Security Report of AP Project

Team 18

Abstract

In the following pages, for each item there is a description of the policies used in our project to secure the code.

1. **Replay Attack**
    - Establishing a random session key, which is a type of code that is only valid for one transaction and can't be used again :

      for important transactions a disposable token is created which cannot be used more than once.These tokens are created by server using *setDisposable()* function on a token.if they are used more than once an exception will be thrown.

    - Using timestamps on messages : for bank transactions the token given to each request has a valid period of time and then expires.this goal is achieved by giving this time period in token constructor (zero equals to no time limit).After a token expires using it will throw an exception.
2. **Improper Inputs**
    - Limit the length of input strings : In some parts of code which are listed as below, we checked that clients' inputs (as strings in TextFields) don't be longer than a maximum length of 30. If a client enters a long input in TextField, it will be considered as an improper input and an appropriate error will be shown. List of places that we implemented such policy follows as below :
        - ❏ Process of Registration
        - ❏ Process of Purchasing (Receiver's Information)
        - ❏ Process of Adding Product (Seller's Description)

- ❏ Process of creating discount code (Discount's code)

- Not allowing that someone changes the messages between server and client : This goal is achieved by...
  - Using a signed jwt in each message sent between server and client.Because the jwt is signed in standard form any change in the message body will make its signature so the whole token invalid and using such token will throw an exception.it is implemented in Token class \ Create JWS method.
  - Creating a hash from the message that is being transferred.This hash is unique for a given message and will be invalid if the message body is changed.it is implemented in *generateHASH(byte[] message)* method.Also the transferred file is encrypted using AES algorithm for more safety.it is implemented in FileManagerMenu *encrypt* and *decrypt* method and also *DataEnDecrypt* class.

3. Broken Authentication

- Limit failed login attempts :

  If a user attempts to login more than 6 times and all of them are failed then we show an error and disconnect the user from the shop's server. (This is implemented in controller(package) -> Manager(class) -> login (method).

- Align password length, complexity and rotation policies :

Our shop has some policies to align passwords complexity that will be checked while the process of client's registration :

- ❏ Passwords should be more than 8 characters
- ❏ Passwords should contain at least one digit
- ❏ Passwords should contain at least one uppercase letter
- ❏ Passwords should contain at least one lowercase letter

4. Brute Force Attack

Our shop's policy to handle Brute Force Attack is basically similar to DOS which is described in the next part. Every 10 seconds a timer checks the number of requests that each client (with specific IP address) sent to the server and they failed. If the number of these failed requests exceeded 5, then it automatically disconnects the client from the server. Therefore, that client's IP will go to the black list and he cannot send any more requests to the server. This policy was implemented in server(package) -> Server (class).

There is an inner class named "*CheckIps*" which extends TimerTask. At first for each client a timer is scheduled to call this class's run method every 10 seconds.

5. DOS

Our shop's policy to handle Distributed Denial of Service is that every 10 seconds a timer checks the number of requests that each client (with specific IP address) sent to the server. If the number of requests exceeded 100, then it automatically disconnects the client from the server. Therefore,that client's IP will go to the black list and he cannot send any more requests to the server. This policy was implemented in server(package) -> Server (class).

There is an inner class named "*CheckIps*" which extends TimerTask. At first for each client a timer is scheduled to call this class's run method every 10 seconds.