Zoo Authentication System

Aaron Guyett

Southern New Hampshire University

IT-145 Foundations in Application Development

Professor John C. Fleming

April 20, 2018

Abstract

A system is not protected without some sort of barrier. Communication controls such as firewalls and encryption processes are necessary, but a system should always have access controls in place. An efficient authenticator will protect the system from those who should not have access. The system should be able to identify the user class of each user logging on. This will limit the privilege that each user has, to the minimum amount. These actions are necessary because of the amount of damage that can be done to a system without protection. Information is a currency that cannot be replaced once it has been discovered. Systems like this are vital, and the following documentation explains the process behind making one.

Zoo Authentication System

The more controls that are present to access a system, the less likely a breach will be. Access controls restrict unauthorized individuals from using information resources (Rainer, et al., 2017). An authentication system is an access control because it limits those who can gain access to a system. It then assigns each user the correct amount of privilege. The attached program will allow users to attempt to login three times and then exit the program if they are unsuccessful. If a user successfully logins in by using a username and password that has been verified by the credentials file, then he or she will be welcomed by a message from an outside file based on the associated user grouping. The three user groups are admin, zookeeper, and veterinarian. The program must be able to check the provided file for credentials and determine if the user deserves entry to the system.

# Process

The program requires user input at multiple stages. Using that as a starting point, the program was broken down into sections. The class was defined and the program was created, then a main class was defined. The method created inside the main class refers to the screen that a user will see when attempting to login. The first screen of the program should be labeled, and it is: “Login”. The user is prompted to enter his or her username. The user is then prompted for his or her password. This information should be converted to MD5 by the program. To do this, the information received by the user was stored. It was then converted using the provided code. After the information is converted to MD5, the provided credentials need to be checked. If the information matches the credentials provided in the associated file, the user can be provided access to the system.

The program needed to check the file by reading through it and checking to see if the username and password matched one of the ones in credentials.txt. A loop was made to check each line of credentials.txt with a file reader. As the program read the file it split each line and converted it into an array. This made determining if the password matched the username easier because the program just checks an array item for each. If a user successfully logs in, he or she needed to be presented with the associated greeting, which was in a separate file as well for each of the three options. The credentials file labeled each user with the appropriate type. This mean that I could use another reader to read the associated file type when a user successfully logs in. A zookeeper account would then see the contents of zookeeper.txt when he or she logs in.

The next part of the program dealt with users who did not use a correct username and password. By defining the number of attempts earlier in the project and decreasing them by one each attempt, the variable could be used to control the loops. Since it starts at 3 and decreases by 1 each attempt, the program performs different actions based on the value of “attempts123”. When “attempts123” is equal to 0, the program is set to kick the user out of the system and print a message letting the user know that he or she has reached the maximum number of attempts. Trying to figure out how to get the program to know a user had been successful and to put them in the “successfully logged in” part of the program was difficult. After much thought, it was clear that another variable had to be proposed and put in the prior part of the program that would label a user as logged in.

The variable “loggedIn” was created to show that a user had successfully logged in with the correct credentials. An if statement is used to tell the program what to do when loggedIn is equal to 1, and that is to show the systemScreen method. The loggedIn variable is set to 1 in the prior loop when a user successfully logs in. The systemScreen method will display the logged in screen to a user after any attempt under 4. If a user has not reached the maximum number of attempts, the else statement is set to let the user know how many attempts he or she has left if it is greater than 0.

The throws that were used needed a catch block. The exception is traced and printed using the printStackTrace method. This was used to help debug the program. The systemScreen method was used but not defined. The system had to have the option to be logged out of. Using a combination of keys (9z) a user is given the option to logout with a prompt after the associated user group text is shown. If the user enters “9z” he or she will be brought back to the login screen by the loginScreen method. The program covers all the necessary parameters as determined by the rubric and the methods invoked are named appropriately. The Oracle Documentation and Chegg were used as resources throughout the project to help discover things such as the “BufferedReader” and the “NoSuchAlgorithmException”.

# Pseudocode

The pseudocode in *Figure 1* is a description of how the Java program was created. It allows users to log in to a system. They will be authenticated by a file on hand. The users must be prompted to login to a system. This system must have the usernames and passwords in a separate file for authentication to occur. The program will check the users’ inputs with the file and verify that they are in the system. Once logged in, the user’s role will be displayed to him or her. The user will have an option to log out. This will bring the user to the credentials screen again. If a user cannot successfully login after three attempts, the program will exit. The program must prompt the user for input, validate the input from an outside file source, contain a lockout mechanism for users who cannot login after three attempts, and allow users who have logged in identify their roles and log out.

The program differs from the pseudocode slightly. The name of the program is different and the finished program contains more detail than the pseudocode. For example, the imported libraries are not included, the name of the methods that were created are not mentioned and the way that the program parses the files is not included. There is no mention in the pseudocode of the program storing any of the information for a short amount of time, and the way that the users’ amount of login attempts decreases is not included. A lot of these things are things that were hard to visualize and were small problems that were worked through during the project. The created methods fit gaps that needed to be filled so that the program would work. The names of these methods were created to fit what they do. Most of them were not mentioned in the pseudocode. The pseudocode does document while and if loops but does not mention the do and try statements and does not include any way to manage exceptions. This information was learned by doing research while working on the project.

# Error and Solution Documentation

The biggest problem that occurred when developing the program was having the program recognize the difference between a user that had logged in and a user that was just starting the program or one that had been logged out. The logic behind the discovery of the solution came from determining that a user had to be at the same screen at multiple times. A user had to be at the same screen-prompted for a username-when he or she was waiting to login and when he or she starts the program. This meant that there had to be something in common between those two occurrences, which led to the method “loginScreen” being created and placed appropriately. Other errors required exceptions to be handled correctly. Debugging led to the discovery of the two exceptions used and outside research was used to utilize the catch statements. This prevented the program from crashing at certain points. The program took a long time to develop because it had to be broken down into parts utilizing knowledge that came from less than seven weeks of learning. It became clear that a developer will always need to access and use outside resources to learn more about a programming language and that there is always more to learn.

References

Liang, K. (2018, April 19). Chegg.com. Retrieved April 20, 2018, from https://www.chegg.com/study/qa

Rainer, R. K., Prince, B., Sanchez-Rodriguez, C., & Hogeterp, I. S. (2017). Introduction to information systems: supporting and transforming business. Toronto: Wiley.

Trail: Learning the Java Language. (2017). Retrieved April 20, 2018, from https://docs.oracle.com/javase/tutorial/java/index.html

Figures



Figure 1. This figure is the pseudocode for the authentication system. It differs from the final output in terms of detail, but the overarching structure remains similar.