Race Condition

Overview

Room URL: https://tryhackme.com/room/race-conditions-aoc2025-d7f0g3h6j9

Difficulty: Easy

Category: Race Conditions

Date Completed: 12/20/2025

Objectives

- Understand what race conditions are and how they can affect web applications.
- Learn how to identify and exploit race conditions in web requests.
- How concurrent requests can manipulate stock or transaction values.
- Explore simple mitigation techniques to prevent race condition vulnerabilities.

Table of Contents

Introduction

This challenge demonstrates a critical race condition vulnerability in a web application's checkout process. The TBFC (The Best Festival Company) shopping cart application features a limited-edition SleighToy with only 10 units available. Through careful timing manipulation using Burp Suite's parallel request feature, it's possible to bypass inventory checks and purchase more items than are actually in stock, causing the inventory to go negative.

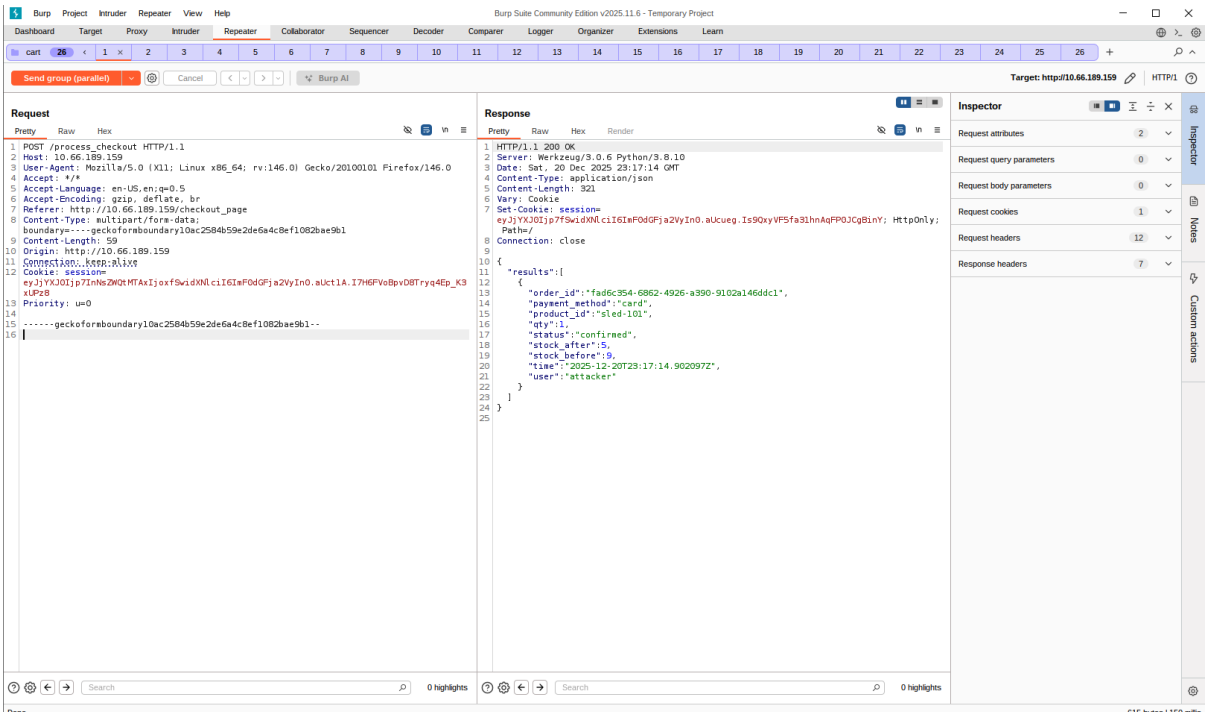
Key Information

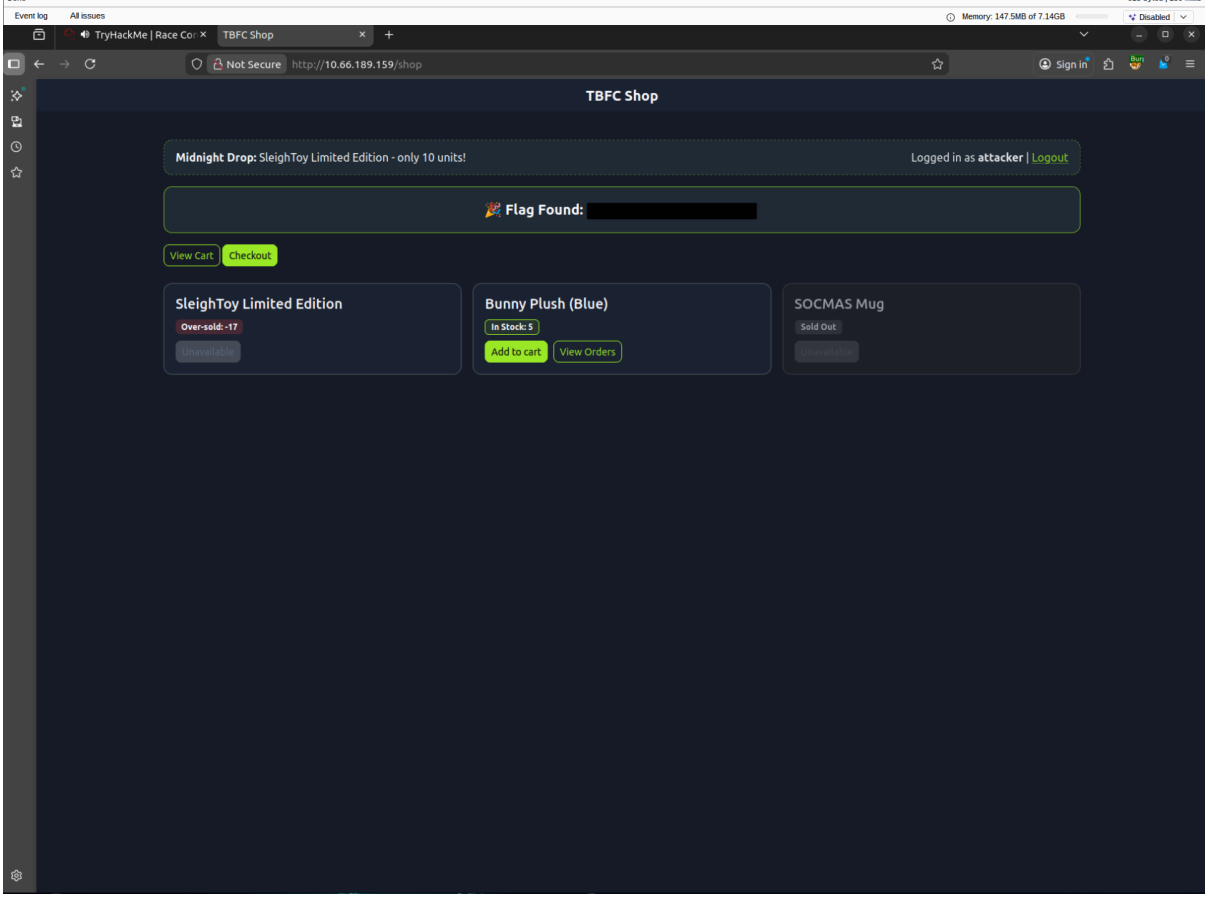
This challenge demonstrates a critical race condition vulnerability in a web application's checkout process. The TBFC (The Best Festival Company) shopping cart application features a limited-edition SleighToy with only 10 units available. Through careful timing manipulation using Burp Suite's parallel request feature, it's possible to bypass inventory checks and purchase more items than are actually in stock, causing the inventory to go negative.

