Instituto Superior Técnico

Network and Computer Security

1st Semester 2017/18

Secure online auction website

Taguspark

Group 36

João Oliveira
72944

Gonçalo Louro
78982

Matilde Nascimento
82083

Problem

Nowadays, web application security is an important problem in the internet, since web applications are popular targets of security attacks. The main cause is that programmers do not have sufficient knowledge about secure coding, so they develop applications with vulnerabilities that can be exploited by attackers. The most common type of attacks are *SQL injection* and *Cross-Site Scripting*.

The project will consist on a online website to secure auctions. The main issue is that the application will communicate over the Internet, where information can easily be intercepted or changed in transit.

Requirements

- Access control: the access to data must be restricted to authorized entities;
- Authenticity and Nonrepudiation: the data must come from the entity it is supposed to come from;
- Confidentiality: the information stored in our website must not reach the wrong people and, simultaneously, the right people can in fact reach it;
- Integrity: the data must be consistent, accurate and trustworthy; it must not be changed at any time, and cannot be altered by the wrong people;
- Availability: authorized people should be able to access the allowed information when needed;
- Freshness: ensure that the website is secure against replay attacks.

Proposed solution

In order to build a security oriented service, we must face the problem step by step starting from a basic solution and gradually increase its complexity by adding new security features until we reach a medium security level and finally an advanced one. For each security level we will cover the correspondent complexity of attacks.

We start with a basic service with a simple website architecture working as an interface for our clients, connected to one or more servers who are then connected to another server running the database.

Basic Level Security – Website

Since the website is the interface for our service we decided to cover the main area of the attack surface which is our website users. We will have a create account/login area, where we will get the user's info (username, password, etc.).

We identified and proposed to prevent the most common threats as Cross Site Scripting (XSS) and SQL Injection. We also proposed to use a secure communication protocol (HTTPS).

Medium Level Security – Server and Database

For a medium level security we decided that the database should be inside a private network connected to our server. Important information inside the database (passwords and address, for example) should be saved in form of a hash. The server needs to be public so we'll need a firewall to filter incoming connections and data. The server should also guarantee that the messages are fresh and authentic.

Advanced Level Security – Fault prevention and tolerance

Finally we decided that our system should be fault tolerant. To accomplish this we proposed to implement a primary-backup replication. In order to prevent denial of service attacks our firewall should also filter the incoming connections.

Given the proposed solution, we made an illustration of the architecture of our network (figure 1).

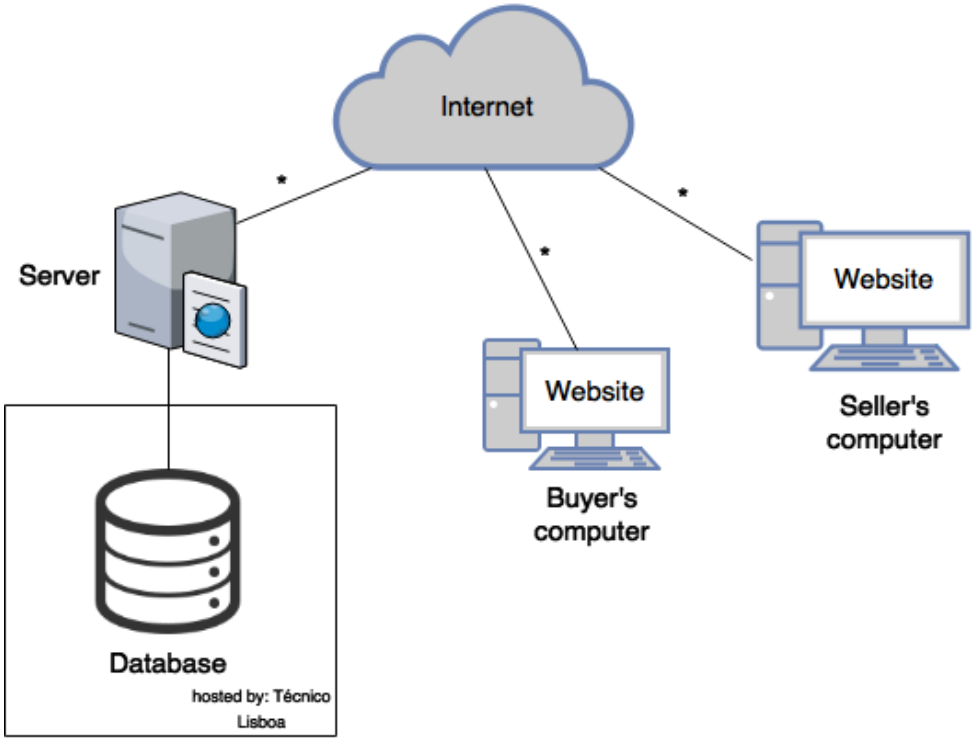

Fig. 1 - Diagram of our proposed solution.

Results

Basic Solution:

As planned, the website, server and database were successfully implemented. We also prevent XSS and SQL injection attacks, and the support for secure communication over HTTPS was added to the server and website.

Intermediate Solution:

The database is running in *Técnico Lisboa*'s server (host: *db.ist.utl.pt*). All the confidential information (user's password, for example) is being saved in form of an hash, using salt.

We used the firewall to accept packets only from port 8443 (which is the one we are using for Spring Boot), and to reject the others.

Advanced Solution:

The replication was successfully implemented.

Evaluation

We think that our solution is secure and actually simulates the way real secure websites are built in the industry.

Weaknesses:

1. The project only tolerates one failure; if the primary and backup server crashes, the website is not available anymore.
2. By using HTTPS, we can't guarantee confidentiality in the communication between the server and the website.
3. The security is not high level.

Strengths:

1. Storing salted password hashes means that if an attacker gains access to it, it will still not know the passwords (it guarantees confidentiality).
2. Using HTTPS, we can guarantee confidentiality in the communication between the Internet and the server, in case of a man-in-the-middle attack.
3. Using *Técnico Lisboa*'s server for the database, we have a higher level of security, guaranteeing more stability. In case of an attack to the server, having the database running on a different machine from the server, the attacker has less power to corrupt the database .
4. Having protection against SQL injection, assures confidentiality and integrity over the user's information.
5. Having a firewall, we can only accept packets from the port we want to.
6. Having protection against XSS, assures confidentiality on the website.
7. Having freshness, we assure that a message is unique and not reused. Thus, we avoid replay attacks.
8. With replication, we assure availability; in case of one problem in the primary server, the application still works (with the backup server).

Conclusion

We were able to implement a system that assures the security requirements that a site of this type requires.

The proposed solution that we intended to do, was not done completely. During the development of the project, we learned that some tools/methods were not what we wanted to apply in the project, and others were not done; however, there was room for improvement, given additional time.

References

Tools

Database:

- *MySQL Server*.

Server:

- To exchange information between the server and the website, we used *JSon*.
- To implement salted hashes, we used *Base64*.
- To prevent SQL Injection, we used *jdbc*, more specifically, the function *PreparedStatement*.
- For the replication, we used *ServerSocket* class.
- To guarantee freshness, we used *Nonces*.
- For the firewall, we used *iptables*.

Website:

- Bootstrap: *HTML*, *CSS*, and *JS* library.
- To prevent XSS, we used *org.apache.commons* (that contains functions to escape html characters).

Database/Server/Website:

- *Thymeleaf Spring Boot*.