# Exploiting File Upload Race Conditions

## Introduction

**race conditions -** common class of vulnerabilities closely related to business logic flaws. They occur when a web application **processes multiple requests concurrently without proper synchronization**. As a result, different **threads interact with the same resource at the same time**, causing inconsistent state or behavior that the application was never designed to handle.

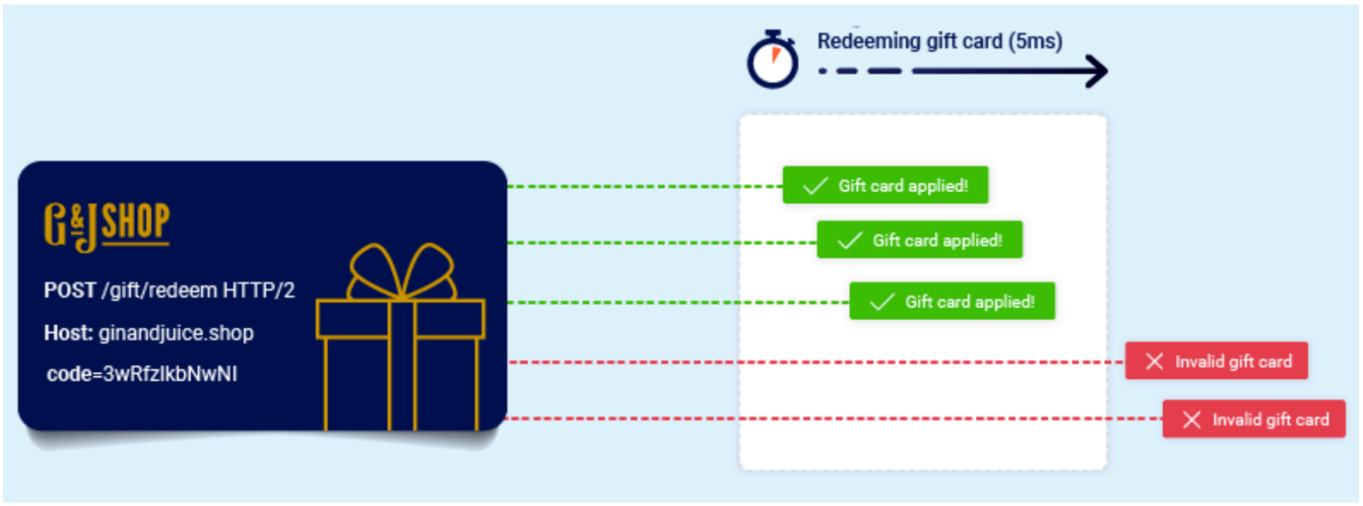
Modern file-handling pipelines typically upload files into a **sandboxed temporary location** with a randomized name to prevent overwriting. Validation is performed on this temporary file, and only after it is confirmed safe is the file moved to its final destination. If any part of this sequence is vulnerable to concurrent access, an attacker may exploit timing discrepancies to bypass validation or manipulate the file-processing workflow.

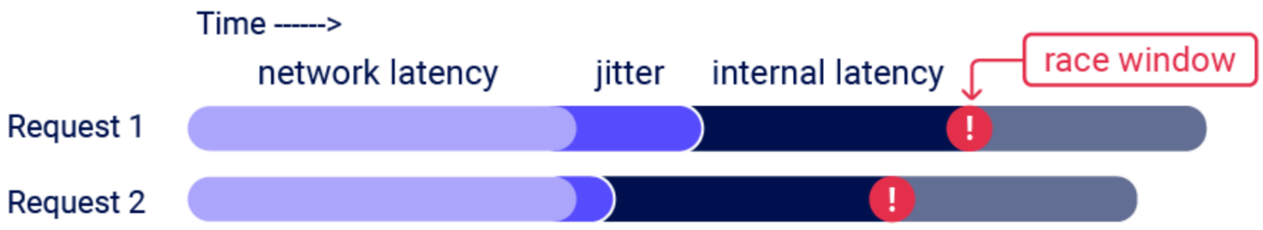
Developers sometimes implement their own processing of file uploads independently of any framework. Not only is this **fairly complex** to do well, it can also **introduce dangerous race conditions** that enable an attacker to completely bypass even the most robust validation.

Such vulnerabilities **lay deep in the web app logic** and **can be very hard to detect** in development phase, unless it's discovered during penetration testing or exploited by a thread actor.

## Example

Flawed file upload mechanism, can upload the file directly to the main filesystem and then remove it again if it doesn't pass validation. This kind of behavior is typical in websites that rely on anti-virus software and the like to check for malware. This may only take a few milliseconds, but for the short time that the file exists on the server, the attacker can potentially still execute it.





## LAB

**goal:** examine a file upload mechanism that is vulnerable for race condition exploit.
**requirements:** Internet connection, BurpSuit, PortSwigger account
**LAB:** link

## Hints

**hint 1:** find file upload function and try to see how it works.

**hint 2:** send payload `PHP` file see what happens.

**hint 3:** obtain `GET` request for valid avatar and edit it to try to access payload file.

**hint 4:** send both `POST` and `GET` requests to repeater, group them and try to exploit race condition.