Assignment # 2: A Secure E-voting System

120917
12HBISA
Eirik Vestreng Solberg

September 17, 2014

Introduction

In this assignment I will describe the implementation of a secure E-voting application. I will not go into programming language specifics, but some pseudo-code will occur.

General system specifications

The application should be able to handle at least 50000 user simultaneously and through an web application. It is a must that the integrity of each vote and the anonymity of each voter is upheld.

Scope

A application created for this purpose is extensive and I will therefore not go into details on anything except the security specifications focused on race conditions and integrity.

System Description

Login module

The login is implemented with a 2-way authentication with something the user knows and possess, in this application these are a password known only by the user and a code chip which is generating a semi random codestring by a press on a button. This is a secure enough sollution and I will not go into more detail on it.

Voting module

This is the main concern of the application. Since there will most likely always be more than one user the integrity in the numbers of votes will be at risk if no safety measures is in place.

note: In most of the Solutions below there is an User ID, this is a anonymous value only created for the purpose hindering double-voting and such.

Threats

Some votes may not be counted as they happen at the same time and therefore possibly setting wrong values.

Solutions

To this problem there is several methods of solving it I have chooses three to discuss. A question which is needed to be answered: Is the application in charge of taking care of the variables itself or is this handled by an database?

Solution 1: If the application itself are responsible for the variables and number of votes. I would implemented it as such:

```
public Stack hasVoted;

public binarySemaphore sem // same as a mutex

function giveVote(Voter userID, Vote x){
        sem.lock();

        addVote(x);
        hasVoted.push(userID);

        sem.unlock();
}
```

As shown in the code above all of the race condition critical functions is locked by a mutex making it impossible for two users to manipulating data at the same time. problems with this is that it will be really slow and create annoyance from the user because of this as it does not implement a good use of multi-core systems.

Solution 2: Since solution 1 will create a bottleneck the moment the application gets more than 100 users at the same time lets find some other solutions. Lets make the database handle the variable integrity for us. by using this code

```
SqlController controller; // is initialized somewhere else

void GiveVote(String userID, Vote v){
        String prepUID = controller.prep(userID); // preparing the statement
        controller.exec(prepUID, "hasVoted");

        String prepVote = controller.prep(v);
        controller.exec(prepVote, "Vote");

}
```

This code calls these two SQL statements which is prepared by the SqlController and sent to the database. *note:* the parameter variables are protected from all manipulation from the outside.

```
UPDATE Votes
SET noOfVotes = noOfVotes+1
WHERE politicalPartyId = @a;

INSERT INTO hasVoted (voterID)
VALUES (@a);
```

Where @a is the variable sent from the application. The SQL database is set to Pessimistic read/write to protect every vote. The voters id is put in an database heap for log purposes and to ensure that everyone can vote just once. The cons of this approach is that it only moves the bottleneck from the application over on the database server and it is troublesome if the voter change his/her mind and want to vote for someone else.

Solution 3: Both of the two other system demands incredible amounts of computational power and will most likely create bottlenecks anyway I have made one more solution. This solution postpone the computational strain until after everyone has voted and thereby lower the required capacity of the system. In this version all the data is transferred to the database but instead of manipulating field its just adds everything to a heap like a log file until the election is over and the operator can then start a SQL script to count the votes using this code:

```
public SqlController controller;

void giveVote(Voter voterID, Vote v){
        String prepVoter = controller.prep(Voter.getValue());
        String prepVote = controller.prep(v.getValue());

        controller.exec(prepVoter,prepVote,"giveVote");
}
```

the sql for this is:

```
INSERT INTO AllVotes (voterID, Vote)
VALUES (@a, @b);
```

This removes most of the computational strain in the system under the election but creates some strain after. this will probably be the most cost-efficient solution.

Conclusion

Race conditions are a constant threat to applications integrity and it something to be very careful of. Therefore it is wise to allow databases and other assisting systems take care of data if possible especially if they are the size of this database with the purpose of handling all the votes of an election.