Homework 2

Java-based Login Application with added security controls

Student: Patrick Walsh

School: University of Maryland Global Campus

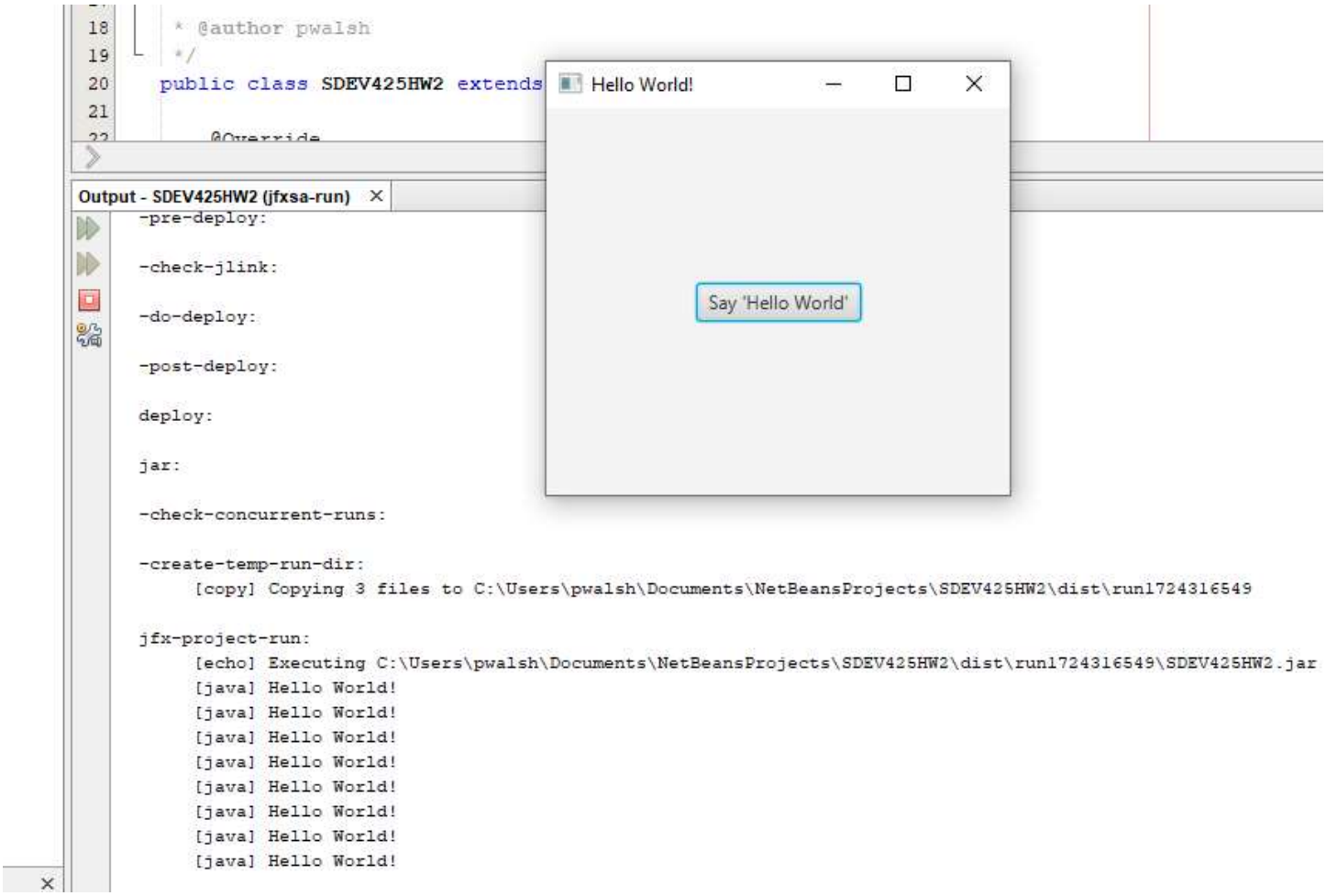
Course: SDEV 425 6980

Date: 7/13/2021

Professor: Dr. Nicholas Duchon

Part I: Review and Understand the Sample JavaFXapplication

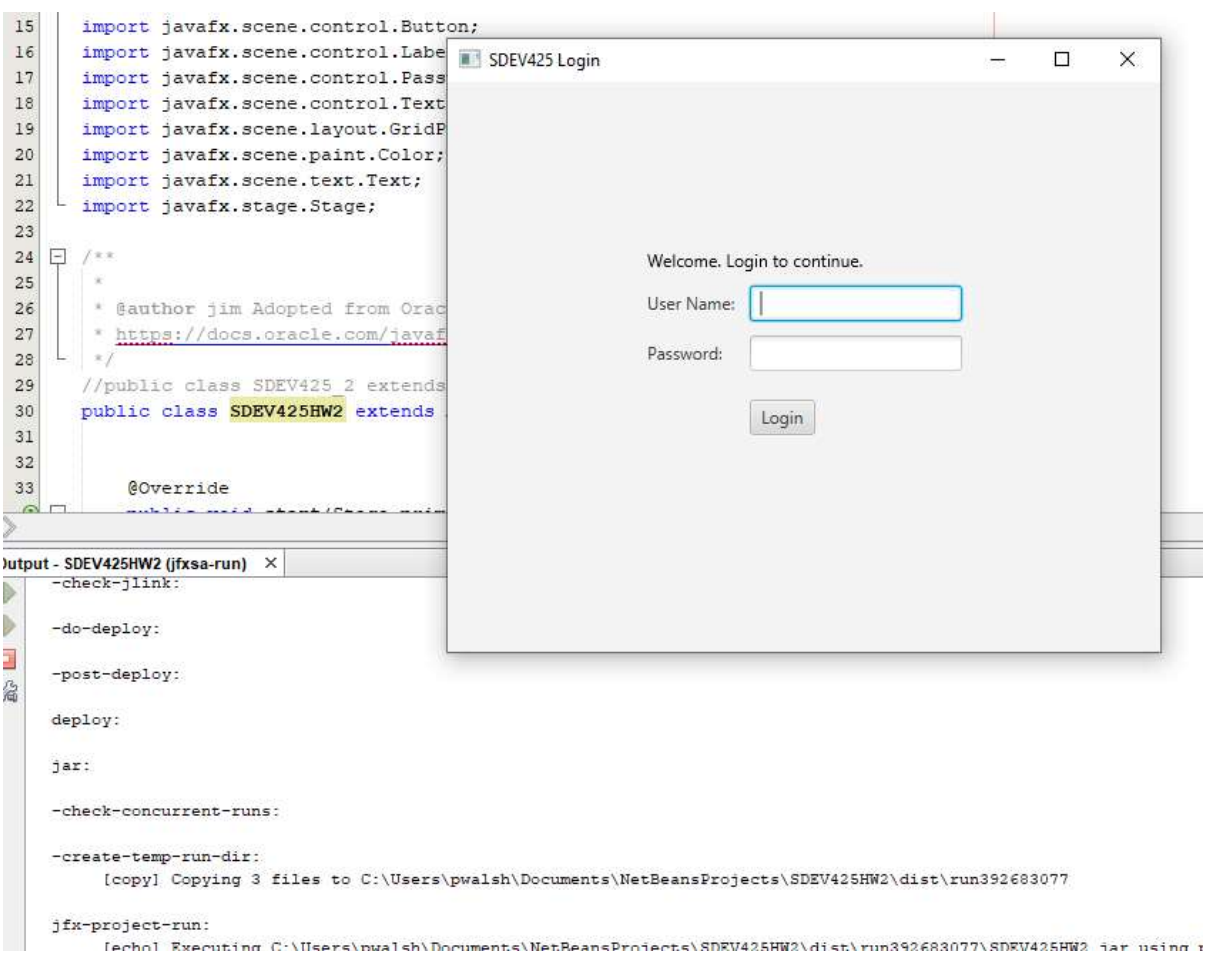
Below is a successful run of the sample JavaFX ‘Hello world’ program:



