Integrated MSc Course on Informatics Engineering, DI/FCT/UNL
Computer Networks and Systems Security / Semester 1, 2019-2020
WORK-ASSIGNMENT #2 REPORT for Evaluation

Secure Blockchain-Enabled Auction Management System

REPORT

Authors:
Manuella Vieira (mra.vieira@campus.fct.unl.pt),
Rafael Gameiro (rr.gameiro@campus.fct.unl.pt)

Summary

The project hereby presented addressed the development of a system that enables users to create and participate in auctions in a blockchain-enabled management environment. The proposed system was required to have a set of security features, such as honesty assurance (ensuring that bids and auctions were publicly accessible to all clients), bid author identity under anonymity, bid acceptance control and confirmation, bids' confidentiality, integrity and authentication, and the protection of the communication channels via TLS. The initial design of the project considered the establishment of all the previously mentioned security guarantees, however, the effectively implemented version of the system presented a few drawbacks, which will be reported in more detail throughout this paper, along with the implemented solutions.

1. Introduction

An auction is a process of buying and selling goods or service by offering them up for a bid [1]. In an auction each participant make a bid for an higher price than the current one and in the end the winner is the one who did the bid with the highest price.

A Blockchain is a data structure, that in the bitcoin context, each entry is a transaction digitally signed in order to secure authenticity and prevent changes. That way, all the transactions inside the blockchain will have an high level of integrity [2].

Our project consisted in the join of these concepts and creating a auction management system, where each auction is implemented by a blockchain in order to provide full integrity during the auction process. In this project, a user can create auctions and make bids for them. The user can also list all the auctions until now, list all the bids made and check the winner of a specific auction.

Throughout this report we will present the system model and the architecture

implemented in this project. We will specify the security properties our project provides, the software used to accelerate and simplify the project development, and the validations and experiments made to verify our functionalities.

To conclude, we will present the final considerations of our project, the drawbacks, limitations and functionalities not implemented, and address some solutions that could be considered in a future work.

2. System model and architecture

2.1 System model

The Auction Management System is composed by three main entities: Auction Client, Auction Manager and Auction Repository. The Auction Client is an application that allows the user to interact with the Auction Management System by sending requests to an Auction Manager and presenting the corresponding responses to the user. It is through this entity that the client can create and choose to participate in auctions, make bids and so on.

The Auction Manager or Auction Server is a REST endpoint responsible for mediating the communication between clients and the Auction Repository in order to provide more control over the data being exchanged and take some of the processing burden away from the Repository and into a more suitable entity. This entity is responsible, for example, for broadcasting relevant updates to all the clients "subscribed" to a specific auction.

The Auction Repository acts mostly as a database, persistently storing information about all the open and closed auctions and their respective bids, as well as relevant data about the clients, such as their unique identifier and the endpoints to which they are connected.

An overview of the system model and how these entities are related can be seen in Figure 1. All of the aforementioned entities communicate with each other through HTTPS (except communications between Auction Manager and Auction Repository) requests and responses, with messages being exchanged as JSON objects.

