Exploring and Exploiting a Race Condition Vulnerability

Step 1: Launch the Environment

1. **Start the Machine**: Click the **Start Machine** button on the right to launch the attached VM.
2. **Start the AttackBox**: Click the **Start AttackBox** button at the top.
3. **Access the Target Application**: Using the AttackBox, navigate to `http://MACHINE_IP:8080`.

Step 2: Credentials for Login The following credentials are available for testing:

- **User 1**:
  - Mobile: `07799991337`
  - Password: `pass1234`
- **User 2**:
  - Mobile: `07113371111`
  - Password: `pass1234`

Understanding the Target Application

The application is a credit transfer portal belonging to a mobile operator. The objective is to check for a race condition vulnerability by attempting to transfer more credit than the account holds.

1. **Log In**: Use either set of credentials to log in.
2. **Initiate Credit Transfer**:
   - Navigate to the **Pay & Recharge** section and click **Transfer**.
   - Input the recipient's phone number and the transfer amount.
   - Experiment with both valid amounts (within the account balance) and invalid amounts (exceeding the balance).

Analyzing HTTP Requests Using Burp Suite

1. Open Burp Suite and set up a proxy:
   - Go to the **Intercept** tab and click **Open browser**.
   - If you encounter a sandbox error, navigate to **Settings** > **Burp's Browser** and enable the option to run without a sandbox.
2. Study HTTP requests:
   - Use Burp's browser to interact with the application.
   - Observe **POST** requests under the **HTTP history** tab.

Exploiting the Race Condition

Step 1: Send Request to Repeater

1. Right-click on a **POST** request in the **HTTP history** and select **Send to Repeater**.

Step 2: Duplicate Requests

1. Navigate to the **Repeater** tab.
2. Create a **tab group**:
   - Click the + icon near the request tab and select **Create tab group**.
   - Assign a group name and include the current request.
3. Duplicate the request:
   - Right-click the tab and choose **Duplicate tab** (or press `CTRL+R` multiple times).
   - Duplicate it 20 times for testing purposes.

Step 3: Send Requests

Burp Suite Repeater provides several methods to send requests:

Option 1: Send Group in Sequence

- **Single Connection**: Sends all requests in sequence over one connection, useful for detecting client-side desync issues.
- **Separate Connections**: Establishes and closes a new TCP connection for each request.

Results:

- Out of 21 requests, **five succeeded** and **sixteen were denied**. Successful requests took ~3 seconds each, while denied requests took ~4 ms.

Option 2: Send Group in Parallel

- Sends all requests simultaneously.

Results:

- All 21 requests succeeded, transferring credit.
- Each request completed in ~3.2 seconds.

Wireshark Observations

1. **Sequence Sending**: Each request is sent as a single packet.
2. **Parallel Sending**:
   - Requests are synchronized to arrive simultaneously.
   - With HTTP/1, synchronization uses **last-byte withholding**, delaying the final byte of each request until all are ready.

By carefully analyzing and exploiting the system's handling of parallel requests, we successfully transferred credit multiple times, demonstrating the presence of a race condition vulnerability.

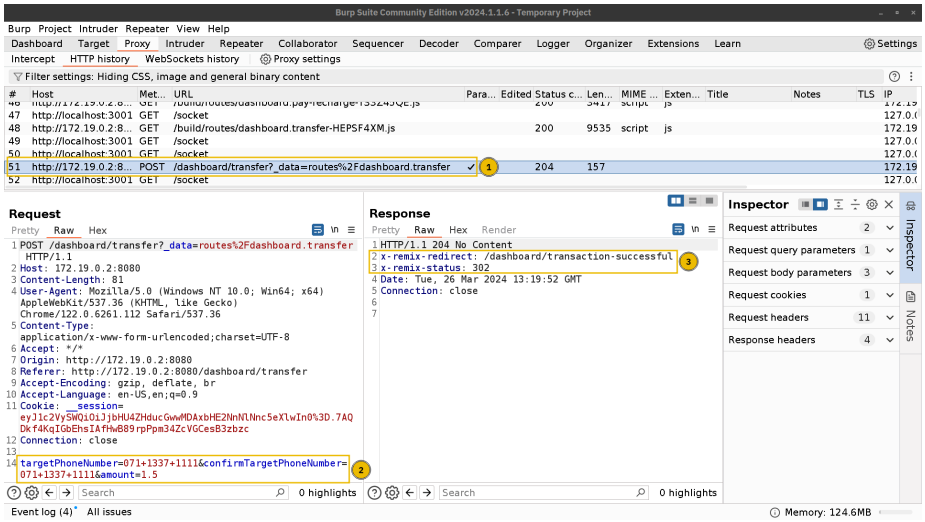

Figure 1: Burp Sutie Proxy HTTP history showing an HTTP POST request and response.