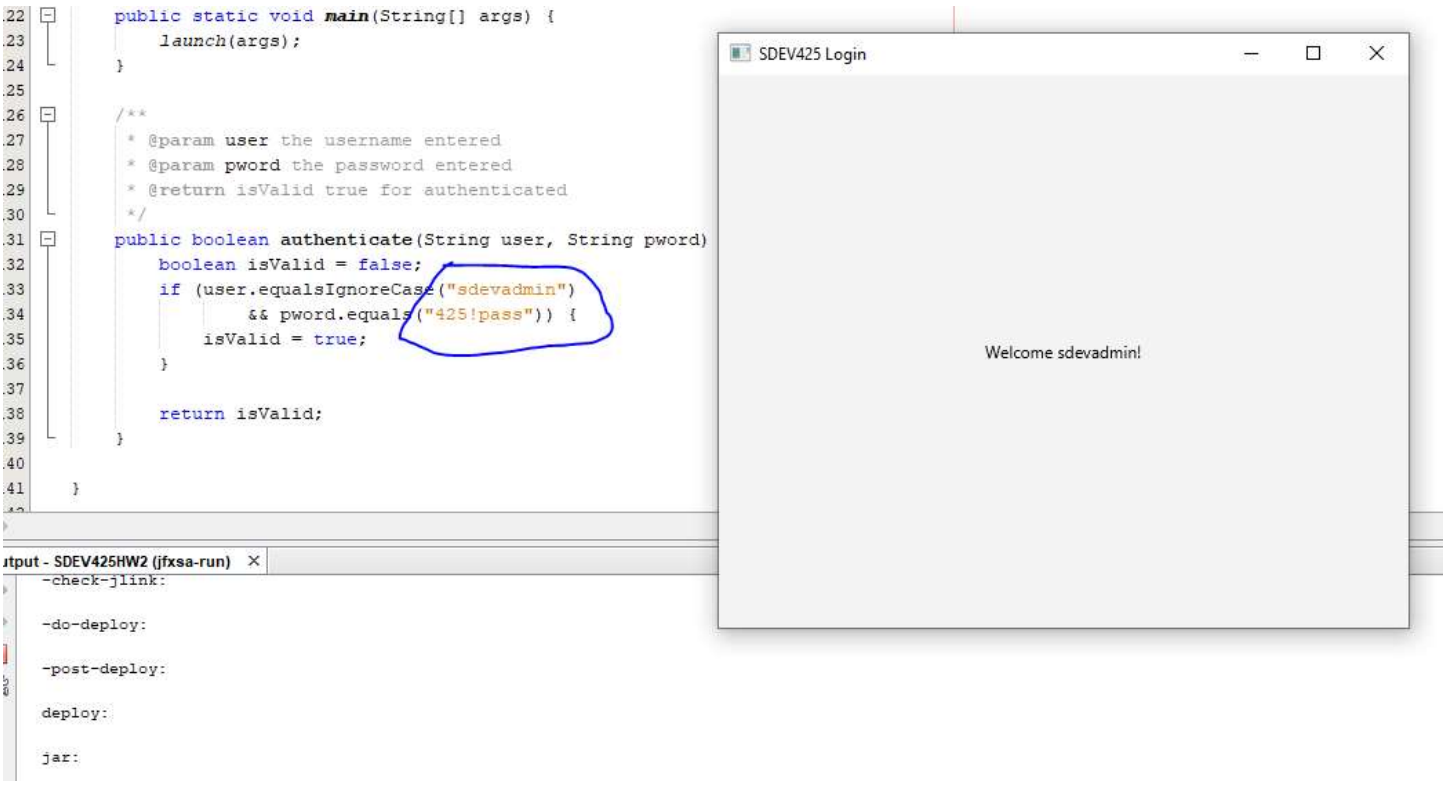
To get the provided java file to run in the project, I copied and pasted the provided SDEV425_2.java code into the project and changed the package name and class names:

```java
 */
//package sdev425_2;  // original code
package sdev425hw2;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;

/**
 *
 * @author jim Adopted from Oracle's Login Tutorial Application
 * https://docs.oracle.com/javafx/2/get_started/form.htm
 */
//public class SDEV425_2 extends Application { // original code
public class SDEV425HW2 extends Application {
```

The program then successfully ran in my project, showing the welcome login screen:



I successfully logged into the application using the username and password found in the authenticate() method:



Overview of program functionality

The login application consists of a main method, an event handler method called handle(), a Boolean password checking method called authenticate(), and a GUI generator method called start(). The start() method builds out the main GUI generated when the program is launched. The GUI is generated with a State called primaryStage (line 34), and the Stage is given a title (line 35). The Stage is then constructed with a GridPane called grid (line 37) and its properties are set (lines 40, 43, and 44). A title for the grid is added (line 47 and 49), then a Label called userName is added (lines 52 and 54). Next, a TextField box for the user to enter the username is added to the grid (lines 57 and 59). See screen shot below for details:

```
29  //public class SDEV425_2 extends Application { // original code
30  public class SDEV425HW2 extends Application {
31
32
33      @Override
34      public void start(Stage primaryStage) {
35          primaryStage.setTitle("SDEV425 Login");
36          // Grid Pane divides your window into grids
37          GridPane grid = new GridPane();
38          // Align to Center
39          // Note Position is geometric object for alignment
40          grid.setAlignment(Pos.CENTER);
41          // Set gap between the components
42          // Larger numbers mean bigger spaces
43          grid.setHgap(10);
44          grid.setVgap(10);
45
46          // Create some text to place in the scene
47          Text scenetitle = new Text("Welcome. Login to continue.");
48          // Add text to grid 0,0 span 2 columns, 1 row
49          grid.add(scenetitle, 0, 0, 2, 1);
50
51          // Create Label
52          Label userName = new Label("User Name:");
53          // Add label to grid 0,1
54          grid.add(userName, 0, 1);
55
56          // Create Textfield
57          TextField userTextField = new TextField();
58          // Add textfield to grid 1,1
59          grid.add(userTextField, 1, 1);
60
```