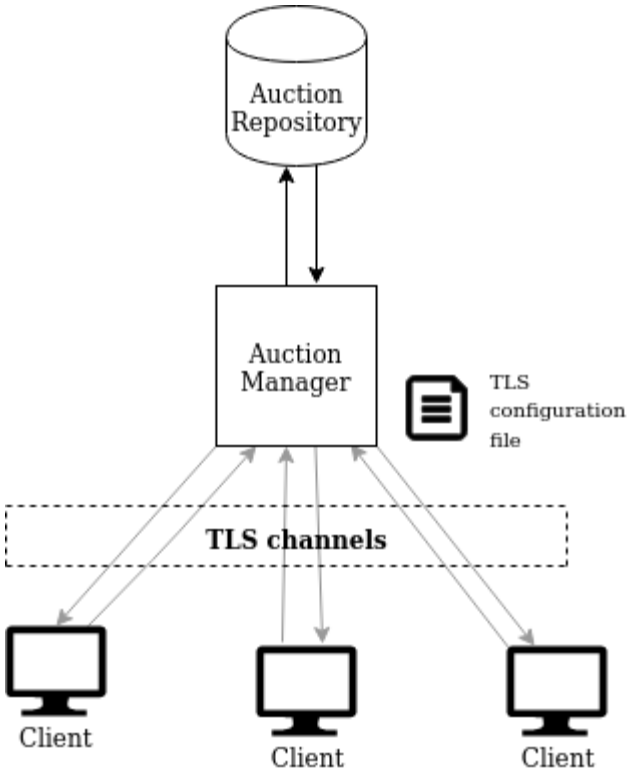

Figure 1 - Overview of the system's architecture

2.2 Architecture

In this section, a more detailed description of all the entities, their respective components and supported services will be provided. In essence, the implemented system supports the following operations:

- Create New Auction: allows the user to create an auction that is then going to be available for bidders in the system
- List Auctions: lists all the auctions registered in the system, whether open or closed
- List Auction's Bids: lists all the bids from a specific auction
- List Client Bids: lists all bids made by the client
- Make New Bid: allows the user to make new bids on any of the registered auctions
- Close Auction: allows the user to attempt at closing an auction they have participated in or an auction they have created
- Check Auction Outcome: allows the user to check who won a specific auction and with what bid amount

Each of the operations are accomplished by following a general set of steps. Initially, the Auction Client sends a request to the Auction Manager either asking for information (such as a list of auctions or a list of bids) or sending information (such as a new bid or a new auction). The Manager then forwards the request to the Auction Repository, which retrieves

the solicited data from its storage and sends it as a response to the Manager. Lastly, the Manager forwards the retrieved data back to the Auction Client.

When the system runs for the first time, the user is prompted to choose a username and the server generates a unique user identifier (UUID) for that client. This UUID is then stored by the client application, so when the client needs to make new requests, the identifier is automatically retrieved and sent to the server.

The bids in an auction are stored as a linked list of blocks, in a way analogous to how a blockchain works, with bids representing the blocks. A block object is composed of the bid's content (amount and author), and two integrity hashes. The first hash is computed over the bid's own content and the second is computed over a combination of the bid's own hash and the hash of the previous bid.

At the time of an auction's creation, the user is prompted to choose which type of auction they wish to create - namely, "Blind" or "English" - as well as basic information about the auction such as a name and description. If the chosen auction type is English, each new bid made to the auction will be required to have a value that is higher than the previous one.

If a user chooses to close an auction they have participated in, they are required to solve a cryptopuzzle. The cryptopuzzle consists in brute-forcing for a solution where a message digest of the "combined hash" attribute of the auction's last bid plus a random value results in a hexadecimal representation of the hash where the last five digits are zeros. Once the client finds the random that solves the puzzle, they send this number to the server so that it can check the validity of the solution. Closing an auction here means that the auction will no longer be mutable (will not accept new bids) and a winner will be chosen.

2.3 Threat model

For this project we only considered an external threat model, where it’s not possible to do attacks inside the program itself, since the authentication of the program is provided during the login phase, and during establishment of the TLS protocol, when required.

The overall properties of the system are mainly guaranteed thanks to the use of TLS channels between the client and the server. The connection between the servers (Auction Manager and Auction Repository) was considered to be secure and therefore not taken into account as a possible vulnerability of the system.

The TLS channels provide between the client and server, message and data-flow integrity (including message ordering control), message and data confidentiality, and peer authentication and message authentication [3]. All these services are only guaranteed during the TLS session and therefore, connection oriented. The peer authentication protection given by the TLS might be only one-sided since during the establishment of the TLS connection, mutual authentication might not be required, so only the server or the client provide proofs of authenticity.

Also, during the auctions, the integrity of the bids are guaranteed by the blockchain concept but mostly by the proof-of-work that gives full integrity of the entire data structure.

The attack surface can then be described as attacks of replaying, Denial of Service, Traffic confidentiality and traffic-flow confidentiality.

3 Implementation details

In order to implement the Auction Server, Auction Repository and also the server side client, we used the Grizzly NIO framework which is a framework designed to help java Developers build scalable and robust servers using the New IO API [4]. In order to create a communication channel between the client and all the servers we use REST on top of TLS protocol.

