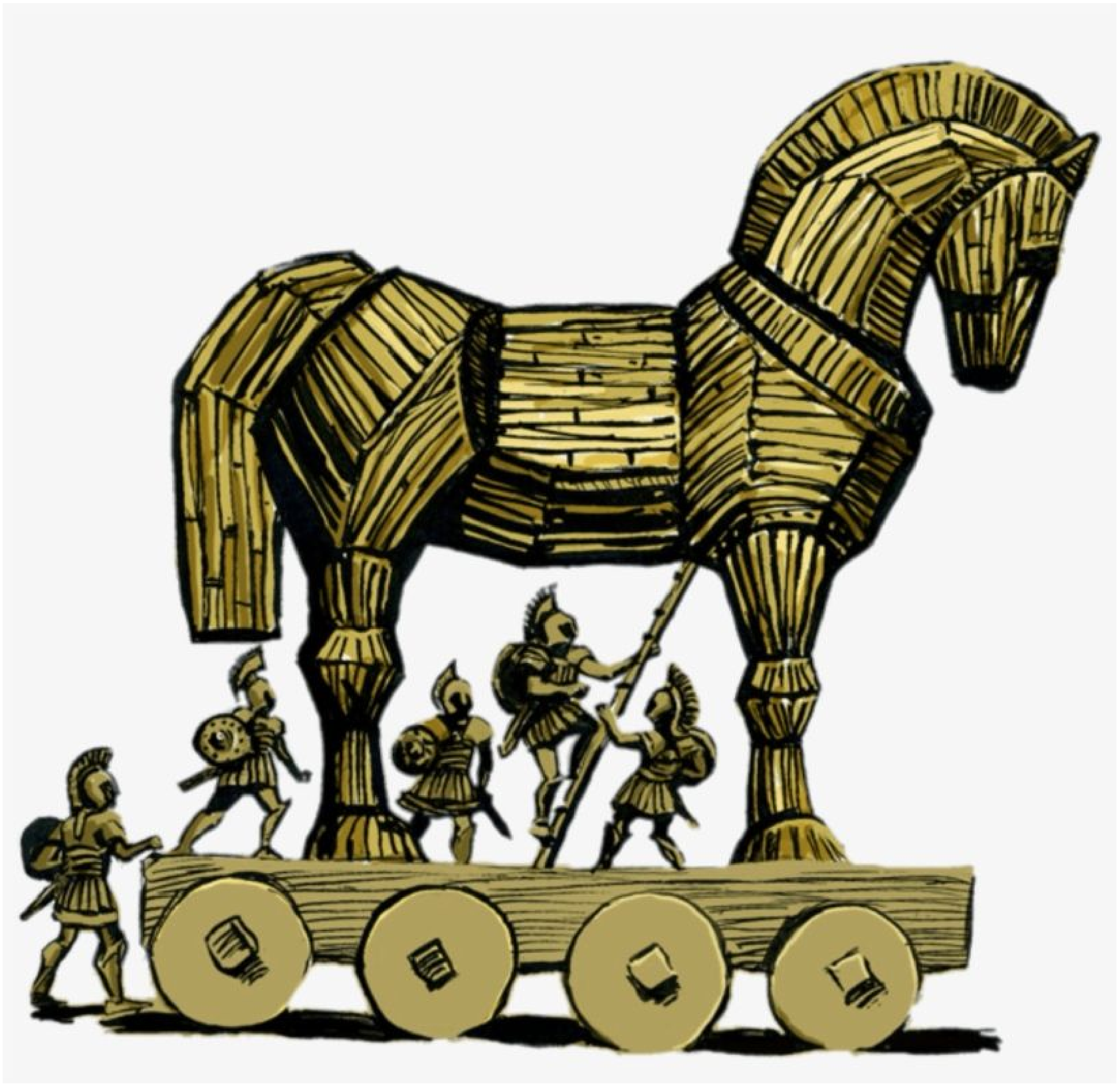
Software Quality Assignment



Bram Gussekloo (0966476) & Matthijs Booman (0964703), INF2D
Teachers: Ahmad Omar, Babak Basharirad
Course code: INFSWQ01-A
Periode: OP4

HOGESCHOOL ROTTERDAM

Table of contents

Preface

The file that you have to run is the `System.py` file. After this you will be using our system. You will have a few options from there but the most important one is the `help` option. This will tell you what you can do. Anytime you are not sure what to do you can use that command.

In the program we used multiple security measures which are shown in this document. We will explain these by going through every command that is available in the program.

The application uses an SQLite-database and has the following tables with the following columns in it:

- Users
  - id - INTEGER PRIMARY KEY UNIQUE AUTOINCREMENT
  - username - VARCHAR
  - password - VARCHAR
  - role - VARCHAR
- Client
  - id - INTEGER PRIMARY KEY UNIQUE AUTOINCREMENT
  - full_name - VARCHAR
  - address_id - INTEGER REFERENCES Address
  - email_address - VARCHAR
  - mobile_phone - VARCHAR
- Address
  - id - INTEGER PRIMARY KEY UNIQUE AUTOINCREMENT
  - street_house_number - VARCHAR
  - zipcode - VARCHAR
  - city - VARCHAR
- Log
  - id - INTEGER PRIMARY KEY UNIQUE AUTOINCREMENT
  - time - DATETIME
  - user_name - VARCHAR
  - log_text - TEXT
  - priority - INTEGER

System explanation

login

When you try to log in, all Null-bytes get removed immediately. We check if the username exists by executing the sql command “ `SELECT * FROM Users WHERE username = ?` ” which is a prepared statement. The username is gotten from the user that is trying to log in, but because we are using prepared statements nothing will happen if the user is trying to use SQL injections to gain access (This is used in the whole application and will not be mentioned again).