Next, another Label is added for the password (lines 62 and 64) and a PasswordField text box is added so that the user can enter a password (lines 67 and 69). Next, a Button called btn is added to the grid (lines 72 and 74) and the Button is given an EventHandler so that it responds when the user clicks on it (line 80). Finally, a Text called actiontarget is added (lines 76 and 77) as a message that will pop up on the GUI if the user enters the wrong username or password. See screen shot below for details:

```
60
61          // Create Label
62          Label pw = new Label("Password:");
63          // Add label to grid 0,2
64          grid.add(pw, 0, 2);
65
66          // Create Passwordfield
67          PasswordField pwBox = new PasswordField();
68          // Add Password field to grid 1,2
69          grid.add(pwBox, 1, 2);
70
71          // Create Login Button
72          Button btn = new Button("Login");
73          // Add button to grid 1,4
74          grid.add(btn, 1, 4);
75
76          final Text actiontarget = new Text();
77          grid.add(actiontarget, 1, 6);
78
79          // Set the Action when button is clicked
            btn.setOnAction(new EventHandler<ActionEvent>() {
81
```

In the handle() method, the program takes an ActionEvent in the form of the user clicking the login Button. The handle() method calls the authenticate() method and passes in the userName and password TextFields as arguments (line 85). If the authenticate() method determines that the userName and password are correct, the handle() method changes the grid to show a welcome screen stating that the user has successfully logged in (lines 87-102). If the userName and/or password are incorrect, the Text actiontarget is set to a display message letting the user know that the userName and/or password is incorrect (lines 104-108). See screen shot below for details:

```
81
82          @Override
83          public void handle(ActionEvent e) {
84              // Authenticate the user
85              boolean isValid = authenticate(userTextField.getText(), pwBox.getText());
86              // If valid clear the grid and Welcome the user
87              if (isValid) {
88                  grid.setVisible(false);
89                  GridPane grid2 = new GridPane();
90                  // Align to Center
91                  // Note Position is geometric object for alignment
92                  grid2.setAlignment(Pos.CENTER);
93                   // Set gap between the components
94                  // Larger numbers mean bigger spaces
95                  grid2.setHgap(10);
96                  grid2.setVgap(10);
97                  Text scenetitle = new Text("Welcome " + userTextField.getText() + "!");
98                  // Add text to grid 0,0 span 2 columns, 1 row
99                  grid2.add(scenetitle, 0, 0, 2, 1);
100                 Scene scene = new Scene(grid2, 500, 400);
101                 primaryStage.setScene(scene);
102                 primaryStage.show();
103                // If Invalid Ask user to try again
104             } else {
105                 final Text actiontarget = new Text();
106                 grid.add(actiontarget, 1, 6);
107                 actiontarget.setFill(Color.FIREBRICK);
108                 actiontarget.setText("Please try again.");
109             }
```

Finally, the authenticate() method, as mentioned earlier, checks to see if the userName and password are correct. The method takes in two arguments, the userName and password TextFields (line 131) and does a Boolean check to see if they match up to the hard-coded values for userName and password (lines 132-136). If they match, the Boolean is returned as true and the Boolean is returned in the handle() method call to let the user login. See screen shot below for details:

```
130         */
131     public boolean authenticate(String user, String pword) {
132         boolean isValid = false;
133         if (user.equalsIgnoreCase("sdevadmin")
134                 && pword.equals("425!pass")) {
135             isValid = true;
136         }
137
138         return isValid;
139     }
```

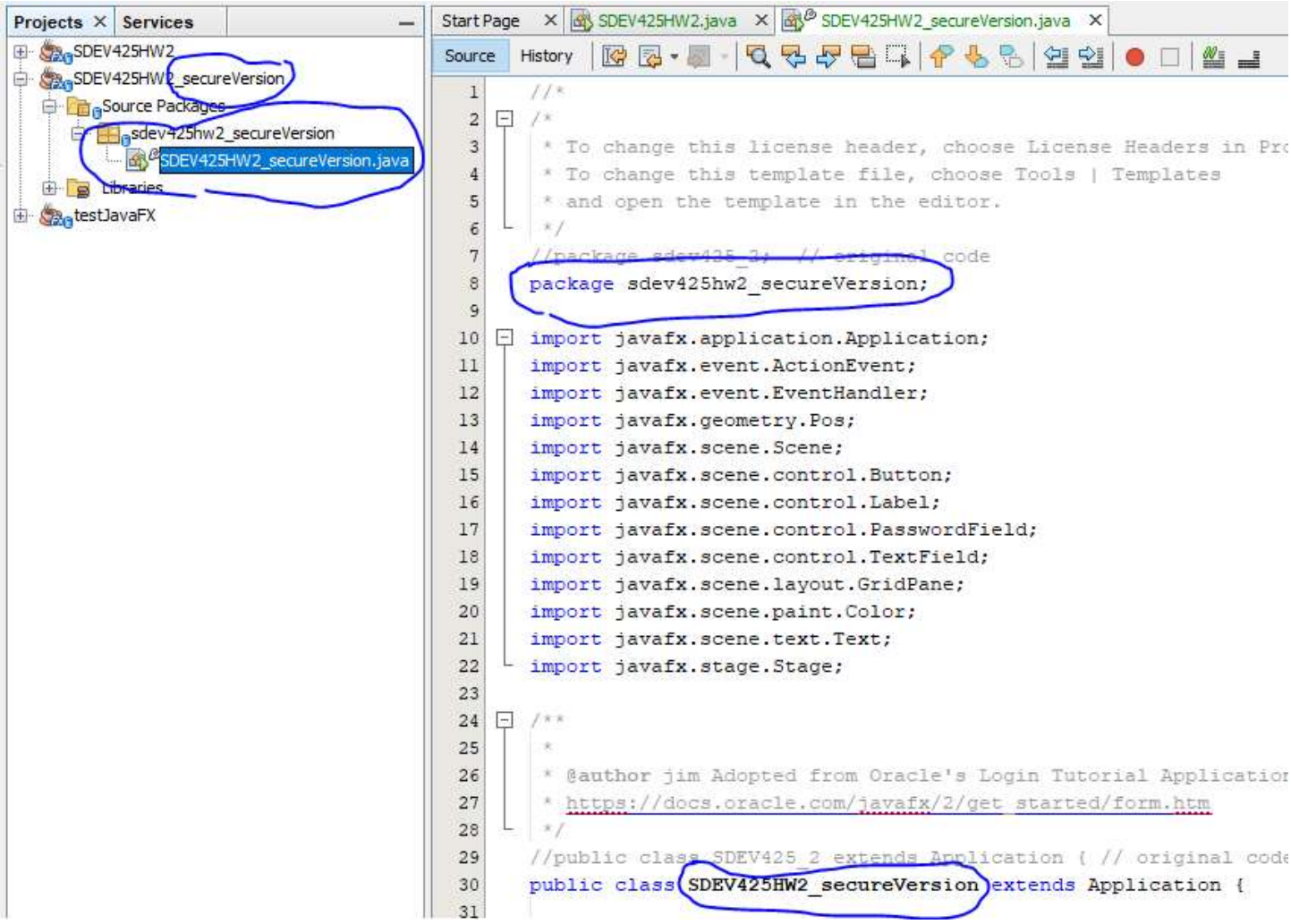
Part II: Apply select NIST low-impact security controls to the JavaFXLogin application.