REST, also known as Representational State Transfer is an architectural style used for distributed systems that allows the separation between client and server in order to improve portability and scalability properties in systems [5]. To fully use the REST functionalities we used the framework Jersey, which is an open source framework for developing RESTful Services in java that provides support for JAX-RS and serves as a reference implementation [6].

TLS or Transport Layer Security was a protocol designed in 1999 and is a cryptographic protocol that provides end-to-end security of data between application components and between applications [7].

With Grizzly, REST and TLS we built simple to HTTPS servers that use RESTful services and implements secure channels using TLS.

The management of the project dependencies and in order to provide a good starting point of the project and allow a fast and easy to use build process we used the Apache Maven Project which is a tool focused on the management and building of Java-based projects [8].

To use cryptography throughout the project we used the JCE to do the proof of work needed to close an auction, and also to generate the hash of the current bid and combined hash of all the previous bids. The Java Cryptography Extension (JCE) provides a framework and implementation for encryption, key generation and key agreement, and Message Authentication Code algorithms [9].

4. Work Evaluation and Validation

Due to time constraints, the system was only tested in a localhost environment. Most of the evaluation and validation of the system consisted in checking whether the messages exchanged were in fact being received as expected and whether TLS was working according to the preset definitions.

The integrity of auctions and bids were validated by checking if the hashes computed on the server side matched the hashes computed on the client side.

In order to confirm that TLS was indeed using the specified configurations (i.e. the authentication mode, supported TLS versions and ciphersuites), the traffic between the

endpoints was inspected with the Wireshark tool. The handshake messages were analyzed and it was confirmed that whenever the server asks for mutual authentication, the client sends a response presenting its certificate, which does not happen when the chosen authentication mode is server-only. It was also checked that the TLS version and the ciphersuite being used were present in the configuration file.

5. Conclusion

We were able to create a Blockchain based auction system, where each auction is built based on a real blockchain system. The system allows to create auctions and bid for them using the security properties mentioned earlier.

Highlights for the fact that the operations performed by the user and communicated to the auction system are performed using REST with security channels with TLS. All the parameters needed to establish a secure channel are editable and the user can change them at will.

Some of the limitations and drawbacks previously mentioned, highlighting the system scalability, could be surpassed if the auction system's server were placed in a cloud environment in order to allow access to the system from any place and mitigate possible hardware limitations. The auction storage and management could be optimized using a Merkle Tree, like the real blockchain system uses. Asides from that, we consider that an attempt at preventing replay attacks could be achieved by adding timestamps to the bids and DDoS attacks could be attenuated by having the user compute a proof-of-work with each new bid.

Also, we considered that the two channels established between the user and server, in order to execute operations from the user side and to also notify the clients of possible auction changes, could be simplified if we used a WebSocket protocol. The security properties provided by TLS channels could be extended if we applied some extra cryptographic operations on the exchanged messages between the user and server.

To conclude, in order to provide a more friendly environment for the user we would implement a graphical interface so that the user can fully use all the provided services without the need of a terminal console.

References

[1] Auction definition, https://en.wikipedia.org/wiki/Auction (available and retrieved in December/2019)

[2] Blockchain definition from a Bitcoin perspective, https://computerworld.com.br/2016/06/22/blockchain-o-que-e-e-como-funciona/ (available and retrieved in December/2019)

[3] Course on Computer Networks and Systems Security, MSc Program in Informatics Engineering, DI/FCT/UNL 2019/2020, Transport Layer Security (TLS), HTTPS and WEB/HTTPS Security, December/2019.

[4] Grizzly framework, https://javaee.github.io/grizzly/ (available and retrieved December/2019).

[5] REST framework, https://restfulapi.net/ (available and retrieved in December/2019).

[6] Jersey framework, https://eclipse-ee4j.github.io/jersey/ (available and retrieved in December/2019).

[7] TLS definition, https://www.internetsociety.org/deploy360/tls/basics/ (available and retrieved in December/2019).

[8] Apache Maven Project introduction, https://maven.apache.org/what-is-maven.html (available and retrieved in December/2019).

[9] JCE definition, https://en.wikipedia.org/wiki/Java_Cryptography_Extension (available and retrieved in December/2019).