Weng Lam Sio
Kris Nakamura
Vinson Gao

Assignment 2 Report

Our app helps Minions to track their work hours, as well as their wages in bananas. The Minions will each have their own username as well as password to access this information. This document covers the possible security breaches that may occur as well as the severity of the issues.

REQUIREMENTS

1. Usually any type of applications which track confidential information such as a time card applications or W-2 forms must be dealt with the utmost care and security. There are two privilege levels associated with these functions: Worker or Manager. For the worker, there are two functions for our app; one part which allows Minions to see their worked hours as well as their wages, and another part which allows the Minions to clock in and clock out. The manager has all of the same functions except that they can also create/modify user account information. The app must be very secure regardless of the user's privilege level. The Minion's username, password, and passcode will be stored in a secure location separate from the worked hours and wages. The clock-in clock-out feature is restricted to simply clocking in and out, so it shouldn't have to take a long time. To keep it simple while enforcing security, that feature will have a separate passcode which is different from their password to login to see their hours and wages.

Any security flaws that become apparent in our app will be tracked via the ‘issues’ tab on the right hand side of Github. The issues can be assigned to a specific member to fix or if it seems complex, multiple people can work on an issue. The issue will be closed once it is solved so that unnecessary attempts to fix a solved issue does not occur. We will also have discussions to ensure that everyone is up to date with any issues at least once a week.

2. End-User Scenarios
    - Critical
        - Lack of notice and consent
            - Example: Minion's username and password are collected and stored without notice. Data are accessible by others.
        - Lack of user controls
            - Example: Automatically collect Minion's wages in bananas at the end of every week, create ranking using the data without the ability in the UI for Minions to stop the collection.
        - Lack of data protection
            - Example: Minions can change their username and password without an authentication mechanism such as entering their current password as they creating a new password and answering security questions.
        - Data minimization

        - Example: Minion's worked hours per week and wages in bananas is transmitted to the supermarket when the manager is considering the amount of bananas needed to be purchased.
    - Improper use of cookies
        - Example: Minion's username and password stored in a cookie is not encrypted.
    - Insufficient legal controls
        - Example: There's no legally approved contract between the Minions and the organization.
- Important
    - Lack of notice and consent
        - Example: Minion's worked hours and wages are collected and stored without notice. Data are accessible by others.
    - Lack of user controls
        - Example: Automatically collect Minion's worked hours at the end of every week, create ranking using the data without the ability in the UI for Minions to stop the collection.
    - Lack of data protection
        - Example: Minions can simply change their username and password by entering their current password as they creating a new username or password.

    - Improper use of cookies
        - Example: The answer to the security question stored in a cookie is not encrypted.
    - Lack of internal data management and control
        - Example: Minion's worked hours and wages stored at organization can be viewed anytime by the manager even when there is no valid business need.
- Moderate
    - Lack of notice and consent
        - Example: Minion's username and password are collected and stored without notice. Data are not accessible by others.
    - Lack of user controls
        - Example: Automatically send email to the manager if the system detected Minions did not clock-in and clock-out by the end of the day, without the ability in the UI for Minions to stop sending the email.
    - Lack of internal data management and control
        - Example: Minion's worked hours and wages stored at organization does not have a retention policy.
- Low
    - Lack of notice and consent

        - Example: Minion's worked hours and wages are collected and stored without notice. Data are not accessible by others.

Enterprise Administration Scenarios

- Critical
    - Lack of enterprise controls
        - Example: Automatically collect Minion's wages in bananas at the end of every week, create ranking using the data without the ability in the UI for the enterprise administrator to stop the disclosure of the ranking.
- Important
    - Lack of enterprise controls
        - Example: Automatically collect Minion's worked hours at the end of every week, create ranking using the data without the ability in the UI for the enterprise administrator to stop the disclosure of the ranking.
- Moderate
    - Lack of enterprise controls
        - Example: No mechanism is provided to avoid the enterprise administrators from receiving unnecessary emails such as when there's no clock-in and clock-out records from a specific Minion on that day.