Now that we have seen how the system reacts to valid and invalid requests, let's see if we can exploit a race condition. Right-click on the `POST` request you want to duplicate and choose **Send to Repeater**.

In the **Repeater** tab, as shown in the numbered screenshots below:

1. Click on the **+** icon next to the received request tab and select **Create tab group**
2. Assign a group name, and include the tab of the request you just sent to the importer before clicking **Create**
3. Right-click on the request tab and choose **Duplicate tab** (If this option is not available in your version, you can press **CTRL**+**R** multiple times instead)
4. As a starting point, we will duplicate it 20 times

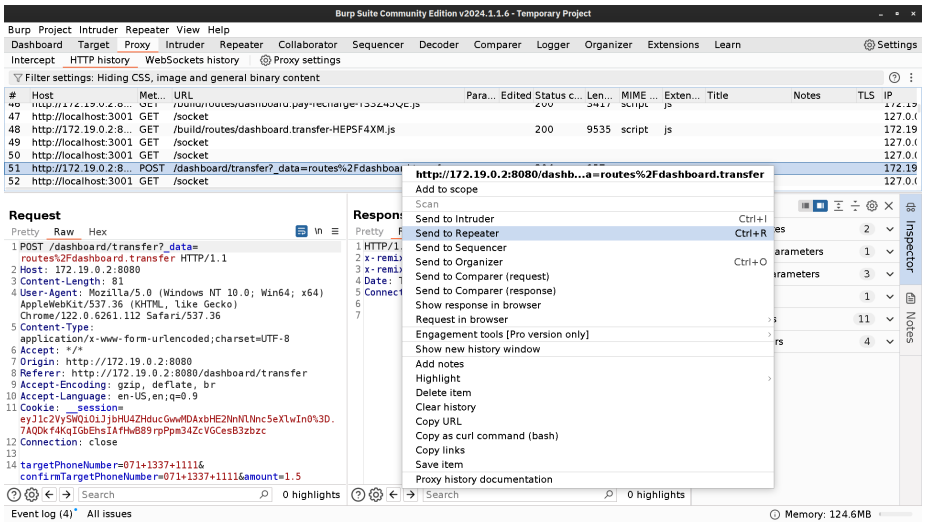

Figure 2: Burp Sutie Proxy HTTP history showing an HTTP POST request being sent to Repeater.

5. Next to the Send button, the arrow pointed downwards will bring a menu to decide how you want to send the duplicated requests

Next, we will exploit the target application by sending the duplicated request. Using the built-in options in Burp Suite Repeater, the drop-down arrow offers the following choices:

- Send group in sequence (**single connection**)
- Send group in sequence (**separate connections**)
- Send group in parallel

Sending Request Group in Sequence

Sending the group in sequence provides two options:

- Send group in sequence (**single connection**)
- Send group in sequence (**separate connections**)

Send Group in Sequence over a Single Connection

This option establishes a single connection to the server and sends all the requests in the group's tabs before closing the connection. This can be useful for testing for potential client-side desync vulnerabilities.