The following security controls should be applied to the application (check the NIST Security Controls Database for details, description and guidance for each control):

- AC-7-UNSUCCESSFUL LOGON ATTEMPTS
- AC-8 -SYSTEM USE NOTIFICATION
- AU-3 -CONTENT OF AUDIT RECORDS
- AU-8 -TIME STAMPS
- IA-2(1) IDENTIFICATION AND AUTHENTICATION (ORGANIZATIONAL USERS) | NETWORK ACCESS TO PRIVILEGED ACCOUNTS (Note this is an enhancement of an existing low-impact security control)

Select one additional low-impact security control and implement it. This can be an enhancement or a required low-impact security control. Selecting a control that provides documentation as opposed to code changes is also acceptable and encouraged.

The first step I took was to copy the SDEV425HW2 project and all its contents and make a second version of this project called SDEV425HW2_secureVersion. All the changes below were done to this second version, including the name changes for package and class. See below:



AC-7-UNSUCCESSFUL LOGON ATTEMPTS

This security control specifies that there should be a limit on the number of consecutive failed login attempts by a user (NIST, 2021a). As a security precaution to prevent brute force or other malicious login attempts, organizations should set a limit on this activity (NIST, 2021a). If the number of failed consecutive login attempts is reached, the system should respond such as by locking the account temporarily or until an administrator unlocks the account, setting a delay to prevent the user from attempting another login for a certain period of time, notifying system administrators of the security event, or other appropriate actions (NIST, 2021a).

To implement this security control, I added a new method called lockout() that takes in two arguments: a loginAttempt int value and a loginLimit int value. If the consecutive login attempt reaches the attempt limit, the method returns a Boolean value locked as true.

```
170         * @param login the login attempt number
171         * @param limit the login attempt limit
172         * @return locked true to lock out account
173         */
174        public boolean lockout(int login, int limit){
175            boolean locked = false;
176            if (login >= limit){
177                locked = true;  // returns locked as true if maximum login attempts have been reached
178            }
179
180            return locked;
181        }
182
```

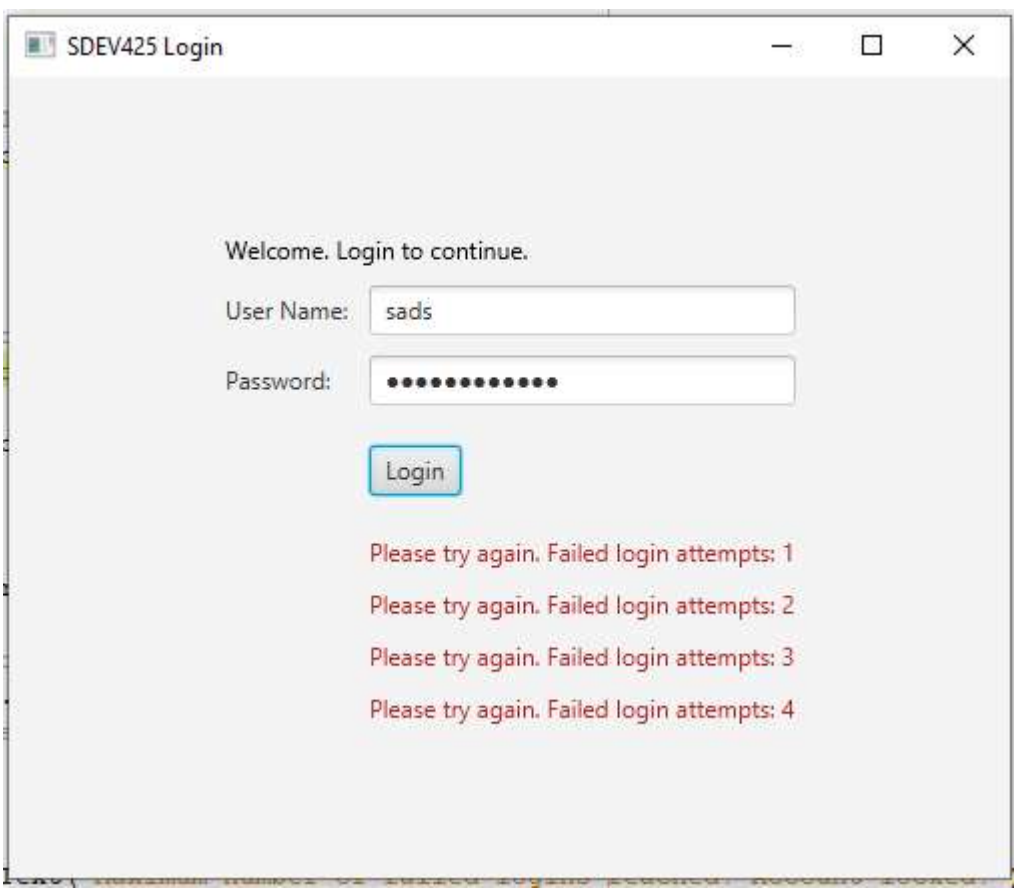
The loginAttempts int value is initially set to 0 and the loginLimit is set to whatever the administrators decide, in this case 5.

```
32
33     //public class SDEV425_2 extends Application { // original code
34     public class SDEV425HW2_secureVersion extends Application {
35
36         // initialize and set the login attempts to 0
37         int loginAttempts = 0;
38         // specify maximum number of failed logins allowed
39         int loginLimit = 5;
40
41         @Override
           public void start(Stage primaryStage) {
```