3. Since the app involves the storage and transfer of Personally Identifiable Information (PII) and monitors the user with ongoing transfer of data, the Privacy Impact Rating of the app is in P1, High Privacy Risk.
    - Describe the PII stored:

      Every Minion is required to enter their own username and password to login to the system.
    - Describe how users can control the feature:

      After Minions login with their account, they can view their worked hours per week and their wages in bananas. They can also change their username and password if necessary.
    - Describe how organizations can control the feature:

      Admins / managers have the same functions as the Minions, but they have additional admin functions that they can use to create user accounts for their staff and modifiering user account detail information. They can also adjust worker’s working hours and wage rate.
    - Describe how you will prevent unauthorized access to PII:

      To prevent unauthorized access to user accounts, we must take serious care on the password by enforcing password complexity or requesting the answer of security questions

DESIGN

1. For the stored worked hours and wages, every Minion's data will be stored in the same database and not encrypted. The reasoning for this is that all of the data will be associate with an ambiguous key that will be stored with the username and password in a separate database. This basically means that even if someone gets access to all of the data in the database, they won't know which data belongs to who. The info will be fetched automatically when the username and password is entered, and care will be taken to prevent SQL injections.

   A simple encryption will be used to store the user's login info. A complex encryption will be avoided due to efficiency reasons as well as the fact that the data being stored is not extremely confidential (no SSN, date of birth, and other sensitive info).

   The timecard feature will be made so that nothing can be input. It will be a simple clock in or out feature.

   For the initial implementation of the app, the admin will only be allowed to create and modify user accounts so that no accidental or malicious deletions of accounts occur.

   One feature we still need to decide on is whether or not to enforce password complexity. Since the regular workers cannot do much with their accounts anyway, a password complexity setting may be enforced for the admin/managers.

2. Privilege Level 1 : Staff, Worker
    - At this level, the user can log in as a staff or worker to perform certain tasks. In our case, the staff or worker is a minion. They will have a username and password to login to the system. They are able to check the amount of hours they have in this week or month as well as check their hourly wage.
    - For convenience, they will be assigned a passcode to perform a quick clock in when they start to work and clock out when they finish work.

