3A DoS/burst protection using Nginx Proxy

1. Screenshot of your nginx configuration file with the newly added items (1 point)

```
worker_processes  1;  # Define the number of worker processes

events {
    worker_connections  1024;  # Set the maximum number of simultaneous connections
}

http {
    limit_req_zone $binary_remote_addr zone=myLimit:1m rate=10r/s;  # Rate limiting: 10 requests
    server {
        listen 80;  # Listen on port 80 for HTTP traffic
        server_name localhost;

        location / {
            limit_req zone=myLimit burst=1 delay=2;

            proxy_pass http://localhost:3009;  # Forward requests to port 3009
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
```

2. Outputs after running the script. (2 points)

Request 1 : HTTP status 200 Request 2 :
HTTP status 200
Request 3 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ... $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+ ~~~~~~
  + CategoryInfo : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
 WebRequest], WebException
  + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
 RequestCommand
Request 4 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3

```
At line:2 char:16
+ ...  $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
   WebRequest], WebException
    + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
   RequestCommand
Request 5 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ...  $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
   WebRequest], WebException
    + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
   RequestCommand
Request 6 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ...  $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
   WebRequest], WebException
    + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
   RequestCommand
Request 7 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
```

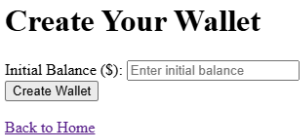
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ... $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+   ~~~~~~
+ CategoryInfo   : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest)
[Invoke-   WebRequest], WebException
+ FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
  RequestCommand
Request 8 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ... $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+   ~~~~~~
   + CategoryInfo   : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
  WebRequest], WebException
   + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
  RequestCommand
Request 9 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ... $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+   ~~~~~~
   + CategoryInfo   : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
  WebRequest], WebException
   + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
  RequestCommand
Request 10 : HTTP status 200
Request 11 : HTTP status 200

```
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ...  $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
   WebRequest], WebException
    + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
   RequestCommand
Request 12 : HTTP status 200
Invoke-WebRequest :
503 Service Temporarily Unavailable
503 Service Temporarily Unavailable
nginx/1.29.3
At line:2 char:16
+ ...  $status = (Invoke-WebRequest -Uri "http://localhost" -Method GET -Us ...
+                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation:
(System.Net.HttpWebRequest:HttpWebRequest) [Invoke-
   WebRequest], WebException
    + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWeb
   RequestCommand
```

3. Did you see any evidence of rate-limiting? (1 point)
   Yes, I did. The evidence of rate limiting that I see is the 503 responses from the server saying the service is temporarily unavailable
4. What is the HTTP status code when you see the rate limiting? (1 point)
   The status code that I see is 503
5. How can you rate limits for a specific IP using Nginx? Write in 3 lines. (3 points) (you don't need to test this)
   limit_req_zone $binary_remote_addr zone=myLimit:1m rate=10r/s;
   limit_req zone=myLimit burst=1 delay=2;
   This will ensure nobody tries to crash our app with a lot of requests.

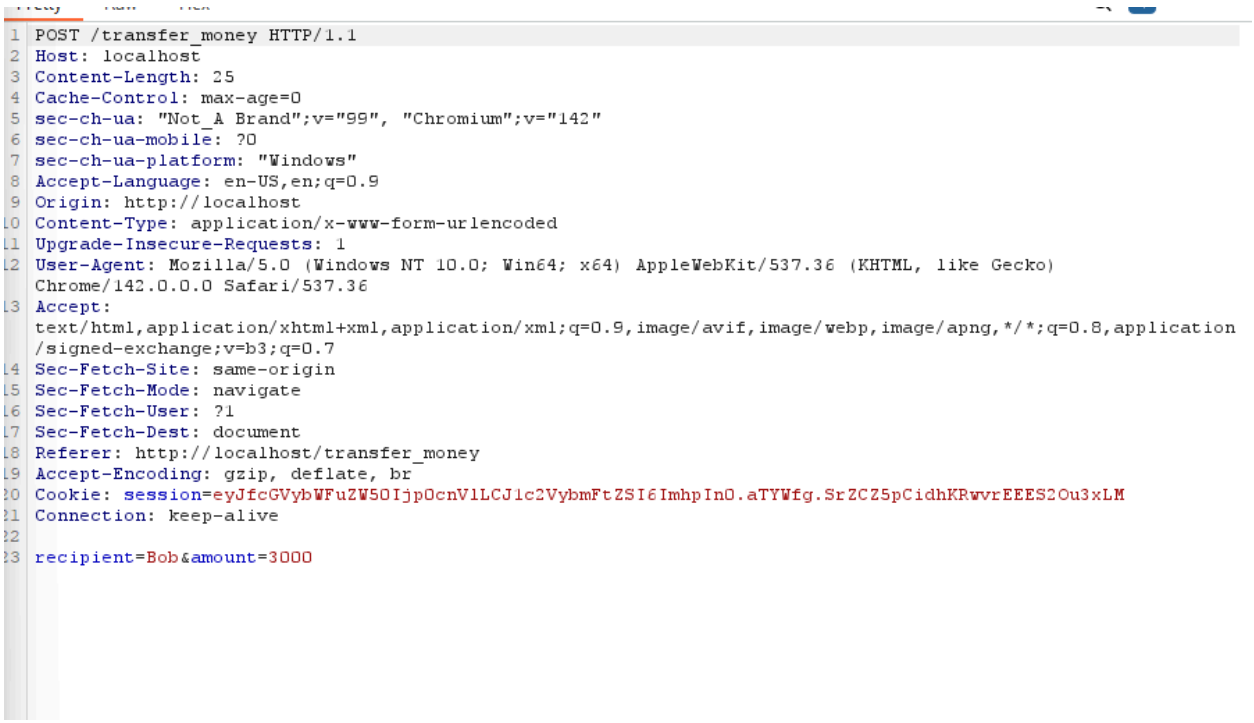
Part 3B: Developing a secured personal wallet system (32 points)