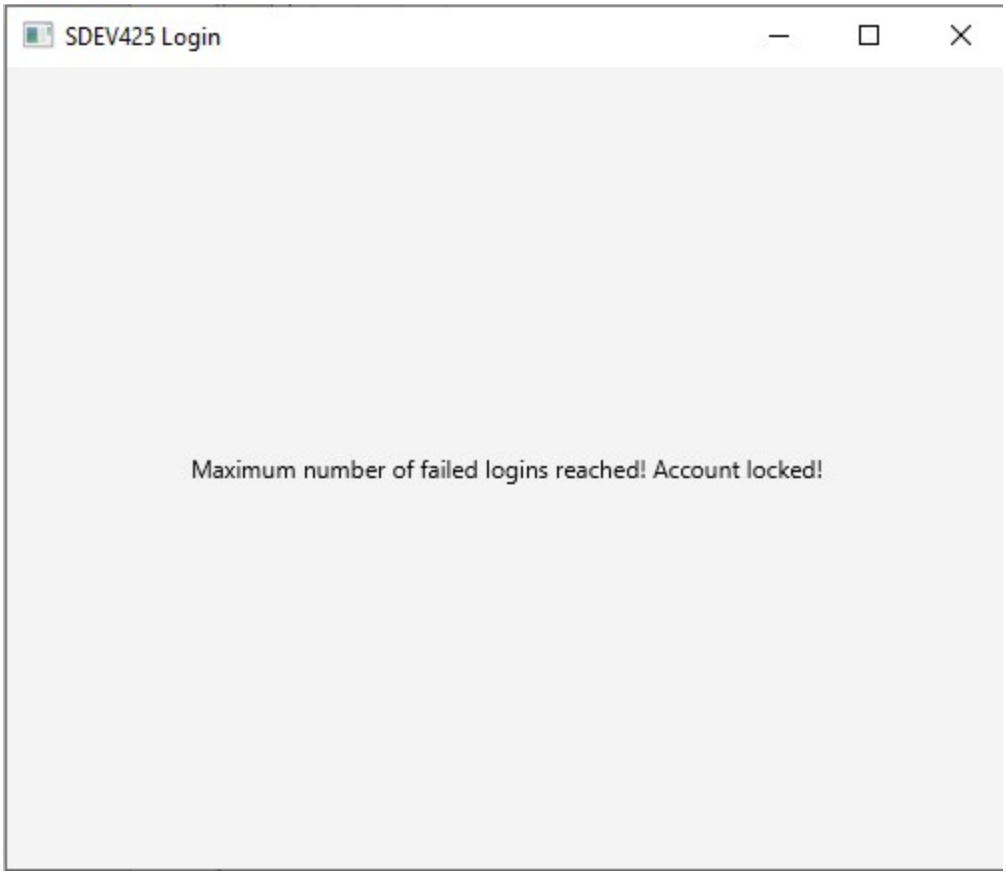
Each time the user clicks the login Button, the handle() method increases the loginAttempts by 1 and calls the locked() method to see if the number of login attempts has reached the specified limit. If the limit has been reached, the program creates a new GridPane called grid3 and gives an error message letting the user know they have reached the maximum number of attempts reached.

```java
@Override
public void handle(ActionEvent e) {

    loginAttempts += 1;  // increase loginAttempts by 1
    boolean locked = lockout(loginAttempts, loginLimit);
    // Authenticate the user
    boolean isValid = authenticate(userTextField.getText(), pwBox.getText());
    // If valid clear the grid and Welcome the user

    if (locked){
        GridPane grid3 = new GridPane();
        // Align to Center
        // Note Position is geometric object for alignment
        grid3.setAlignment(Pos.CENTER);
         // Set gap between the components
        // Larger numbers mean bigger spaces
        grid3.setHgap(10);
        grid3.setVgap(10);
        Text scenetitle = new Text("Maximum number of failed logins reached! Account locked!");
        // Add text to grid 0,0 span 2 columns, 1 row
        grid3.add(scenetitle, 0, 0, 2, 1);
        Scene scene = new Scene(grid3, 500, 400);
        primaryStage.setScene(scene);
        primaryStage.show();
    } else if (isValid) {
        loginAttempts = 0;  // reset failedAttempts to 0
```

Each time the user enters a failed login, the attempt is captured on screen:



Once the maximum number of login attempts is reached, the program will lock the user out:



AC-8 -SYSTEM USE NOTIFICATION

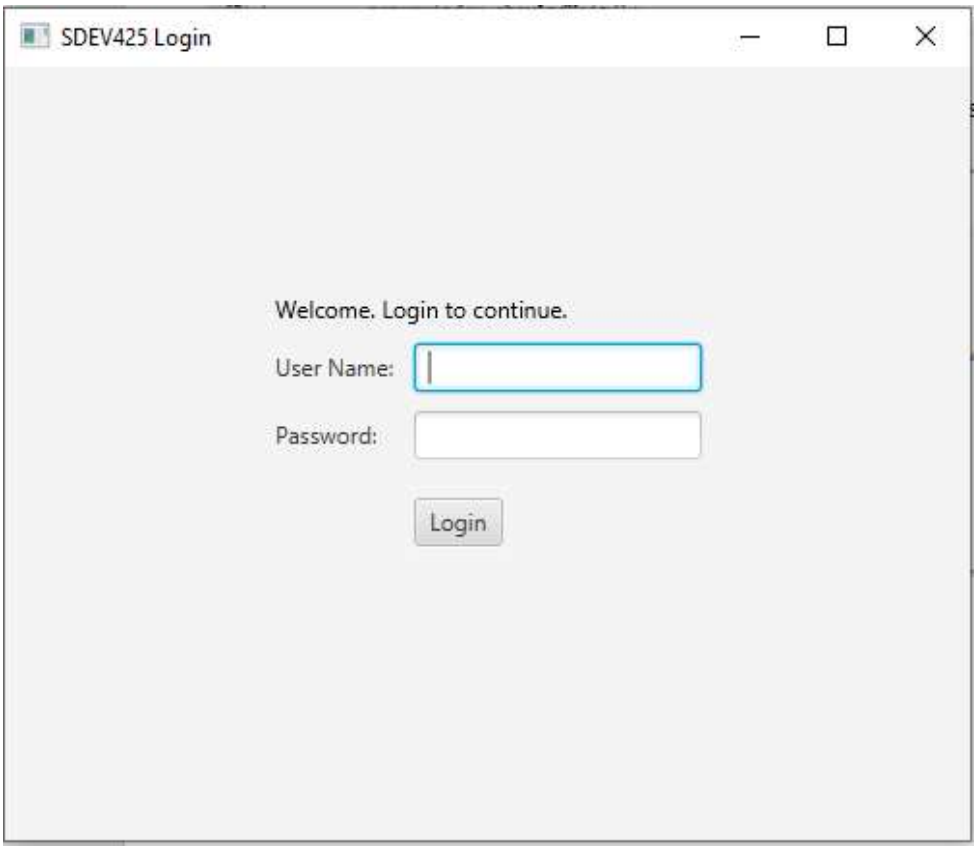
This security control states that users should be informed of any security and privacy notices pertaining to their use of the system (NIST, 2021b). Such notices may be determined by applicable laws, executive orders, directives, regulations, policies, standards, and guidelines (NIST, 2021b). The user should be required to acknowledge these notices prior to being able to login and use the system (NIST, 2021b).