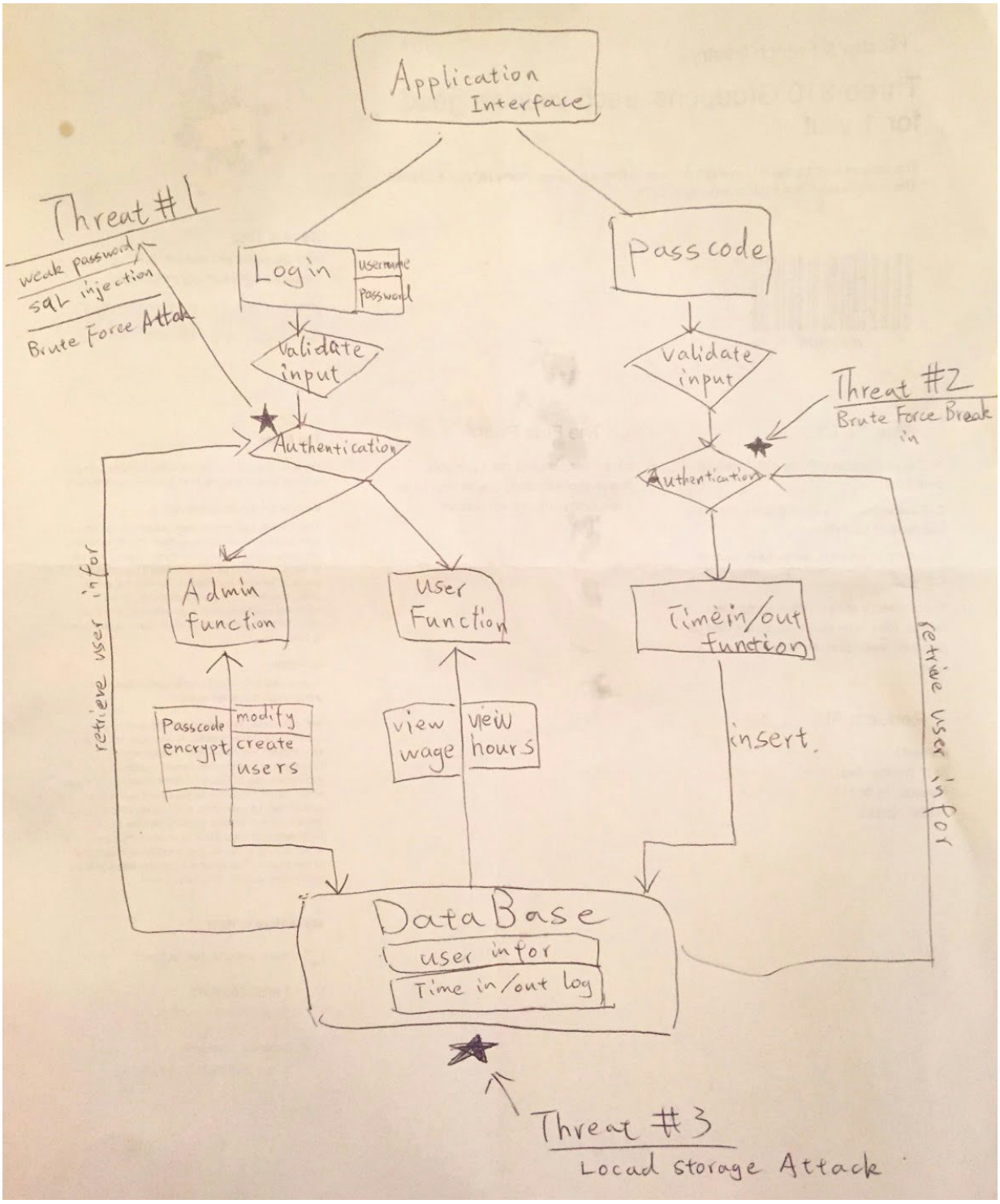
    Privilege Level 2 : Admin, Manager
    - At this level, the user can log in as a manager or admin, who is going to manage user account informations, work hours, and wage.
    - The user can create user / staff accounts, which includes the username, password, and passcode.
    - The user can also create a different admin account, which also includes the username, password, and passcode.
    - The admin/manager can modify the staff or manager's working hours if the staff forgot to clock in or out.

    A List of Attack Surface
    - GUI - forms input, or text field.
    - Local MySQL Databases
    - Login or authentication entry points

    A List of vulnerabilities:

- SQL injection
- Lack of password complexity enforcement.
- Lack of password retry logic.
- Insecure Cryptographic Storage (Did not encrypt Sensitive Data)

3. *see image below*


Application Interface
Threat #1
weak password
sql injection
Brute Force Attack
Login
username
password
Passcode
Validate input
Validate input
Threat #2
Brute Force Break in
Authentication
Authentication
retrieve user infor
Admin function
User Function
Time in/out function
retrieve user infor
Passcode encrypt
modify
create users
view wage
view hours
insert
DataBase
user infor
Time in/out log
Threat #3
Local storage Attack

Assignment 3 Report

Project Repository: https://github.com/rinnsio/ICS491APP

1. Approved Tools:
    - **IDE:** Eclipse (Luna)
    - **Java compiler:** Eclipse Compiler for Java (ECJ)
    - **JRE System Library:** JavaSE-1.8
    - **HTML:** Version 5.0
    - **Source Code Management System:** GitHub
    - **Static Analysis Tool:** Eclipse Checkstyle Plug-in
2. List of unsafe functions followed by safer alternatives:
    - Thread.stop -- Thread.interrupt
    - Thread.suspend -- None that is the same. Use interrupt if anything.
    - Thread.resume -- None at all. Only needed for suspend.
    - Runtime.runFinalizersOnExit -- Use exit or gc (both not very recommended for use though. Better to have it naturally finish). This will help prevent erratic behaviors and/or deadlocks.
    - java.lang.Thread.setPriority -- This method probably isn't required since we don't care much about how much priority our app receives. Would be dangerous if someone sets it to high priority to eat up resources.
    - System.runFinalization -- Similar to runfinalizersonexit, finalizers in general shouldn't really be necessary to force any manual exits on app functionality. Should be avoided.