After the username is gotten the password is returned from the database to the program and then the password is going to be matched. The password in the database is hashed by sha256 with a salt. After this is checked the User object is created and stored in memory, which tells the system which commands the user is allowed to use and what information corresponds to the user.

If the user tries to log in with a wrong username or password multiple times, the user will be kicked and the system will exit (as a security measure).

When the user tries to execute a command the authorization process checks if the user is logged in and if said user is allowed to use the command based on the user's role. The execution of the command is also based on the user’s role.

If the user is not allowed to use the tried command a counter will go up. If the counter passes 4 not permitted actions the system will kick and ban the user, or kick and close the system if the user isn’t logged in.

logout

The User object is deleted from memory which causes the user to lose access to its allowed commands. Command authorization is checked before the user gains access to said command based on the user role, logging out will set the user to ‘None’ which means it loses access to most commands.

new-user

This command is executed differently based on what role the logged in user has, the super administrator is able to create a new administrator while an administrator is able to create a new advisor. The command will ask what the username of the new account should be. This will make sure that the username is at least 5 characters long and no longer than 20 characters. It also has to start with a letter and must contain only letters, numbers and certain characters. The way we achieved this is by whitelisting these characters with regular expression. The Null-bytes also get removed. When you type something that does not meet these requirements there will be a message which says what it has to comply with, this will be repeated until it is in accordance with the requirements.

The command will then ask for a new password that has to be created. This works almost the same as the username, only the restrictions are less heavy. You can put in more characters and more symbols, only you need to have at least 8 characters as a password. After the validation the password is hashed by sha256 with a salt and then sent to the database along with the username by using prepared statements.

add-client

The administrator is the only type of user allowed to add a client to the system. The command will ask for the Full Name, Address (Street and House number, Zip Code, City), Email Address and mobile phone number. Null-bytes get removed for every input, and the input gets validated with regular expressions according to the restrictions set by the assignment. User gets asked to correct their input and told the restrictions when it doesn't pass the input validation, this will repeat until the input is in accordance with the restrictions.

The newly created client will be saved in the database using prepared statements.

get-logs

The Super Administrator is the only one that is allowed to use this command. This command is used to get all the logs that are in the database. These are categorized by priority. As the Super administrator you can choose from which priority you want to see the logs. There are three priorities. 1 through 3.

The lowest priority is 3 and will contain logs about people logging in, logging out, creating users. These will be displayed white in the console.

The middle priority is used as a warning to the Super Administrator, these are also displayed yellow in the console. These will contain logs about users that are trying to log in with wrong credentials or people that are trying to create passwords that do not comply with the regular expression.

The last priority is the highest one. This will contain all the bad stuff. So the people that are trying to use commands that they are not allowed to use or logging in while already being logged in. Mostly this is used to indicate that there is something wrong with the application or that there is someone trying to break into the application.

help

The help command can be executed by every user, even if the user isn’t logged in. When the user isn’t logged in this command will tell the user to login to get more help information, and how to stop/exit the system. If the user is logged in the response will be based on the user’s role. A list of available commands will be displayed under each other.

stop / exit

The ‘`stop`’ and ‘`exit`’ commands are basically the same command, it’ll close the system. It doesn’t matter if the user is logged in or not, the main loop will break and everything not in the database will not be saved.

Explanation of the criterias

C1 Authentication for users is properly implemented.

Users are able to login using a username and password, all user data is saved and read from an SQLite file which isn't readable by text editors. When a user gives a wrong username or password as input they will receive an error message that explains that either the username or password is wrong. When a user fails to login after several attempts they'll get kicked out of the system and the system exits as a security measure. We think we have achieved L3 satisfactory for this criteria.

C2 Users access level are implemented.

With our program we think we got to L3. We implemented authentication by adding a column to our SQLite database which is called "role". This column tells the system on what authorization level the user is. This variable is used throughout the whole application. We base the authorization and execution of the commands on if the user is logged in and the role that the user has.

C3 All inputs are properly validated.

Input validation is fully implemented using regular expressions that follow the restrictions listed in the assignment. All Null-bytes are removed from every input. In order to implement proper input validation we retrieved the input, removed all Null-bytes and sent it off to validation functions that use regular expressions to check range, length & formatting of the input. We think that we've achieved L3 because we think there is nothing an attacker could abuse using current implementations.

C4 Invalid inputs are properly handled.

We think we got to L3 with this one. In the program we created custom exceptions, which means that we can somewhat accurately say what the problem is. For example: If a user logs in but the username is not found, the system will throw an usernameException. This exception tells the login function that the username does not exist in the database. We created a few of them so that our exceptions are tidy. We hide the type of error to the user, they'll only see a message like: "Wrong username or password!"

C5 Suspicious activities are logged.

We implemented logging and made sure that every function that is called will log their actions. This means that we log an action if the user: logs in, creates a new user, creates a new client, uses a not permitted command, fails to log in, gets kicked or gets banned, etc. We categorized the logs by using a column called “priority”. This column will tell the severity of the log. A one is high and will be red. In the logs we put the function that it is called in at the front of the log text. We think we got to L2 on this one, but only because we are not sure what the good practices are and that we don’t know if we got the categorization completely right.