Critical Thinking 6

Gregory Norris

CSC 450

Colorado State University Global Campus

Critical Thinking 6

Option 1: Compliant / Noncompliant Solutions

In chapter 1 of Java Coding Guidelines there is a non-compliant code section given for reading a password from a file stream with the theme of “Limiting the lifetime of sensitive data”:

void readData() throws IOException{

BufferedReader br = new BufferedReader(new InputStreamReader(

new FileInputStream("file")));

// Read from the file

String data = br.readLine();

}

This is non-compliant because it passes the data into a String, which is an immutable data type. The problem with storing sensitive information as a String is that when altering a String, an altered copy of the string is created, which can leave the original string floating around in memory until the Garbage Collector handles it. Since garbage collection is handled automatically, the programmer gives up any control of what happens to that data as soon as it is copied into a string. A preferred alternative is to use either character arrays or a bye buffer which can be manually cleared by the programmer immediately after they are no longer needed.

In my example Java program, I choose to use a Java Swing application to accept a password from a user, then check if the password is accepted or not. My program utilizes JPasswordField which returns a visible array of obscured characters as the user types ‘\*”, then returns the password as a character array. Upon the user hitting enter, the program clears the obscured characters since they are no longer needed. These characters are stored in a string, but contain the length of the password, so clearing it immediately can help prevent some shoulder-surfing attacks which may be possible if someone knows the length of the password. After clearing JPasswordField, the program uses a static method to validate the password input. I decided that the program should clear the char array at this point since it is the very next line after the password has been approved or not. If I cleared the char array within the parent method containing the JPassword field, the program will return a Boolean value, close the scope of CheckPassword, perform a Boolean logic check on the return value, then (potentially) accidently print the password to the console. Although it would have been easier to clear the password after the if/else statement, the additional program steps increased the time the unsecured password is stored in system memory after validation.

Another way I could have done this would be to take the password as a byte buffer instead of a character array. This would be more efficient if I wanted to use a passed hash value to check against another hash since the operation is performed as a btye stream. This would be more efficient than converting between characters to check the password. As with the character array above, the byte buffer needs to be cleared as soon as the stored information is no longer needed to limit the opportunity for a malicious attacker to dump the memory from the program to retrieve the password.

Ideally, the program can convert the password into a hash using something like SHA or bcrypt before performing any logical checks. The program should also store passwords in hashed form and never perform any sort of validation operations on a plaintext password, nor should plaintext passwords ever be stored.