3. The tool we have decided to use for static analysis is the Eclipse plugin, Checkstyle. The main reason for this choice was that it seemed to be simple to install and work nicely with the IDE (Eclipse) the group is using. Some of us had issues installing the add on while others didn't; Problem was resolved with all persons able to run Checkstyle.

   The tool can be used to check individual files, or to have it automatically check the working project as it is updated.

   The effectiveness of the tool cannot be determined at the moment due to our code being short, new, and incomplete. As of the moment, the tool simply points out spacing errors which is nice to keep the code clean and nicely formatted (easily maintained without needing to scrutinize code). If any depreciated items and/or bad coding practices are used, the add-on will also point them out (tested by purposely introducing a depreciated class).

Assignment 4 Report

Link to repository as well as README: https://github.com/rinnsio/ICS491APP

1. Due to changing our code drastically from the first iteration to the next, as well as not being comfortable with JUnit, our group initially used a simple, separate code that basically does almost the same thing as JUnit does.

   For our main dynamic analysis tool, our group decided to use JUnit to test our program. The reason we chose this tool is that it is easy to setup on Eclipse and it carries most of the methods we needed to test our program. JUnit has methods for you to make sure invalid inputs from users won't crash the program and appropriate error messages would appear to notify the user where the problem is. After a few times using JUnit, we found successful results from testing different inputs from the user. When the user entered appropriate or inappropriate inputs, existing or non-existing usernames, and accurate or inaccurate passwords, the JUnit tests helped us verify that the program returns the values as expected.

2. **SQL Injection:** Since the GUI we have now is not linked to our functionality, testing was done using test files to do SQL injections in code by typing "**bobob"+"; DROP TABLE pin**" for the password input. However, because we used Java's PreparedStatement class for our queries, the pin table did not get deleted from our database. Instead, "**bobob"+"; DROP TABLE pin**" is viewed as the string "**bobob; DROP TABLE pin**" and no user information is returned because the password is inaccurate.

Buffer Overflow: Most of our variables are Strings so we don't have any buffer overflow problem at this moment. Even though we entered more than 100 characters for the inputs, the program still runs fine and returned the expected result. Also, Java pretty much makes it impossible to have buffer overflows in its language to begin with.

Integer Overflow: Same reason as the buffer overflow testing. Since we are using Strings for the variable type, even if the user enters integers, the program will view it as a String and return the expected result. As mentioned earlier, even if we were taking integer inputs, overflows will be prevented because Java will throw type exceptions. Those exceptions simply must be handled correctly.

Decompile the jar file: .jar files are just .zip files with some specific metadata files (META-INF/MANIFEST.MF), so they can be opened with any unzip program like CFR decompiler, Procyon Decompiler, Fernflower Decompiler, JD Core Decompiler, and JAD decompiler. I have used CFR decompiler to decompile the jar file of our project. The result was amazing. All of the class of our project are clearly decompiled and back to java files, which all of our code is visible to the person who decompiled it. For example, hackers can look over our java files and find our local database login information which is unencrypted and can be to used to log in our database externally via terminal or cmd.

To prevent this, we can obfuscate and protect our source code using reverse engineering using a program called ProGuard a free GPL licensed software, which will make the coding more difficult to read and understand, for example:

```
btnNew = changeButtonLabel(btnNew, language.getText("new"));
btnOpen = changeButtonLabel(btnOpen, language.getText("open"));
to
d = a(d, n.a("new"));
e = a(e, n.a("open"));
```

You can see that the line “btnNew = changeButtonLabel(btnNew, language.getText(“new”));” got translated to “d = a(d, n.a(“new”));”, by the ProGuard, which will not make any sense to someone who is using java decompiler tools to reverse engineer the class file. But with anything else, this is only a rough fix and anyone with enough time will be able to break into it.