I implemented this security control through another Stage that opens up before the primaryStage is set. This Stage, called popupwindow informs the user of various security and privacy notices and requires the user to click 'I AGREE' before moving onto the main login screen. Code was taken and modified from Learning About Electronics (2018). See screen shot below for details:

```java
public void start(Stage primaryStage) {
    primaryStage.setTitle("SDEV425 Login");

    // create a popup window that requires the user to acknowledge the posted
    // security and privacy notices before logging in.
    Stage popupwindow = new Stage();

    popupwindow.initModality(Modality.APPLICATION_MODAL);
    popupwindow.setTitle("PRIVACY NOTICE: CONSENT TO MONITORING");
    Label labelPrivacy= new Label("1.   You are accessing a U.S. Government system;\n" +
    "2.  System usage may be monitored, recorded, and subject to audit;\n" +
    "3.  Unauthorized use of the system is prohibited and subject to criminal and civil penalties; and\n" +
    "4.  Use of the system indicates consent to monitoring and recording;\n" +
    "\nPlease click 'I AGREE' to continue using sytem.");
    Button buttonAgree= new Button("I AGREE");
    buttonAgree.setOnAction(e -> popupwindow.close());
    VBox layout= new VBox(10);

    layout.getChildren().addAll(labelPrivacy, buttonAgree);
    layout.setAlignment(Pos.CENTER);
    Scene popup= new Scene(layout, 550, 200);

    popupwindow.setScene(popup);
    popupwindow.showAndWait();
```

When the program is launched, this is the screen that the user is first shown:

After the user clicks ‘I AGREE’ then the main login menu appears:



AU-3 -CONTENT OF AUDIT RECORDS

This security control specifies that relevant information about system events are logged to allow for future audits (NIST, 2021c). Audit records should include a description of the type of event that occurred, a timestamp for the event, a source and destination address, user or process identifiers, the outcome of the event (i.e. success or failure), and any filenames involved (NIST, 2021c). Organizations should also consider how audit records can potentially affect personally identifiable information (PII) (NIST, 2021c).

I implemented this control with a FileHandler and event logger . The logger captures relevant info such as a datetime stamp, system info pertaining to the program build, the host IP address and hostname, and event info explaining what event has taken place, along with a success/failure message. The log is written to the logging.txt file in the project main directory. See code example below:

```java
// get host IP and hostname info for log
InetAddress ip;
String hostname;
ip = InetAddress.getLocalHost();
hostname = ip.getHostName();

// Create a file handler object
FileHandler handler = new FileHandler("logging.txt");
handler.setFormatter(new SimpleFormatter());

// Add file handler as handler of logs
logger.addHandler(handler);

// Set Logger level()
logger.setLevel(Level.FINE);
// write to log
logger.log(Level.INFO, "PRIVACY NOTICE MESSSAGE\nHost IP address: "
        + "{0}\nHostname: {1}", new Object[]{ip, hostname});
```

Below is a copy of the log generated when a user launched the application, accepted the privacy notice information, failed to login twice, then successfully logged in.

Jul 12, 2021 9:22:12 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion start

INFO: PRIVACY NOTICE MESSSAGE

Host IP address: PWalsh-E570/172.31.208.1

Hostname: Pwalsh-E570

Jul 12, 2021 9:22:14 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion start

INFO: USER ACCEPTED PRIVACY NOTICE MESSSAGE

Host IP address: Pwalsh-E570/172.31.208.1

Hostname: Pwalsh-E570

Jul 12, 2021 9:22:17 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle

INFO: Login attempt # 1

Host IP address: Pwalsh-E570/172.31.208.1

Hostname: Pwalsh-E570

Jul 12, 2021 9:22:17 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle

INFO: Failed to login

Jul 12, 2021 9:22:18 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle

INFO: Login attempt # 2

Host IP address: Pwalsh-E570/172.31.208.1

Hostname: Pwalsh-E570

Jul 12, 2021 9:22:18 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle

INFO: Failed to login

Jul 12, 2021 9:22:29 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle

INFO: Login attempt # 3

Host IP address: Pwalsh-E570/172.31.208.1

Hostname: Pwalsh-E570

Jul 12, 2021 9:22:29 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle

INFO: Successful login

AU-8 -TIME STAMPS

This security control specifies that audit records should include a datetime stamp in Coordinated Universal Time (UTC) (NIST, 2021d).

To implement this control, I added an Instant object called timestampUTC to capture the current datetime in UTC:

```
51        private static final Logger logger = Logger.ge
52        // initialize datetimestamp in UTC for log
53        Instant timestampUTC = Instant.now();
54
```

I used timestampUTC in the logger to capture the UTC time:

```
74                logger.setLevel(Level.FINE);
75                // write to log
76                logger.log(Level.INFO, "PRIVACY NOTICE MESSSAGE\nHost IP address: "
77                        + "{0}\nHostname: {1}\nUTC timestamp: "
78                        + timestampUTC, new Object[]{ip, hostname});
79
```

Below is an example of an updated log that includes the datetime in UTC:

```
logging.txt - Notepad
File  Edit  Format  View  Help
Jul 12, 2021 10:12:02 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion start
INFO: PRIVACY NOTICE MESSSAGE
Host IP address: PWalsh-E570/172.31.208.1
Hostname: PWalsh-E570
UTC timestamp: 2021-07-13T02:12:02.132Z
Jul 12, 2021 10:12:22 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion start
INFO: USER ACCEPTED PRIVACY NOTICE MESSSAGE
Host IP address: PWalsh-E570/172.31.208.1
Hostname: PWalsh-E570
UTC timestamp: 2021-07-13T02:12:02.132Z
Jul 12, 2021 10:12:27 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle
INFO: Login attempt # 1
Host IP address: PWalsh-E570/172.31.208.1
Hostname: PWalsh-E570
UTC timestamp: 2021-07-13T02:12:02.132Z
Jul 12, 2021 10:12:27 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle
INFO: Failed to login
Jul 12, 2021 10:12:28 PM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle
INFO: Login attempt # 2
```

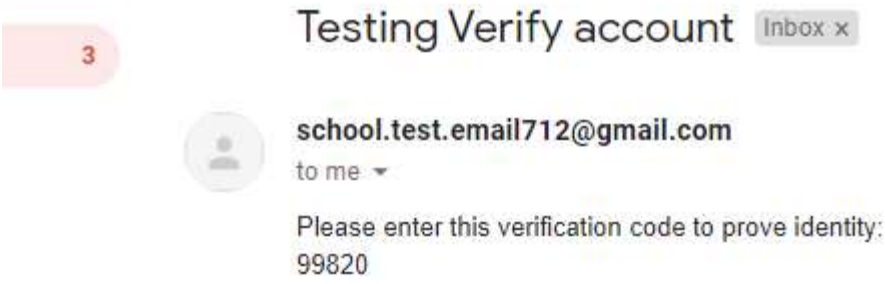
IA-2(1) IDENTIFICATION AND AUTHENTICATION (ORGANIZATIONAL USERS) | NETWORK ACCESS TO PRIVILEGED ACCOUNTS (Note this is an enhancement of an existing low-impact security control)

This security control requires that system uniquely identify and authenticate users prior to accessing systems (NIST, 2021e). This can be accomplished through Multifactor Authentication (MFA) techniques such as passwords, physical authenticators, and biometrics (NIST, 2021e).