Walk Through

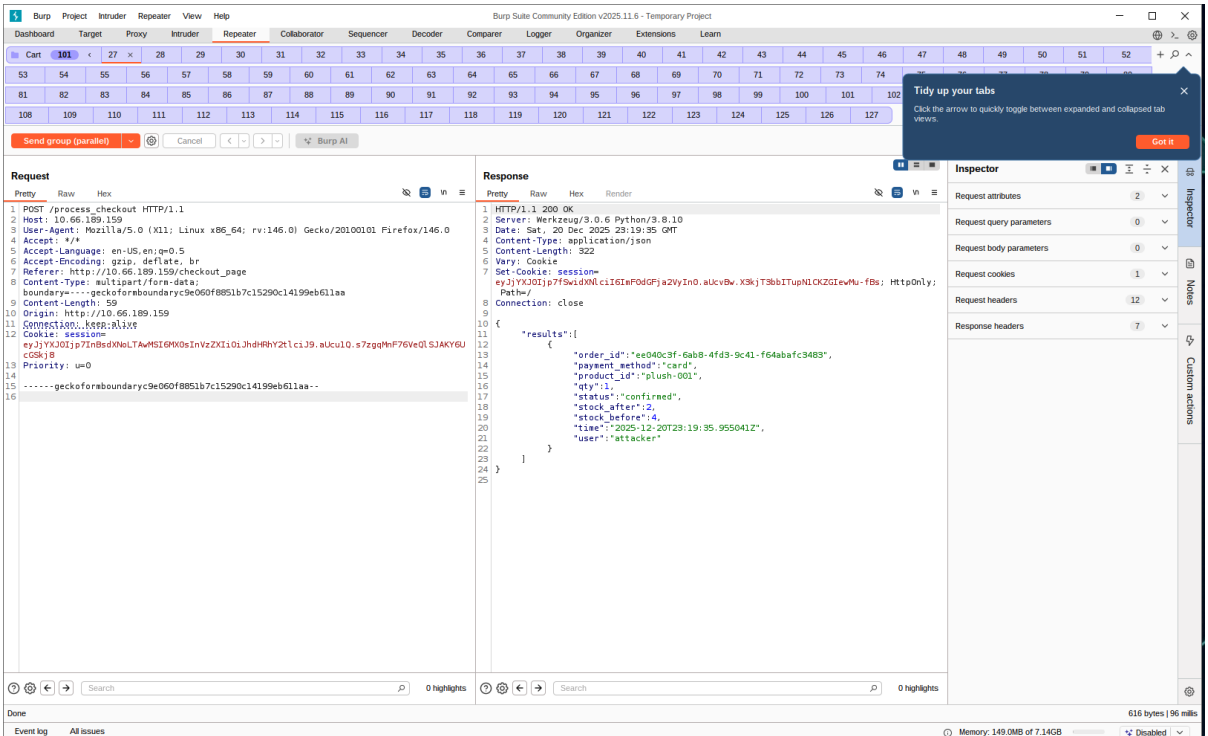
1. Start target machine and connect to vpn
2. **Proxy Configuration:** Launched Burp Suite Community Edition and immediately disabled the Intercept feature (Proxy → Intercept → Intercept is off) to allow HTTP traffic to flow freely while still being logged in the HTTP history
3. **Application Access:** Navigated to the target IP address in Firefox to access the TBFC shopping cart login page
4. **Authentication:** Logged into the application using the provided credentials: `attacker:attacker@123`
5. **Baseline Transaction:** Executed a legitimate purchase workflow by adding the limited-edition SleighToy to the shopping cart and proceeding through the checkout process. This established a valid request pattern to replicate.
6. **Request Capture:** Switched to Burp Suite and navigated to Proxy → HTTP History. Located the critical `POST` request to the `/process_checkout` endpoint that was generated during the legitimate purchase. Right-clicked the request and selected "Send to Repeater" to begin the exploitation phase.
7. **Tab Group Creation:** In the Repeater tab, right-clicked on the request and selected "Add tab to group" → "Create tab group". Named the group appropriately (e.g., "race-condition-attack") to organize the duplicated requests.
8. **Request Duplication:** Right-clicked the request tab within the group and selected "Duplicate tab". When prompted for the number of duplicates, entered **25** to create 25 identical concurrent requests, maximizing the probability of successfully exploiting the race condition.
9. **Parallel Execution:** Selected "Send group in parallel (last-byte sync)" from the Send dropdown menu in the Repeater toolbar. This critical feature synchronizes all requests to arrive at the server simultaneously, exploiting the timing window between inventory