3 Approved Tools:

- **IDE:** Eclipse (Luna)
- **Java compiler:** Eclipse Compiler for Java (ECJ)
- **JRE System Library:** JavaSE-1.8
- **Server Package of mySQL:** XAMPP v.3.2.1
- **mySQL Connector:** Version 5.1.36
- **Dynamic Analysis Tool:** JUnit 4
- **Source Code Management System:** GitHub

- **Static Analysis Tool:** Eclipse Checkstyle Plug-in

At the moment, there hasn't been any new vulnerabilities that we are aware of in terms of the language/functions we are using. The checkstyle plugin has not detected any issues with our code so far (except for the spacing issues that are easily fixable). This would be a large problem if it were coding in a language where tabbing mattered, such as Python, but Java doesn't have any issues with that.

Basically the only changes there have been would be the version of databasing used since we didn't really have a database before.

Assignment 5 Report

Link to repository as well as README: https://github.com/rinnsio/ICS491APP

1. Privacy Escalation Team:
    - Weng Lam -- Legal Representative: Takes all of the legal issues possibly associated with the issue and acts as a single entity to represent the whole. Deals with legalities such as compensation for any damages caused by the issue.
    - Kris -- Public Relations: Manages any announcements as well as customer support to fix the issue when the vulnerability is fixed/under control.
    - Vinson -- Escalation Manager: Gathers the information on possible issues with the program. Using the information, determines the severity as well as the job allocation to fix the issue.

    Contact email: bananaclockin@hotmail.com.

    This current system of email contact is highly undesirable though, due to the fact that the emails must be manually read by a person logging into the account and then moved to the appropriate folder to prevent multiple people from working on the same issue. Also, there is a security risk with three different people having the same login and password for the email.

    The ideal system would be to use a more professional email or ticketing system such as ServiceNow. That way, multiple people can access the same

issue while allowing them to allocate an issue to a specific person (assignment of tasks).

Incident response procedures:

1. Determine urgency of the support email/ticket and allocate appropriately.
2. Gather all information possible on the possible vulnerability as well as possibly affected persons if applicable.
3. The escalation manager will appropriately allocate the job responsibilities on fixing the issue.
4. The legal representative will take care of any possible legal complications that may have occurred with the discovery of the issue.
5. Once the problem is in control, PR will disclose any required information as well as work together with the legal department if needed.

2. After conducting the final security review with our threat models, junit test, and bug bars, our program passed FSR with exceptions. Below are the reasons of our decision:

   A. For SQL injection:

      Since our program's attack interface is on the username and password field, we are using parameters and Java's PreparedStatement class to prevent SQL injection attacks, i.e. passing "' or 1=1" will not result in unintended data being returned. The username and the query will be

sent to the database as two separate entities, and the database-engine will be responsible for putting the two back together. The query is already compiled by the engine by the time the parameter is read, so the two are never considered part of the same statement.

B. Decompile the built jar file on the client side that installed our application:

This is the the part that caused the FSR with exceptions. Because the .jar files are simply .zip files with some specific metadata files (META-INF/MANIFEST.MF), they can be 'opened' with any unzip program like CFR decompiler, Procyon Decompiler, Fernflower Decompiler, JD Core Decompiler, and JAD decompiler. If a hacker de-compiles our application, they could easily find our database login information and attack our mysql database directly whether local or remote. We still could not find a method or a working program that helps us to encrypt the built file to prevent hackers from reverse engineering our program. This is the reason the status of our program is 'passed FSR with exceptions'.

Overall, we can state our application passed FSR with exceptions.

3. All of the code necessary to run our program is located in the repository and can be easily run when pulled as a project and executed. The one small exception is that the program uses a local database which must be imported or created manually. There is a sample database included in the file path:

ICS491Project/src/clockinsystem/minions.sql

The easiest way to import this file on both Windows and Linux would be to have a database called “minions” on a local sql server, make sure the active directory is where the file is, then enter the following command:

mysql -u *username* -p*password* -h localhost minions < minions.sql

Wiki page Link: https://github.com/rinnsio/ICS491APP/wiki

Final Release Version Link: https://github.com/rinnsio/ICS491APP