Send Group in Sequence over Separate Connections

As the name suggests, this option establishes a TCP connection, sends a request from the group, and closes the TCP connection before repeating the process for

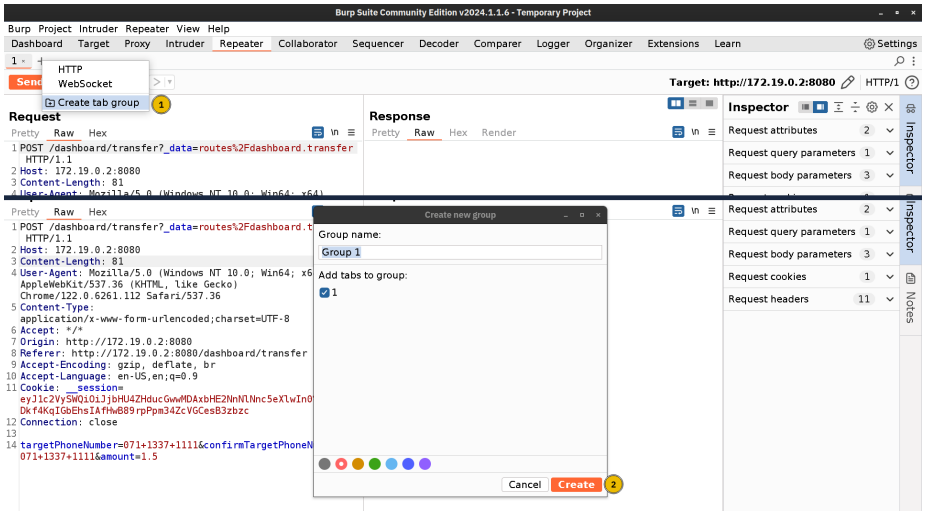

Figure 3: Creating a tab group in Burp Suite Repeater.

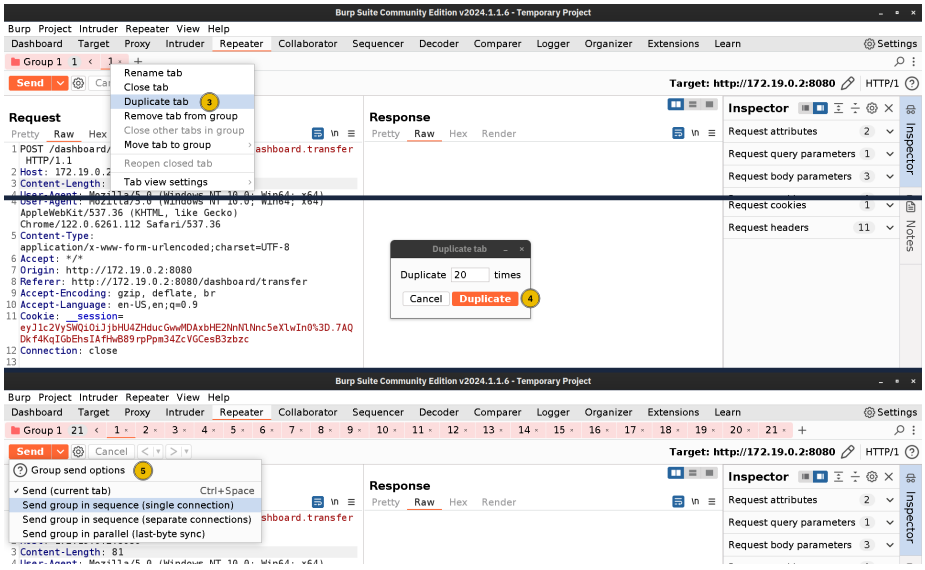

Figure 4: Duplicating a tab 20 times within a tab group in Burp Suite Repeater.

the subsequent request.

We tested this option to attack the web application. The screenshot below shows 21 TCP connections for the different POST requests in the group we sent.

- The first group (labelled 1) comprises five successful requests. We could confirm that they were successful by checking the respective responses. Furthermore, we noticed that each took around 3 seconds, as indicated by the duration (labelled 3).
- The second group (labelled 2) shows sixteen denied requests. The duration was around four milliseconds. It is interesting to check the Relative Start time as well.

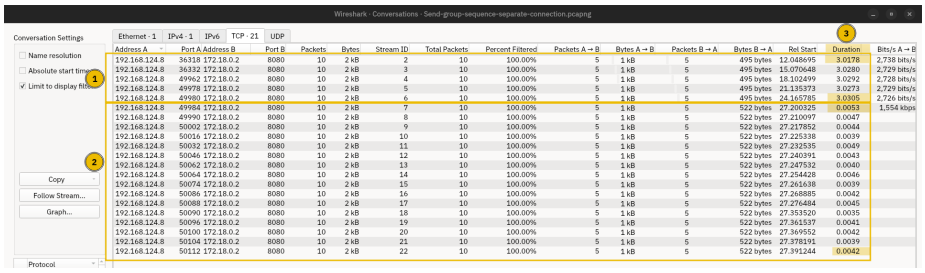

Figure 5: Wireshark showing 21 different POST requests; five are successful and the remaining ones are not successful.