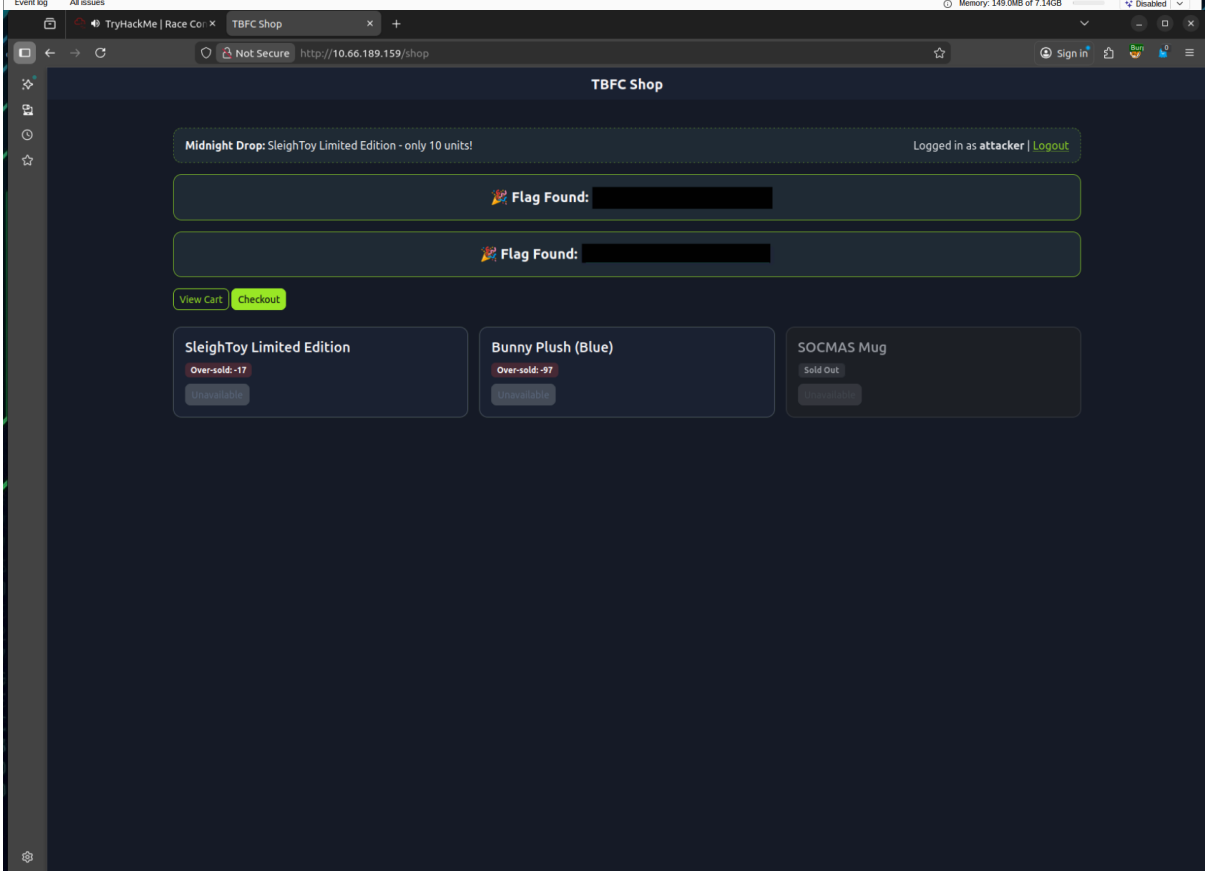
check and stock update.

1. 

2. 

10. **Extended Exploitation:** Repeated the entire process (steps 5-9) for additional items in the store to demonstrate the vulnerability was not limited to a single product. This confirmed the systemic nature of the race condition across the application's checkout functionality.

1. 

2. 

Lessons Learned

- **Core Vulnerability:** The application violated the atomicity principle by separating inventory checks from stock updates, allowing multiple concurrent requests to bypass inventory controls. Proper mitigation requires atomic database transactions with row-level

locking and idempotency keys to prevent duplicate order processing.

- **Attack Technique:** Burp Suite's "Send group in parallel (last-byte sync)" feature is essential for exploiting race conditions, as it maximizes timing overlap between concurrent requests. This methodology (capture → duplicate → synchronize → exploit) can be applied to test other TOCTOU vulnerabilities in web applications.

Resources

TryHackMe

Race Conditions

Revision #2
Created 2025-12-21 00:40:37 UTC by David Rizzo
Updated 2025-12-21 00:45:48 UTC by David Rizzo