SCENARIO

This lab's purchasing flow contains a race condition that enables us to purchase items for an unintended price. We will try to purchase a product without appropriate balance in our account.

# PROCEDURE

1. Open the application and inspect it properly to look for possibilities of any potential collision and we see that there exists a coupon code functionality.
2. Now inspect the coupon redeem request in BurpSuite’s Proxy and send it to Repeater 20 around times.
3. Now click on the **+** sign after the tabs and click create group, add all tabs to that group.
4. Now from Send’s dropdown menu, select **Send group(parallel)** and click it.
5. Now we can see that the single use coupon is applied more than 20 times.

**PAYLOAD**

**PROOF OF CONCEPT**

**REMEDIATION**

1. **Atomic Transactions:** The process of checking the validity of a coupon and its redemption should be atomic. This means the entire operation, from check to redemption, should complete fully or not at all. Databases like SQL support atomic transactions that can be leveraged to ensure the consistency of operations.
2. **Rate Limiting:** Implement rate limiting on sensitive endpoints, such as the one used for coupon redemption. This limits the number of times an endpoint can be hit within a specified time window, making it much harder for attackers to exploit race conditions.
3. **Mutex (Mutual Exclusion):** Mutexes can ensure that only one thread or process is allowed to execute the critical section (code that accesses shared resources) at a time. By locking around the critical section, other threads are blocked until the lock is released.
4. **Queueing:** Instead of processing requests as they come in, place them in a queue and process them sequentially. This ensures that only one request is handled at a time, removing the possibility of a race condition.
5. **Optimistic Concurrency Control (OCC):** OCC is a concurrency control method applied to transactional systems such as relational databases. Before committing a transaction, the system checks whether other transactions have modified the data. If there is a conflict, the committing transaction is rolled back.
6. **Avoid Client-Side Checks:** Don't solely rely on client-side validations or checks. Always ensure critical operations and checks are performed on the server-side.