The screenshot below shows the whole TCP connection for a request. We can confirm that the `POST` request was sent in a single packet.

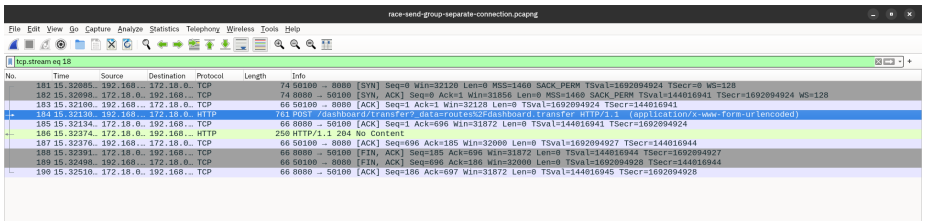

Figure 6: Wireshark showing the TCP connection related to a POST request.

Send Request Group in Parallel

Choosing to send the group's requests in parallel would trigger the Repeater to send all the requests in the group at once. In this case, we notice the following, as shown in the screenshot below:

- In the Relative Start column, we notice that all 21 packets were sent within a window of 0.5 milliseconds (labelled 1).

- All 21 requests were successful; they resulted in a successful credit transfer. Each request took around 3.2 seconds to complete (labelled 2).

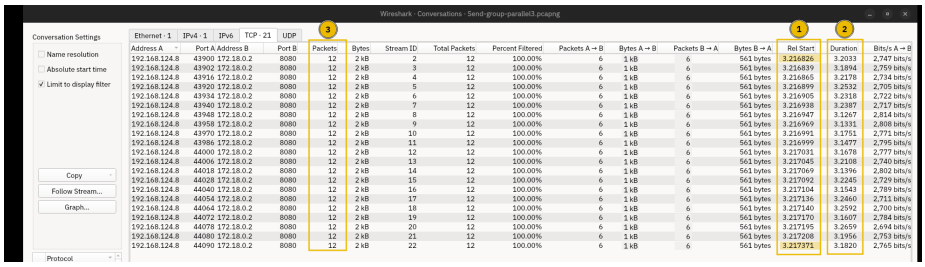

Figure 7: Wireshark showing 21 requests sent in parallel and at the same time.

By paying close attention to the screenshot above, we notice that each request led to 12 packets; however, in the previous attempt (send in sequence), we see that each request required only 10 packets. Why did this happen?

According to Sending Grouped HTTP Requests documentation, when sending in parallel, Repeater implements different techniques to synchronize the requests' arrival at the target, i.e., they arrive within a short time frame. The synchronization technique depends on the HTTP protocol being used:

- In the case of HTTP/2+, the Repeater tries to send the whole group in a single packet. In other words, a single TCP packet would carry multiple requests.
- In the case of HTTP/1, the Repeater resorts to last-byte synchronization. This trick is achieved by withholding the last byte from each request. Only once all packets are sent without the last-byte are the last-byte of all the requests sent. The screenshot below shows our `POST` request sent over two packets.

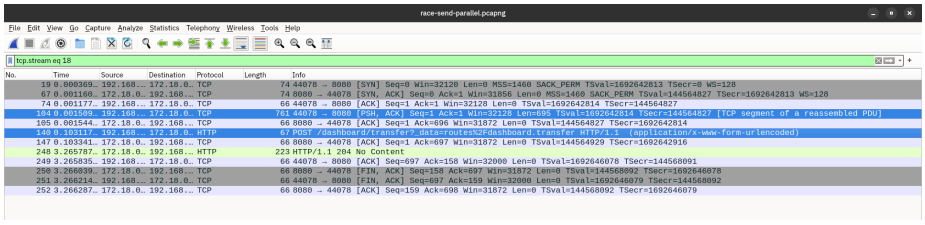

Figure 8: Wireshark showing the TCP connection related to a POST request when using last-byte synchronization.