Lab 4

By

Swapnil Patel

In this lab, I used an application I developed prior to show the vulnerabilities- Missing Authorization & Use of Hard-coded Credentials. This is a simple JavaFX application, in the unsecure version, the application simply asks for the user to login with a username and password, without any authentication factors (MFA), and uses a hardcoded password and username. In the fixed solution, a MFA was added, as well as mitigating the use of a hardcoded password.

The original login credentials were simply an equals to comparison, which was hard coded as seen below:

public static void main(String[] args) {

launch(args);

}

/\*\*

\* @param user the username entered

\* @param pword the password entered

\* @return isValid true for authenticated

\*/

public boolean authenticate(String user, String pword) {

boolean isValid = false;

if (user.equalsIgnoreCase("sdevadmin")

&& pword.equals("425!pass")) {

isValid = true;

}

return isValid;

}

}

You can see here that the password and username is in plain sight and obviously hard coded, however, to mitigate this, I created another class, which is private, with the username and password fields, using getters I am ensuring that I am being safe as possible. In reality, I would have created a hashmap of the usernames and passwords, however for the sake of simplicity, the account class was left private and secured through getter user only.

The mitigated solution:

public boolean authenticate(String user, String pword) {  
 boolean isValid = false;  
 Account acc = new Account();  
 if (user.equalsIgnoreCase( acc.getUsername() )  
 && pword.equals( acc.getUpass() )) {  
  
 isValid = true;

You can see here that the main class, where the user is interacting with does not have hardcoded username and password. This mitigates the CWE-798: Use of Hard-coded Credentials.

The next security vulnerability patched was the missing authentication factor, as 2-factor or MFA has become a standard practice today due to the massively singular point of failure in the single authentication standard of just a username and password. To create this MFA part of the application, I used javamail and created a key generator that uses Java’s execution method to advantage.

The secure MFA was created using a class and a method in the main class:

public boolean MFACode(String user, String twofnum) {  
 boolean accessCode = false;  
 if (twofnum.equalsIgnoreCase( JavaMail.*getSEND*() )) {  
 accessCode = true;  
 }  
 return accessCode;  
}

Class:

package LogMeOn;  
  
import java.util.Random;  
  
public class KeyGen {  
  
 static char[] OTP(int length) {  
 System.*out*.println( "Generating OTP" );  
 String numbers = "0123456789";  
 Random random = new Random();  
 char[] otp = new char[length];  
 for (int i = 0; i < length; i++) {  
 otp[i] = numbers.charAt( random.nextInt( numbers.length() ) );  
 }  
  
 return otp;  
  
  
 }  
  
  
  
}

You can see how the random key is generated in the source code, but I added to function that the keygen.java class does not actually store or create any key. The key is created and stored in a private static variable, and only instantiated during the successful matching of the first authentication method which is the username and password. The main class is using a getter created in the JavaMail.java class to reference a private static variable. Also, the key stored is destroyed on application exit. This mitigates the CWE-862: Missing Authorization, more specifically, NIST 800-53 IA-2.