1. Download the project-3 template directory. The template contains incomplete code components that need to be filled. There are four missing sections in the app.py web server code. Your task is to complete these missing parts and submit the updated project_3 files with your modifications, along with a detailed report (refer to the submission instructions). **(5 points)**
2. Login to the system using your registered username and password.
3. After successfully logging in, you should be able to access the wallet functionalities. Capture a screenshot of this page and include it in your project report **(1 point)**.



4. Create two user profiles and add different amounts of money to the wallets of each profile. You can choose any amount you like since this is our dream wallet.
5. Install Burp tool into your system. We will use Burp to find out possible vulnerabilities.
6. **Finding vulnerabilities - check for CSRF token:**

a. Open the Burp Suite tool that you have already installed on your system. Launch the Burp browser and log in to your wallet application using the Burp browser. Attempt to transfer some money between two registered profiles.
b. Using Burp, inspect the request payload and headers to determine if a CSRF protection token is present (e.g., a hidden form field or a token in the headers). Capture a screenshot **(1 point)**. Do you notice any mechanism for CSRF protection? Provide your observation **(1 point)**?

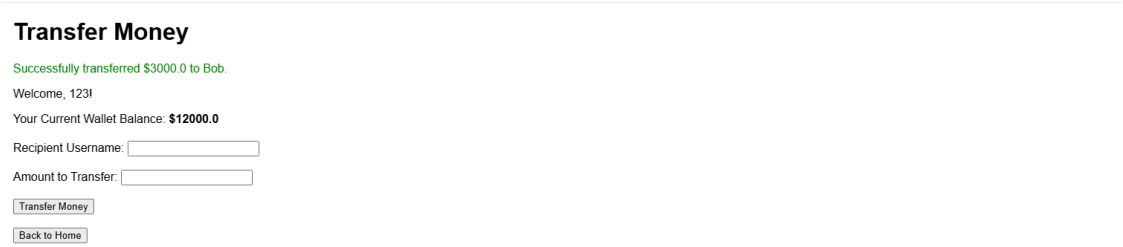


```
POST /transfer_money HTTP/1.1
Host: localhost
Content-Length: 25
Cache-Control: max-age=0
sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/142.0.0.0 Safari/537.36
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
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/transfer_money
Accept-Encoding: gzip, deflate, br
Cookie: session=eyJfcGVybWFuZW50IjpOcnVlLCJ1c2VybmFtZSI6ImhpIn0.aTYWfg.SrZCZ5pCidhKRwvrEEES2Ou3xLM
Connection: keep-alive

recipient=Bob&amount=3000
```

There is no section for CSRF protection. So, with that observation, we can make a CSRF attack

7. **Write an attack:** If there is no CSRF token present, the endpoints might be vulnerable to CSRF attack. Your next job is to write some code for a CSRF attack.
   Hint: As part of this task, you are required to write an HTML file to simulate a Cross-Site Request Forgery (CSRF) attack. This can be achieved using the POST method through an input submission form. First, identify the transfer money URL by observing a previous successful transaction, and then use this URL in your attack HTML file.
   Choose any registered users to act as the victim and recipient for this transaction. Create a separate file named attack.html and include it in your project submission, along with the project report **(3 points)**. To execute the simulated attack, ensure the victim is logged into the web application. Then, open your attack HTML file in a browser to simulate the attack. While this is for educational purposes, in a real-world scenario, an attacker might trick the victim into clicking a malicious link or opening a malicious file that executes the attack.
8. **Showing the success of your attack:** Capture a screenshot that will show the success of your attack. Add that screenshot to your report **(1 point)**.