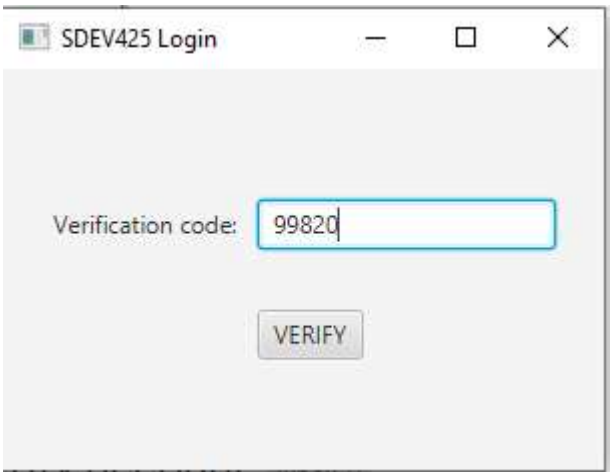
I implemented this control using the JavaMail API to send an email when the user attempts to login. The API connects to my Gmail SMTP server to send an email with a randomized 5-digit verification code. The user must access the email and retrieve the 5-digit verification code. Once they do so, the program will ask them to enter the verification code to authenticate. See the code snippet taken and modified from Oracle (2021):

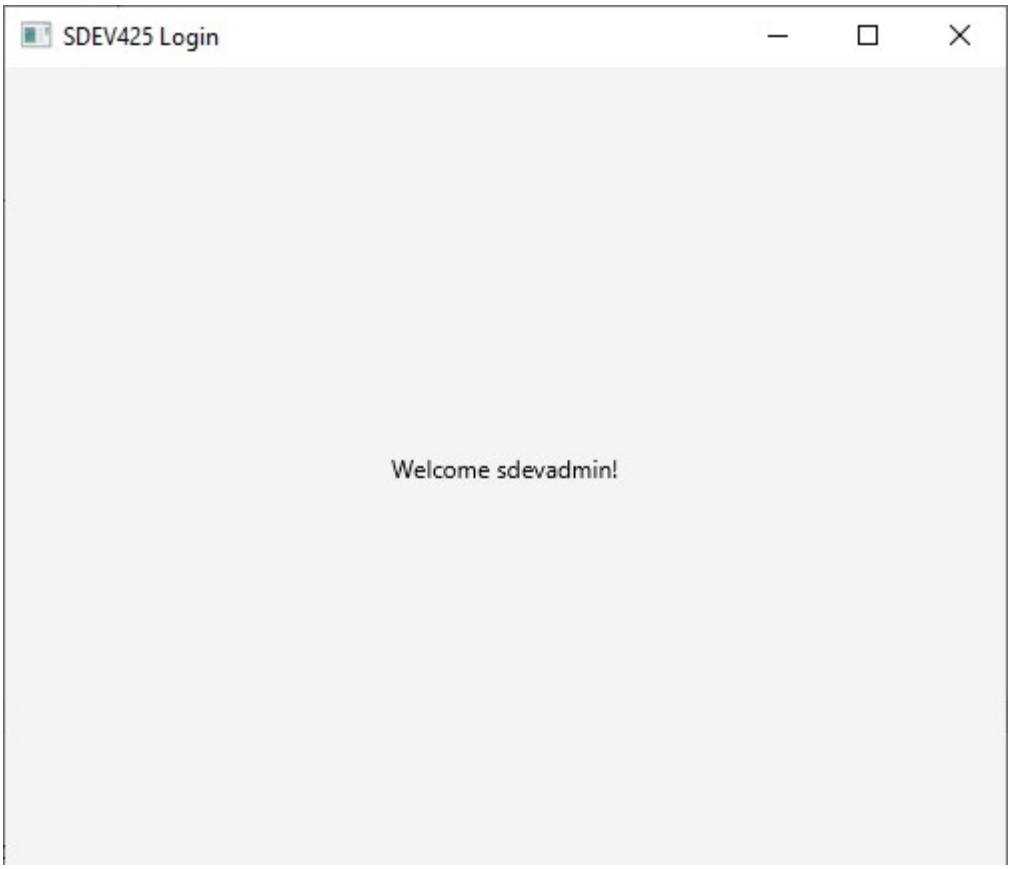
```java
public int twoFactorAuth(){
    Properties props = new Properties();

    props.put("mail.smtp.host", "smtp.gmail.com");
    props.put("mail.smtp.socketFactory.port", "465");
    props.put("mail.smtp.socketFactory.class",
                "javax.net.ssl.SSLSocketFactory");
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.port", "465");

    Session session;
    session = Session.getDefaultInstance(props,
            new javax.mail.Authenticator() {
                @Override
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication("school.test.email712", "schoolll!!");
                }
            });

    try {

        // Generates random verification code between 10000 and 99999
        int min = 10000;
        int max = 99999;
        int verify = (int)(Math.random() * (max - min + 1) + min);

        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress("school.test.email712@gmail.com"));
        message.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse("school.test.email712@gmail.com"));
        message.setSubject("Testing Verify account");
        message.setText("Please enter this verification code to prove identity:" +
                    "\n" + verify);

        Transport.send(message);

        return verify;
```

When the correct username and password are entered, the verification screen appears. The user must check their email to retrieve the verification code:



When the code is entered, the user has successfully authenticated and they are allowed to login:





