

**Description of the Attack:**

A brute force attack is an attack where the intruder uses a dictionary of combinations to attack a target and analyze the response until he succeeds. The attacker tries these passwords one by one for authentication. If this dictionary contains the correct password, the attacker will succeed. Success depends on the number of possibilities. If it is larger, it will take more time, but there is a better probability of success.  However, this traditional technique will take longer when the password is long enough. These attacks can take several minutes to several hours or several years, depending on the system used (in terms of storage and processing) and the length of password.

In our case, for simplicity, we restricted the password to consist of 4 digits only. These can digits are numbers ranging from 0 to 9.

**Algorithm of the Attack tool:**

String[] pass = ReadMyPasswords(); this function will read all the passwords in the dictionary and store them into an array, so we iterate on each of them.

String infoToServer = (command + "," + info1 + "," +pass[0]); we send to the server’s socket that we want to login with a certain username and we start by trying the first password.

cc.SendStringToServer(infoToServer);

Thread.sleep(100);

if(cc.din.readUTF().equals("Incorrect password please try again")){ if the client received an incorrect password, the tool begins and sends multiple requests to authenticate by using each password available in the dictionary, until it succeeds.

int j = 1;

while(j<9999) {

String authentication = (command + ","+ info1 + "," + pass[j]);

cc.SendStringToServer(authentication);

Thread.sleep(100);

if(cc.din.readUTF().equals("Incorrect password please try again")) {

j++;}

else if (j == 9999) {

System.out.println("Password not in dictionary");

break;}

else {

System.out.println("Login Succeeded");

break;} } }

**Contributors of the Attack tool:**

Ralph and Alain teamed up to work the tool.

**Description of The Detection Tool:**

The detection tool works by checking if a user has tried to input their password, incorrectly, more than ten times in less than six seconds.

Initially, the variable *din* opens an input stream on the socket and retrieves the data coming through that socket.

The *password* string is a returned string of the readUTF() function. The readUTF() function reads in a string, from the socket, that has been encoded using a modified UTF-8 format. This original string is then decoded from the UTF and returned as a string.

**Algorithm of The Detection Tool:**

initialized integer counter : this variable is used to count the number of incorrect password attempts being made.

initialized string password : this variable is used to store the password that has been retrieved from the incoming data of the socket (this is the password that the user has inputted)

initialized string word : this variable is used for the input string from the user

get time at start and get time at end : the function currentTimeMillis() returns the current time in milliseconds. In other words, this function was used to keep track of how long a user was trying incorrect passwords for. Start gets the time at the very beginning, when the first password is inputted. End gets the time after the third attempt, at inputting a password, has been made.

if(password != word) : if *password* is not equal to the variable *word*

counter ++ //then counter is incremented by 1

if (counter == 10 AND end-start < 6 seconds) : if these conditions are met, then a hacker is trying to crack the password.

print (attack) : Thus, an error message is outputted.

socket.close(); The client will be disconnected to prevent him from cracking.

else //if the above *if* condition is not met, then a client is not trying to get into the system.

counter = 0 : Therefore, counter is reset to 0

get new time again : *start* is reset to the current time at this point

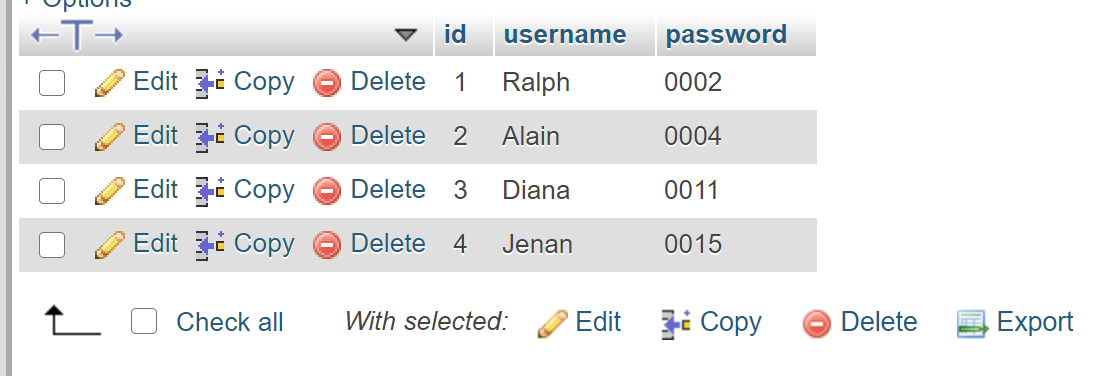
**Contributors of The Detection Tool:**

Diana and Jenan teamed up to work on the detection tool. Both Diana and Jenan met over Zoom to discuss the algorithm and write up a pseudocode for the detection algorithm.

**Implementation****:** There are 6 classes in our project: Client, ClientConnection, Server, Server Connection, MySqlConnection, SqlQueries. This model will allow us to simulate the attack as if a hacker is trying to guess a google account’s password by connecting to their server and trying to crack the password that is stored in a database.

