Race Condition vulnerability and attack

TCS 591 : Unit 2

What Is a Race Condition Vulnerability?

- A race condition attack happens when a computing system that's designed to handle tasks in a specific sequence is forced to perform two or more operations simultaneously.
- Other names used to refer to this vulnerability include Time of Check/Time of Use or **TOC/TOU** attacks.

What Is a Race Condition Vulnerability?

A race condition is when a system uses a resource that can be accessed concurrently (by the system itself or others). An attacker is able to take advantage of the **time gap** between:

1. when the system performs a security control on the shared resource;
2. when the system performs an operation on this shared resource.

If the attacker manages to change the shared resource between 1 and 2, then a race condition vulnerability is present

Race Condition : Real Life Example

ATMs are designed to handle one withdrawal at a time of your remaining balance.

Let's say your remaining balance is Rs 50 and you arrange with a friend to withdraw your Rs 50 remaining balance from **two different ATMs** at the same time.

If you and your friend are successful in withdrawing Rs 50 each simultaneously from two different ATMs then you have successfully exploited a race condition attack on your bank

Race Condition Attack

- Race conditions can be found in many common attack vectors, including web applications, file systems, and networking environments.
- The good news is that, because they rely on attackers exploiting a system's process during a very short window of time, race conditions are somewhat rare.
- Most attackers will instead focus on easier types of attack, such as SQL injection vulnerabilities.

Race Condition Attack

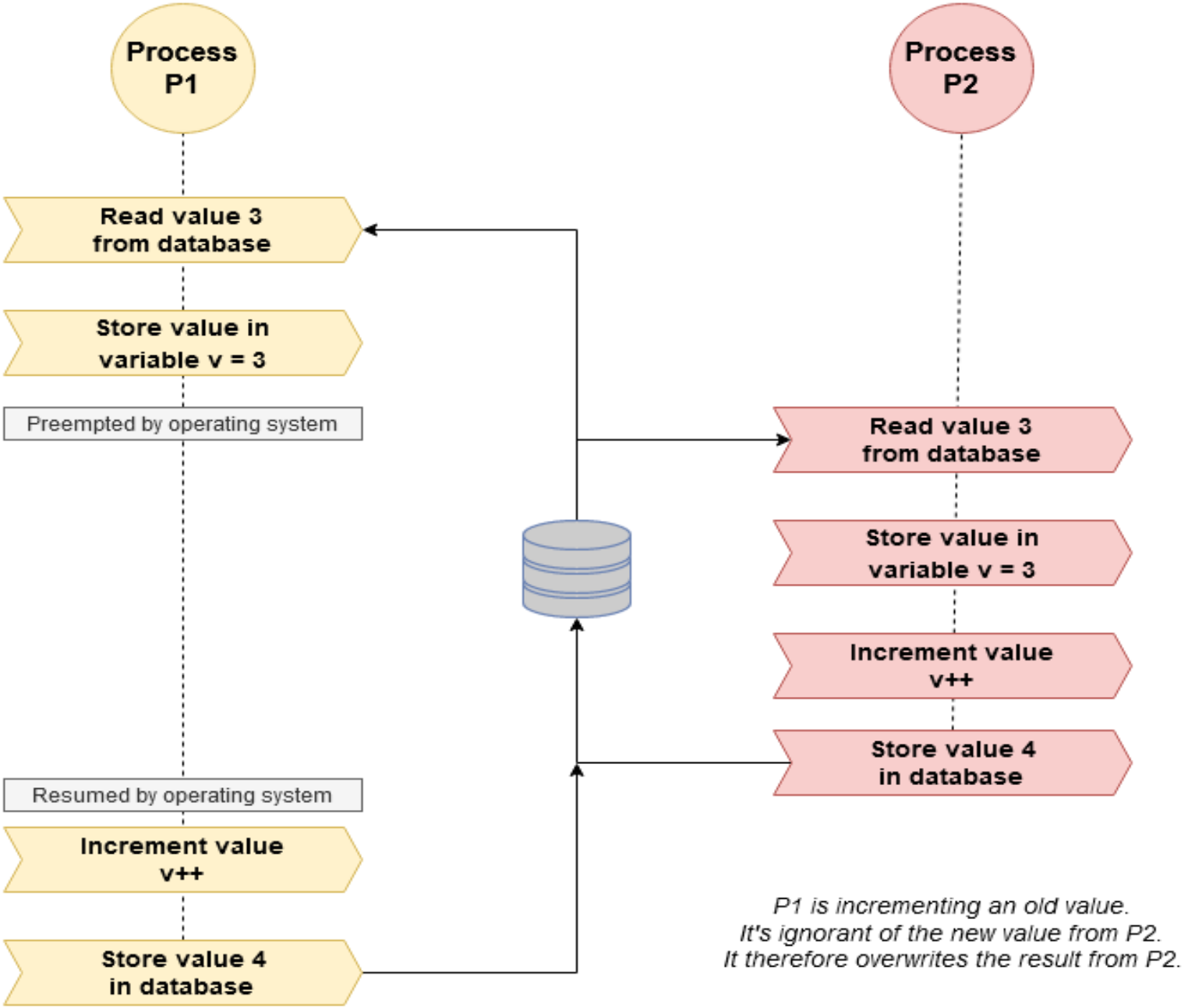
- Race condition attacks can lead to serious leaks and grant attackers complete access to secured systems, confidential data, and other information they shouldn't be able to view.
- Just a few of the types of systems that can be targeted with this kind of attack are access control list (ACL), a payroll or human resources database, a transactional system, a financial ledger, or any other data repository.

Race Condition Attack

- Almost all programs and web applications today use what is called **"multi-threaded"** processing, where they are capable of executing multiple actions at once.
- Although this allows applications to be significantly faster, it does introduce the potential for errors if more than one process (or "thread") tries to access the same data at the same time.

Race Condition Attack : Example

- when a SQL system performs an update to a database, it creates a temporary file during the update process.
- It's this temporary file that eventually replaces the data in the database.
- With a well-timed attack, malicious users can replace the temporary file for a SQL update of an administrative access table with one of their own, granting themselves administrator privileges to the system.



Illustrating software race condition. Source: Devopedia 2020.

Preventing Race Conditions

- To prevent race conditions from occurring you must make sure that the critical section is executed as an atomic instruction.
- That means that once a single thread is executing it, no other threads can execute it until the first thread has left the critical section.
- Race conditions can be avoided by proper thread synchronization in critical sections. Thread synchronization can be achieved using a synchronized block of Java code.
- Thread synchronization can also be achieved using other synchronization constructs like locks or atomic variables like java.util.concurrent.atomic.AtomicInteger

Preventing Race Conditions

- The usual solution to avoid race condition is to **serialize access to the shared resource.**
- If one process gains access first, the resource is **"locked"** so that other processes have to wait for the resource to become available.
- Even if the operating system allows other processes to execute, they will get blocked on the resource.
- This allows the first process to access and update the resource safely. Once done, it will **"release"** the resource. One of the processes waiting on the resource will now get its chance to access the resource.
- Code protected this way using locks is called **critical section**. The idea of granting access to only one process is called **mutual exclusion**.