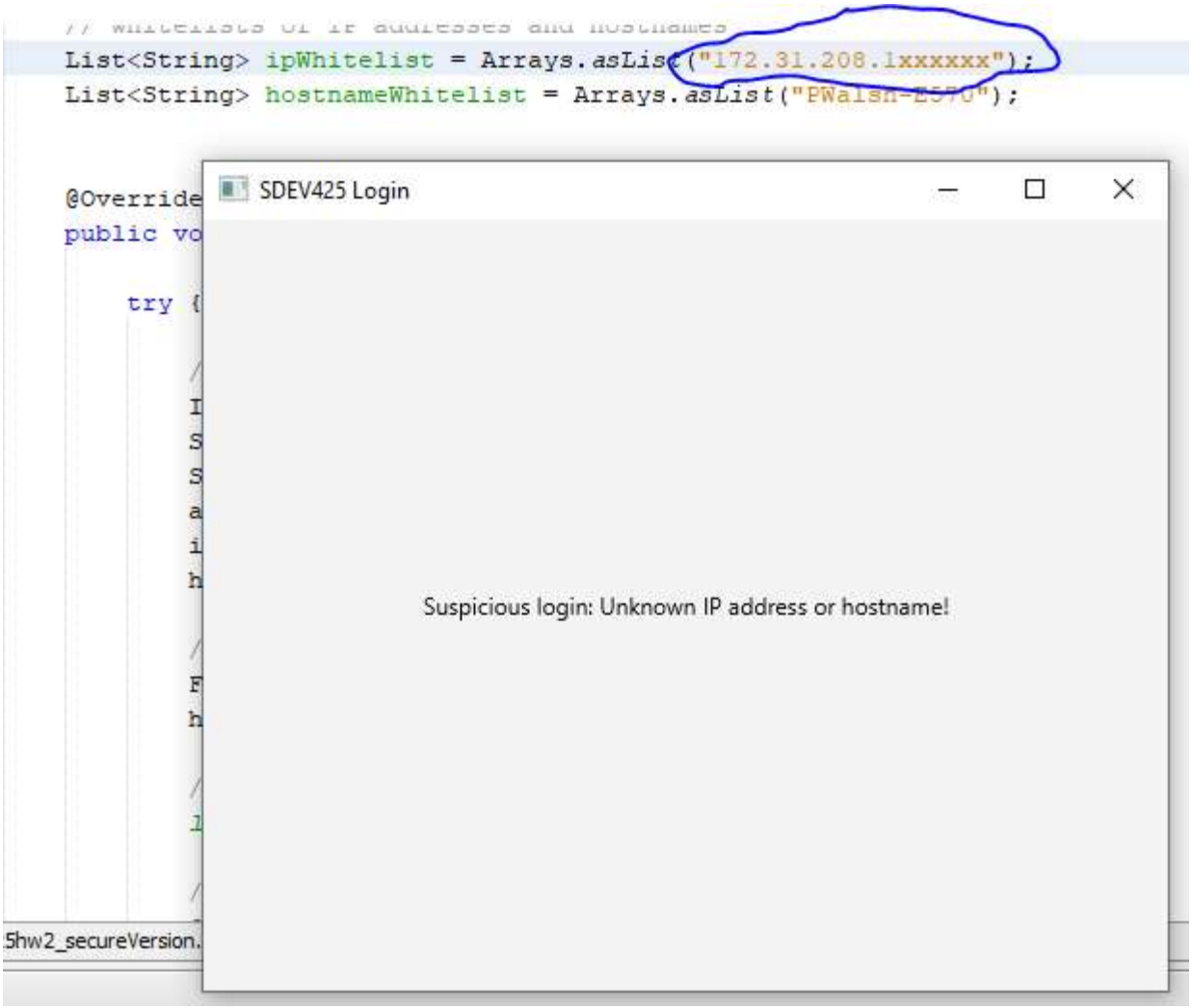
Select one additional low-impact security control and implement it. This can be an enhancement or a required low-impact security control. Selecting a control that provides documentation as opposed to code changes is also acceptable and encouraged.

For the additional low-impact security control, I chose SI-4 SYSTEM MONITORING. This security control advocates internal and external system monitoring for indicators of attacks or other security threats (NIST, 2021f). This includes identifying unauthorized use of systems by monitoring and analyzing suspicious activity (NIST, 2021f). One type of suspicious activity could be a login from an unknown IP address or hostname. So to implement this security control, I added a security check where the system checks the IP address and hostname of the user attempting to login before letting them access the system. In the code snippet below, an IP address and hostname whitelist are created to check against what is logged in the system:

```java
boolean useWhitelist = true;
// whitelists of IP addresses and hostnames
List<String> ipWhitelist = Arrays.asList("172.31.208.1");
List<String> hostnameWhitelist = Arrays.asList("PWalsh-E570");
```

The program first checks if the whitelist is being used. If it is, the IP address and hostname are checked against the whitelist and the system gets locked out if there is a suspicious login from an unknown IP address or hostname. In such events the activity is written to the log along with the captured IP address and hostname.

```java
if (useWhitelist == true){  // checks IP address and hostname to make sure they are on whitelist
    System.out.println("working");
    if (ipWhitelist.contains(ip) && hostnameWhitelist.contains(hostname)){
//          System.out.println("contain(ip)");
    } else {
        useMFA = false;
        locked = true;
        // write to log
        logger.log(Level.INFO, "Suspicious login: Unknown IP address or hostname\nHost IP address: "
                + "{0}\nHostname: {1}\nUTC timestamp: "
                + timestampUTC, new Object[]{ip, hostname});
        grid.setVisible(false);
        GridPane grid4 = new GridPane();
        // Align to Center
        // Note Position is geometric object for alignment
        grid4.setAlignment(Pos.CENTER);
         // Set gap between the components
        // Larger numbers mean bigger spaces
        grid4.setHgap(10);
        grid4.setVgap(10);
        Text scenetitle = new Text("Suspicious login: Unknown IP address or hostname!");
        // Add text to grid 0,0 span 2 columns, 1 row
        grid4.add(scenetitle, 0, 0, 2, 1);
        Scene scene = new Scene(grid4, 500, 400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



Jul 13, 2021 11:35:25 AM sdev425hw2_secureVersion.SDEV425HW2_secureVersion$1 handle
INFO: Suspicious login: Unknown IP address or hostname
Host IP address: 172.31.208.1
Hostname: PWalsh-E570
UTC timestamp: 2021-07-13T15:35:17.367Z

References

Learning About Electronics. (2018). *How to Create a Pop Up Window in JavaFX*. Learningaboutelectronics.Com. http://www.learningaboutelectronics.com/Articles/How-to-create-a-pop-up-window-in-JavaFX.php

National Institute of Standards and Technology (NIST). (2021a, May 28). *AC-7 UNSUCCESSFUL LOGON ATTEMPTS*. NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/control?version=5.1&number=AC-7

National Institute of Standards and Technology (NIST). (2021b, May 28). *AC-8 SYSTEM USE NOTIFICATION*. NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/control?version=5.1&number=AC-8

National Institute of Standards and Technology (NIST). (2021c, May 28). *AU-3 CONTENT OF AUDIT RECORDS*. NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/control?version=5.1&number=AU-3

National Institute of Standards and Technology (NIST). (2021d, May 28). *AU-8 TIME STAMPS*. NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/control?version=5.1&number=AU-8

National Institute of Standards and Technology (NIST). (2021e, May 28). *IA-2 IDENTIFICATION AND AUTHENTICATION (ORGANIZATIONAL USERS)*. NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/control?version=5.1&number=IA-2

Oracle. (2021). *Using the JavaMail API*. GlassFish Server Application Development Guide. https://docs.oracle.com/cd/E26576_01/doc.312/e24930/javamail.htm#GSDVG00021

National Institute of Standards and Technology (NIST). (2021f, May 28). *SI-4 SYSTEM MONITORING*. NIST Risk Management Framework. https://csrc.nist.gov/Projects/risk-management/sp800-53-controls/release-search#!/control?version=5.1&number=SI-4