TÉCNICO LISBOA

Dependable Public Announcement Server

Highly Dependable Systems

Project - Stage 1

84708 - David Coimbra
84774 - Tomás Carrasco
87563 - Ricardo Bandeira

Implementation

We implement the DPAS system using the Java Remote Method Invocation (RMI) library. We develop three modules: a client, a server, and an interface mediator, which acts as an API for client-server interactions.

Server

The server is managed through the `ForumServer` class, where an instance of the `Forum` skeleton class is created or read from a serialized file and bound to an RMI `Registry` instance. This `Forum` class represents the current state of the system, housing a set of `Account` instances which are identified by the respective user's public key and contains the user's announcements board, through the `Board` class. The `Forum` also has a single instance of `Board`, representing the general board. The `Announcement` class represents a post, identifiable by a unique ID, corresponding to the SHA256 hash of its contents. Server responses are encapsulated in a `Response` object, which has different implementations depending on the nature of the response: `WriteResponse` for *post* operations, `ReadResponse` for *read* operations, and `ExceptionResponse` for errors, as well as a `NonceResponse` for sending nonces to the client. These objects include a signature from the server, and provide a means to verify it through a `verify` method.
The `ForumInterface` stub class provides an API implementing the following operations: `getNonce`, `register`, `post`, `postGeneral`, `read`, and `readGeneral`.

Client

The client module features a single Client class which employs a simple CLI (command line interface) to interact with the API. For every request, the CLI prompts the user for their KeyStore password in order to retrieve the corresponding private key.

Security requirements and solutions

- **Resistance to deliberate message drops:** an attacker can drop messages and prevent them from reaching the receiver. The system must detect this and resend the message.

- Solution: Java RMI is built upon the TCP/IP transport-layer protocol, which retransmits a packet after a timeout until an acknowledgment packet is received. As such, RMI will try to re-send messages by default, in an effort to ensure message delivery.
- **Resistance to replay attacks:** an attacker can repeat previously sent messages in order to bypass authenticity checks and carry out denials of service to the client. The sender must be able to detect repeated messages and reject them.
- Solution: We include a sequential `nonce` in all message signatures. We associate a `nonce` to each `Account`. Nonces start at 0, and begin when the corresponding account is registered, incrementing with every request from the client. For every request except `register`, the client must invoke the `getNonce` remote method before it can successfully interact with the server, carrying out a challenge-response handshake. After executing a command, the server increments the nonce and includes it in the signature of the response, which is, in turn, verified by the client. As the `nonce` is sequential, it does not need to be sent in cleartext. While the base `read` and `readGeneral` remote methods do not change the application state and as such have more relaxed security requirements, we decided to prevent replay attacks in those methods as well. Therefore, we expanded both methods to include the public key of the sender and a signature as extra parameters, in order to verify the corresponding nonce.
- There is a vulnerability associated with this solution: if the `register` command is invoked with a public key that is already registered, the server will always return 0 as the response nonce. This gives an attacker an opportunity to replay this response before the client makes another request, opening the possibility for a denial of service attack on the client.
- **Resistance to server failure:** the server can crash, losing or corrupting the existing announcements. The server must be able to recover in a valid state.
- Solution: we employ hardware redundancy through Java Object Serialization as a backup of the current state of the server. For every state change, two backup files are sequentially created. This ensures that there is at least one file available if the other one is corrupted during serialization. When the server starts, it checks for the first file and loads the second one if there are errors. If both are damaged or nonexistent, a new Forum instance is created from scratch. If there is an internal server error while serializing, an error is returned to the client.
- **Message integrity**: an attacker can alter messages while in transit between the sender and the receiver. The receiver must be able to verify whether a message was altered before it was received.
- Solution: A SHA256 hash of the message parameters is computed and sent with every message. It is then verified by the receiver and the message is rejected if there is a mismatch. Integrity is guaranteed for both the client's requests and the server's response, as an attacker cannot recompute the hash without invalidating the signature (addressed below). As we assume the server is honest, an announcement will not be altered once it has been posted.
- **Message authenticity:** attackers can pretend to be someone else (for example, a user can pretend to be another user, or a rogue server can pretend to be legitimate). The receiver must be able to ensure that the sender is who they claim to be.
- Solution: the hash included with all messages is signed with the sender's private key. As the private key is unique for the user, messages signed with it are unequivocally associated with that user, guaranteeing authenticity. The attacker cannot replace the signature with his

own, as the inclusion of a `nonce` that is associated with the sender's account will cause the signature verification to fail.

- **Non-repudiation:** since attackers can pretend to be legitimate users, users can repudiate authorship of announcements. At any time, it must be possible to verify that an announcement was posted by the associated author, and by no one else.
- Solution: non-repudiation of messages is guaranteed by the existence of signatures. Additionally, we guarantee non-repudiation for posted announcements by including the signature of the message that created the announcement, as well as the parameters of such message, in the announcement itself. As a result, it is always possible to verify the authorship of any announcement on the server.

To demonstrate both the functionality and security features of our implementation, we implement automatic tests using JUnit 5. Functionality tests are present in the `ForumTest` class while security tests belong to the `ForumSecurityTest` class.