9. Wait until your login cookie expires. Now, try to load the same attack.html file in your browser. Is your attack working **(1 point)**? Why **(1 point)**?
   It fails because the victim is logged out. The victim must be logged in for the attack to be successful.
10. **Defending against CSRF attack:** To defend against CSRF attacks, we will introduce a CSRF token in our POST requests. For Python Flask, we can utilize the flask_wtf.csrf library. You need to install this library using either pip3 or pip. Your task is to implement the flask_wtf.csrf library to protect the application from CSRF attacks. This will involve making changes across multiple files and including a hidden field in forms to pass the CSRF token **(4 points)**. You should submit the whole modified code base as a submission of this project.
11. **Checking CSRF token in Burp:** Now, open Burp, and from Burp browser submit some form. Check whether you can see any CSRF token in the POST request. Capture a screenshot **(1 point)**.

Request

Pretty Raw Hex

```
1  POST /transfer_money HTTP/1.1
2  Host: localhost
3  Content-Length: 129
4  Cache-Control: max-age=0
5  sec-ch-ua: "Not_A Brand";v="99", "Chromium";v="142"
6  sec-ch-ua-mobile: ?0
7  sec-ch-ua-platform: "Windows"
8  Accept-Language: en-US,en;q=0.9
9  Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 Upgrade-Insecure-Requests: 1
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/142.0.0.0 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: http://localhost/transfer_money
19 Accept-Encoding: gzip, deflate, br
20 Cookie: session=
   .eJwFwUEOgCAMBMC_7NkLghT4DC1QojGiQTwZ_-7Mi3hJP7hJGwijPzIh373Gce7SEJBV8pQckXKWSiqSndMyV8XWW21IM5dsFo8Jzy298SE
   IWDd8P2B8HKQ.aTZS1g.BzqsmjI58mYthqOSUATi28SZ6HE
21 Connection: keep-alive
22
23 csrf_token=ImMxYjk3Yjg3NzE4NjdkYmRlYzg4M2UyZjFhNjk2MzQ3M2FhZGMONTki.aTZS1g.JCr6RSKC1zzNfoMYuMDw9Ud723U&
   recipient=Bob&amount=10000
```

12. Reflection questions:
    a. The current code is not yet ready for production deployment. One of the significant shortcomings is the lack of comprehensive input validation throughout the application. What are the input validations that are already done **(2 points)**? What are the input validations that you can do more **(2 points)**? Modify your code accordingly **(2 points)**.
    So we have already added checks for negative numbers, checks for the balance being a number, and also checks for a user's wallet actually existing. However, we could also add amount formatting and reject balances that are expressed in scientific notation.
    b. How can you strengthen your session security more against CSRF attacks **(1 point)**? Where do you need to make the change in code—just mention where do you need to make the change **(1 point)**?
    You could make the session cookie harder to steal and this would go right after you create the Flask app. The way you make it harder to steal is you dont send the actual cookie for every post request you make. You put a random word in there, like "valid" instead of the real cookie.
    c. How did your implementation ensure protection against SQL injection attacks **(2 points)**? Can you come up with an SQL injection attempt, and check whether your implementation protects against SQL injection attempts **(2 points)**? Just write your attack attempt and show whether it was successful or unsuccessful.
    So my implementation uses a dictionary. So if I type apple for username, it will be

{“Username”: apple}. Here apple is a string. So this means lets say i do {“Username”: 1=1}, this will be false cus 1=1 does not match any username in my database. The input is string and it does not directly execute anything. So because of this, SQLi fails.

You see it's a string, it's not an executable statement, so it does no harm.

d. Open ended question: What additional measures can you implement to enhance the usability of your code or strengthen the security features of your system? **(1 point)**? Mention just one improvement.

I can add encryption and hashing to provide confidentiality and integrity for the user. Furthermore, I can encrypt and hash the user's username and password and then have MongoDB decrypt it and store the hashed value in its database. It can then add a salt value so attackers can use a rainbow table to brute force their way into my application.