**Implementing Low-Impact Security Controls**

**Adding Select Security Controls to a Login Application**

Benjamin M. Brandhorst

University of Maryland Global Campus

SDEV 425 7980 Mitigating Software Vulnerabilities

Doctor Ronald McFarland

April 12th, 2020

**Part 1: Review and Understand Sample JavaFX application**

Successful setup and execution of the “Hello World” application is demonstrated in Figure 1.

A screenshot of a cell phone

Description automatically generated

Figure 1.1 – Hello World application execution

The SDEV425 Login application is a simple JavaFX application with hardcoded credentials saved in the source code. Figure 2 shows both the source code containing the credentials and the login menu. Figure 3 shows the application response once a user has successfully logged in.

A screenshot of a cell phone

Description automatically generated

Figure 1.2 – Hard coded credentials example

A screenshot of a cell phone

Description automatically generated

Figure 1.3 – Successful login demonstration

The first part of this program I’d like to review is the GUI. As you can see from the code below, the GUI is constructed using a Grid layout and consists of a window title, text scene title, two labels, a user text field, a password field, and a button.

A screenshot of a cell phone

Description automatically generated

Figure 1.4 – GUI

Clicking the “Login” button triggers the response seen in Figure 1.5. The very first thing checked in the event handler pulls a response from the authenticate method shown in Figure 1.6.

A screenshot of a cell phone

Description automatically generated

Figure 1.5 – Event Handler

A screenshot of a cell phone

Description automatically generated

Figure 1.6 – Hard coded credentials in authenticate method

The authenticate check confirms the username is a non case sensitive version of “sdevadmin” and the password is “425!pass”. If both inputs match, “isValid” returns as true and the event handler executes the “if (isValid)” portion of the method. Otherwise the “else” portion I executed, and users are asked to try again.

Finally, the Main method is what actually executes the program and is shown in Figure 1.7

A close up of a black background

Description automatically generated

Figure 1.7 – Main method

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

**AC-7 - UNSUCCESSFUL LOGON ATTEMPTS**

The application as is, had no controls in place to limit the number of consecutive login attempts. The challenge for me here was implementing something locally since this application does not connect to a database. If it did, the implementation of a lockout timer would have been more elegant but working with what tools I had available this vulnerability was mitigated through the use a of a counter in the “else if” block of the Login button handler (seen in Figure 2.1). Once a user has tried to login unsuccessfully five times in a row, the “badLogin()” method is run which uses a buffered writer to write the current system time in milliseconds to a text file. (see figure 2.2).

A picture containing green

Description automatically generated

Figure 2.1 – x counter demonstration

A screenshot of a cell phone

Description automatically generated

Figure 2.2 – badLogin() method demonstration

Once this event is triggered, users are given a warning that they have tried to log in too many times and should try again later (see Figure 2.3). Finally, even if the user closes the application, the value of the lockout.txt file is compared against the current system time. If the current system time value is not 1.8 million greater (30 minutes in milliseconds), the user continues to be locked out of the program (see Figure 2.4).

A screenshot of a cell phone

Description automatically generated

Figure 2.3 – Login lockout results

A screenshot of a cell phone

Description automatically generated

Figure 2.4 – lockout.txt value check against current system time

Any time a user successfully logs in, the value assigned to lockout.txt is reset to zero.

**AC-8 - SYSTEM USE NOTIFICATION**

The System Use Notification requirement was met through the use of a modal information alert. Each time a user clicks the login button, a modal notification pops up that requires users to interact with before they can use the application. Figure 2.5 shows the method used. Figure 2.6 shows the actual notification window in use.

A screenshot of a cell phone

Description automatically generated

Figure 2.5 – notification() method source code example

A screenshot of a cell phone

Description automatically generated

Figure 2.6 – System Use Notification example

**AU-3 - CONTENT OF AUDIT RECORDS & AU-8 - TIME STAMPS**

Audits with time stamps were implemented by the creation of a separate AuditLogger class from which methods that record actions taken in the program are called and records of the events are stored in log file.

A screenshot of a cell phone

Description automatically generated

Figure 2.7 – AuditLogger class source code

Figure 2.8 shows one of several methods from the AuditLogger class being called to record the program starting up. Figure 2.9 shows the contents of the Audit.log after testing the application.

A picture containing drawing

Description automatically generated

Figure 2.8 – Calling the programStart() method

A screenshot of a social media post

Description automatically generated

Figure 2.9 – Contents of the Audit.log file after a couple of tests were performed with the application

**IA-2(1) Identification and Authentication (Organizational Users) | Network Access to Privileged Accounts**

Two factor authentication was implemented through the creation of a separate TwoFactor class that uses much of the sample Javamail program code to send an email with a randomly generated four digit code. Figure 2.10 shows the TwoFactor class. The program checks the username and password against the hard coded credentials and if the user inputs match, then a new window asking the user to provide the authentication code to continue is shown. Figure 2.10 shows the source code of the class file. Figure 2.11 shows the source code used to pass the four digit code through the call to the sendEmail() method. Figure 2.12 shows the successful execution of the code in question.

**A screenshot of a cell phone

Description automatically generated**

Figure 2.10 – TwoFactor class source code

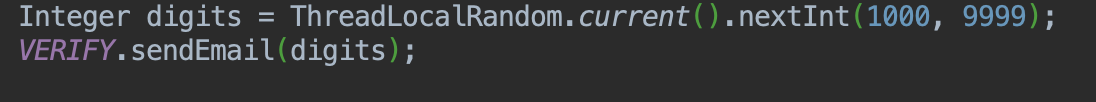


Figure 2.11 – Calling the sendEmail() method from the Homework2 class

A screenshot of a cell phone

Description automatically generated

Figure 2.12- Two Factor menu and Email

**AU-11 Audit Record Retention**

This control specifies that our organization must retain records for an organizationally specified period of time which should be consistent with our records retention policy. Ideally this will be until the records are no longer needed for audit, legal, administrative, or any other purposes.

I am using Baylor University’s Records Retention and Archival Policy as an example from which my own could be built.:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

References

FileHandler (Java platform SE 8 ). (2020, January 6). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/util/logging/FileHandler.html

How do you store logs in a single log file using java.util.logging.FileHandler? (n.d.). Retrieved from https://stackoverflow.com/questions/50964565/how-do-you-store-logs-in-a-single-log-file-using-java-util-logging-filehandler

Java - Sending email. (n.d.). Retrieved from https://www.tutorialspoint.com/java/java\_sending\_email.htm

JavaFX alert dialogs. (n.d.). Retrieved from https://o7planning.org/en/11529/javafx-alert-dialogs-tutorial

Jenkov, J. (n.d.). JavaFX stage. Retrieved from https://tutorials.jenkov.com/javafx/stage.html

Logging in Java. (2018, May 21). Retrieved from https://www.geeksforgeeks.org/logging-in-java/

Records Retention and Archival Policy. (n.d.). Retrieved from https://www.baylor.edu/content/services/document.php/184655.pdf