IMT3501 Software Security

Race Conditions

Obligatory assignment 2, 12HBISA

Anders Storsveen,120928

17. september 2014

Chapter 1

Introduction

This assignment consist of three objectives: Imagine an electronic id solution, think about two (or more) possible race conditions and their effects and to describe countermeasures for the race conditions I observe.

1.1 Race Condition

A *Race Condition* occurs when two treads access a shared variable at the same time [1]. One thread reads the value of the variable, and starts to perform its operations. Before the first thread finishes its operations and write back to the variable, the second thread reads the original value of the variable. The first thread to writes its changes back to the variable, get its value overwritten by the other thread. Therefore only the value of the thread that writes its value last gets preserved. The reason race conditions can occur is because the operations are not atomically executed. To prevent race conditions we can declare a portion of our code as *critical region* and let thread lock a semaphore or a mutex when they enter the critical region, to stop other threads accessing the critical region (protected by the same semaphore/mutex). A critical region is defined as piece of code that accesses a shared resource that must not be concurrently accessed by more that one thread at a given time [2].

Race conditions is difficult to reproduce and debug, since the end result is nondeterministic and depent on the relative timing between interfering threads. It is therefor better to avoid race conditions by careful software design rather than attempting to fix them afterwards [3].

1.2 TOCTTOU

In software development, *time of check to time of use (TOCTTOU)* is a class of software bug caused by changes in a system between checking of a condition and the use of the results of that check [4]. For an adversary to be able to exploit a TOCTTOU race condition, he or she needs to attack at the exact timing between the time of check and the time of use of the shared resource.

Chapter 2

Race Conditions in an E-voting System

Imagine a simple system for registration of votes, where the votes are counted by incrementing a shared integer for each political party, without sufficient synchronisation mechanisms.

2.1 Race Condition 1

One possible race condition for this system might be that two voters vote for the same party concurrently. The first voter read the current value of the counter before the process increment the value. While the first voters process is changing the data, a second voter reads the same value of the counter. Both voters have incremented the counter value by 1. Only the last voter to write back to the counter variable get its vote registered. The total votes for the political party is only incremented by 1, and the voter that did not get its vote registered does not know that the vote got overwritten by someone else.

2.2 Race Condition 2

Another race condition that could occur is that two users simultaneously tries to log in to the system. The first user submit its credentials and get authenticated by the server, but before the system return acceptance to the user, another user logs in and the same situation arise. Then the system switches back and returns the first users interface to the second user, therefore it is possible for the second user to log in as the first user. This it possible because of the vulnerability known ad TOCTTOU, as described in section 1.2. By constantly sending logon credentials to the system, an attacker can get access to other users account if they log in simultaneously.

2.3 Avoid the Race Conditions

In order to prevent the race condition described in section 2.1, we have to do the registration as an atomic operation. We can achieve this using a mutex. We have to lock the mutex before the porcess enters the critical region when it reads the counter value, and release the mutex after we have written the new value back. To prevent TOCTTOU race conditions, from section 2.2, we can treat the login mechanism as a transaction. That means to run the whole authentication as a atomic operation. If we choose to do this we have to lock the process in a critical region from before the username and password is checked, and release it after the user has bin granted/denied access.

Referanser

[1] Description of race conditions and deadlocks. Microsoft. [Online]. Available: http://support.microsoft.com/kb/317723

[2] Critical section. [Online]. Available: http://en.wikipedia.org/wiki/Critical_section

[3] Race condition. [Online]. Available: http://en.wikipedia.org/wiki/Race_condition

[4] Time of check to time of use. [Online]. Available: http://en.wikipedia.org/wiki/Time_of_check_to_time_of_use