Client class contains everything the hacker needs to login, activate his tool, whereas ClientConnection class handles the input, and output stream for every client. The server is multi-threaded and ServerConnection handles the server’s connections. MySqlConnection connects to the database with the help of WAMP Server Application (It should be downloaded and active. Open any browser and type localhost/phpmyadmin and you can access the database with **username = root** and **password is blank.)**

RUN: Server should be on first, then Client in any Java IDE. (We used Eclipse) Once the Client is connected it asks the user if he/she wants to login. The client responds by “login” in the console. Then it requests a username. We will assume that we know the username of the victim and we write it down in the console. The attack tool is activated and will start trying different passwords, until the detection tool will prevent the hacker from continuing and log him out. The attacker has a chance of success with 10/10000 (10,000 possibilities for four numbers) since our detection prevents him from trying more than 10 requests in a short amount of time.

Figure1: Database Table showing the different usernames and passwords stored.

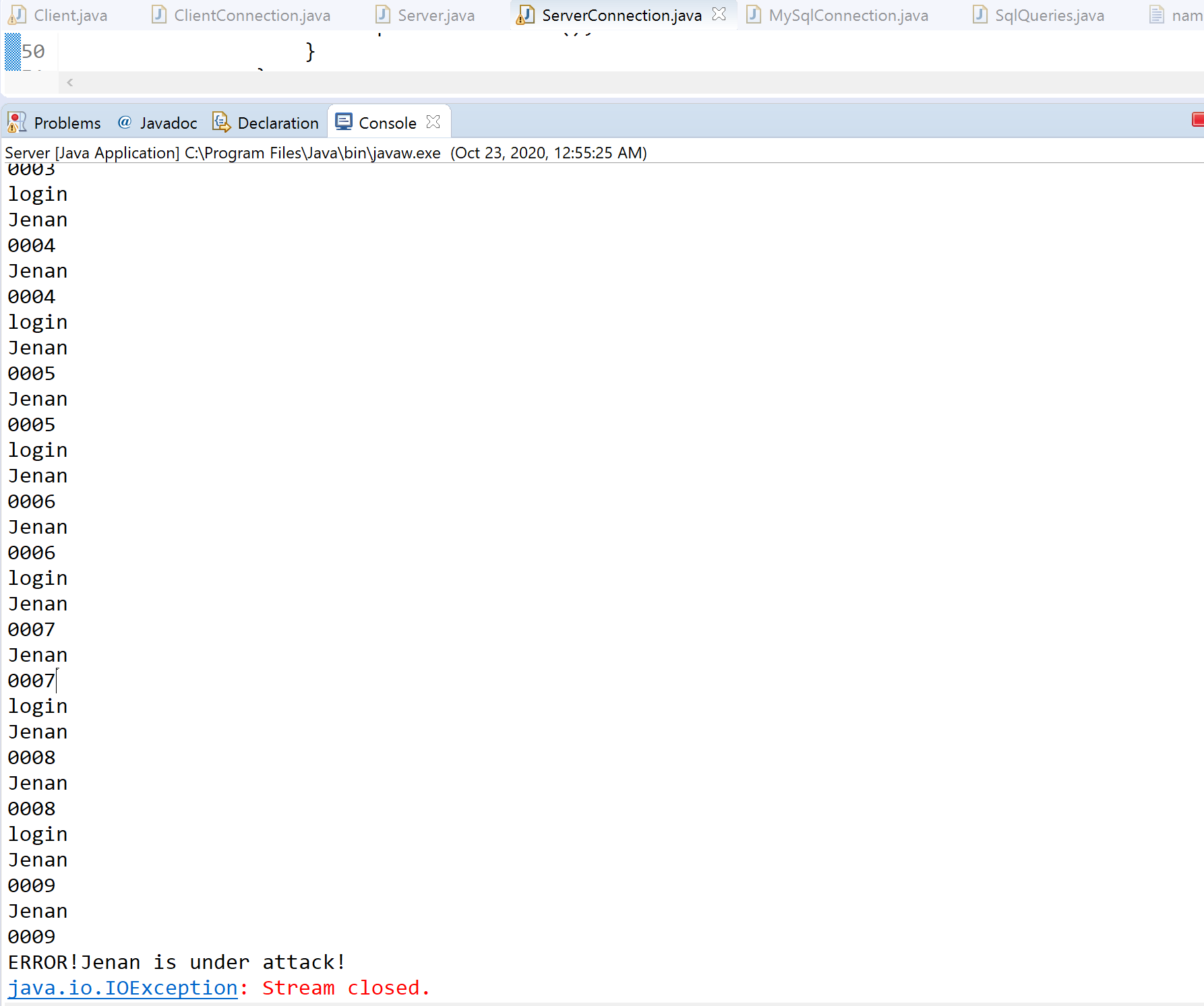


Figure2: Console output when the attacker tried his tool to breach Jenan’s account. He was logged out after performing malicious requests.

His socket is closed.