req)
```

After that, replacing request 1 by POST request post getstarted.php and replacing request 2 by GET request to get the content of getstarted.php (seen as below)

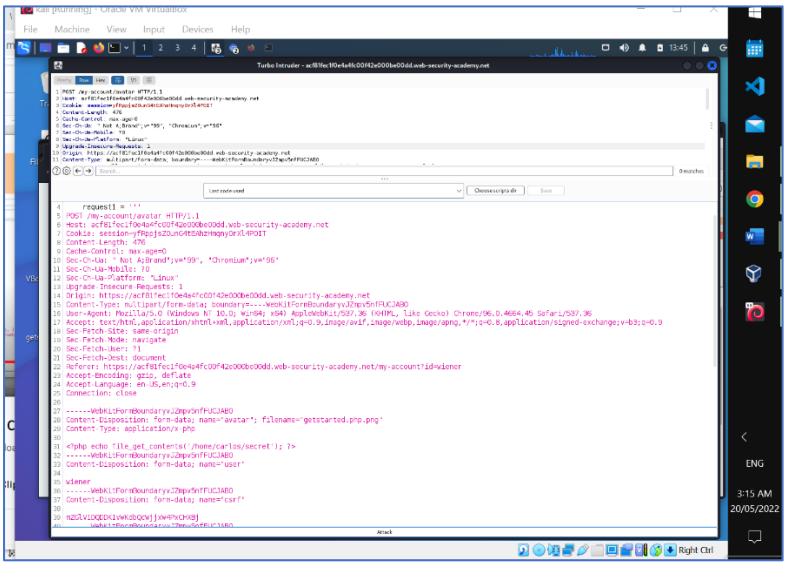


Figure 29: Request1 in pattern of Turbo Intruder

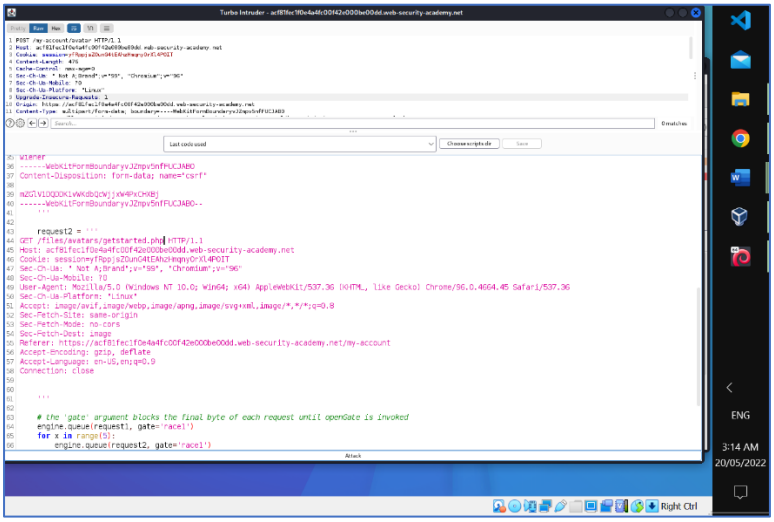
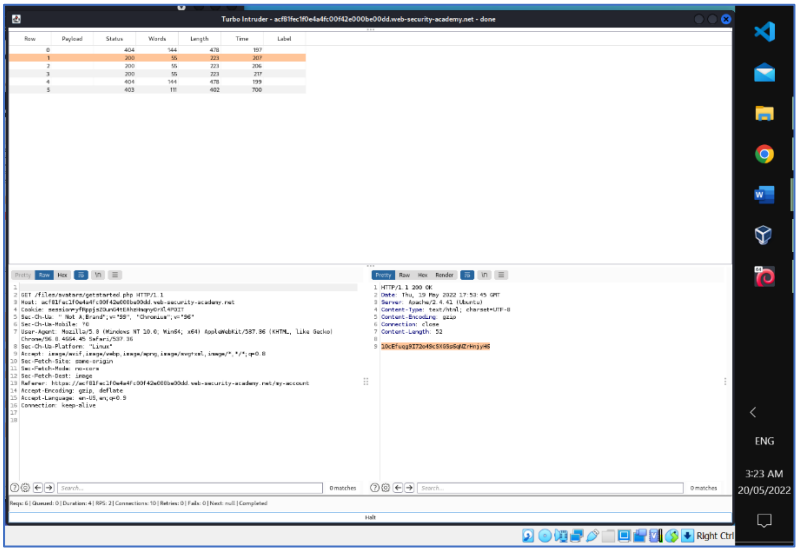


Figure 30: Request2 in pattern of Turbo Intruder

	Step 6: after we activated the attack button, there will some results. Success results with code 200 will return us a code to solve this task (as below)  *Figure 31: Results of the attacks*

8. Dashboard of completing all labs of “File upload vulnerabilities” Topic

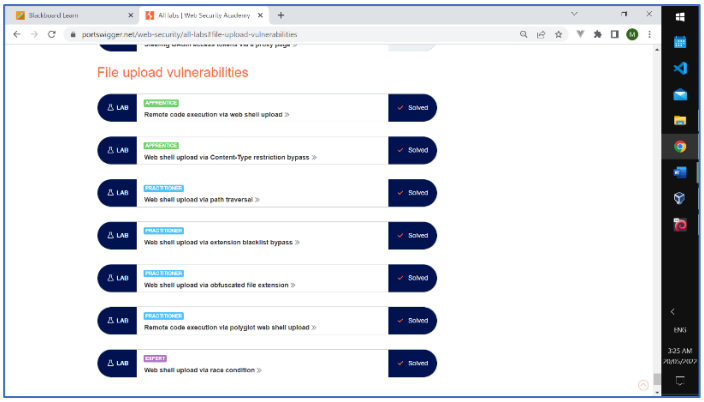


Figure 32: Dashbboad of all tasks

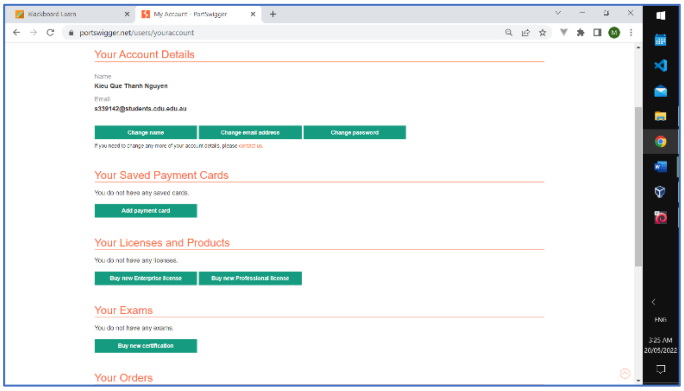


Figure 31: Account used in Portswigger