Gjøvik University College

Software Security

Assignment #2

Electronic Voting System - Race Conditions

Victor Rudolfsson - 120912

September 17, 2014

1 Electronic Voting System - Imaginary Race Conditions

I wasn't sure what type of system to pick, so I decided to go with the provided example. As there was no given specifications for the system, I decided to pick a system designed by a not-so-up-to-date PHP programmer implementing a voting system with a custom log in system in Java, and a voting system for authenticated users written in an unspecified language.

1.1 Logging in

The developer of this authentication system is a PHP ninja but a mere novice when it comes to Java. As such, he has developed the log in system using Java HTTPServlet but has failed to take into consideration that unless properly configured, the servlet runs as according to the singleton pattern. This means there is only one servlet, and any data in this is shared between all connections. It is therefore vital to make sure the correct data is accessed only by the correct user.

One of the most important part of a users session, is the users ID. The users ID determines what account the user is logged in to. When a new sign in occurs, the users ID is retrieved and pushed to the list of currently logged in users, after which it is returned and set as the currently logged in user.

All of this worked fine during the testing process, and even testing on multiple computers simultaneously. It was only after opening the system up to more users that this turned out to be an issue – support ticket after support ticket started flooding the customer service department, as customers logging in occasionally ended up logged in to somebody elses account. [1]

What had happened was that after the users credentials had been validated, the users ID was retrieved from the database and pushed on top of the list of currently logged in users. Somebody else would then log in, and his/her user ID would be pushed on top of that list, before the most recent item was retrieved for the first user. Now, the first user instead got the second users ID, and the second user may or may not have become logged in to the first users (or a third's) account.

1.1.1 Solution

The issue here is, as it often is with race conditions, that the data storage is shared between multiple entities trying to access it. This issue would have been resolved if the servlet would have been configured with SingleThreadModel: By implementing the SingleThreadModel interface, the servlet will run in a different instance for each request, and guarantees that no two threads (or requests) will execute the same service method of that servlet at the same time. [2]

1.2 Voting

The user is logged in to the global e-voting system, which allows users to set up polls and vote on anything, and the user decides to vote on one of the active polls. The poll includes several different colors, and the user decides to vote black. He clicks the black color, and the vote has been cast (in the users eye).

What really happens, though, is the value is retrieved from the current statistics at 219540062 votes on black. The value is the incremented by one, yielding 219540063. However, at the same time, another user have also voted, without the value being locked before incrementing. This means both of them retrieved the value 219540062, incremented it by one each, and updated the value.

Black now has 219540063 votes, whereas it should have had 219540064. One of the votes got lost in the update process, as a race condition was created.

1.2.1 Solution

A solution to this would have been to lock the value down, and not allow the value to be retrieved until it was updated by the first user. This could have been done by using transactions in the database, and queueing up the actions to be performed, making sure they're performed in the correct order, and that a row that has been retrieved is locked down during updating, before it's allowed to perform another update on it.

References

[1] HP Enterprise Security. Race condition: Singleton member field. http://www.hpenterprisesecurity.com/vulncat/en/vulncat/java/singleton_member_field_race_condition.html. Accessed Sep 15, 2014.

[2] Niki A. Rahimi. Java servlet programming - 3.4 single-thread model. http://repo.hackerzvoice.net/depot_madchat/ebooks/Oreilly_Nutshells/books/javaenterprise/servlet/ch03_04.htm. Accessed Sep 15, 2014.