Chris Caoagdan
Chase Matayoshi
Jaimar John Sison
ICS 491, Endicott

Secure Development Lifecycle

Security and Privacy Requirements

- **Requirements**
  - Secure storage of User Accounts (usernames and password).
  - Secure payments, transfers and other money transactions.
  - Secure password resets and change of other confidential information.
  - Recognizable of device use to log-in.
  - Detection of location of user.
  - Handling of inactivity of user.
  - Notification of any changes of the account to the user.
  - Secure network/server settings.
- **Tracking errors**
  - Usage of handling event errors in specific areas (i.e. inputmismatchexception, etc).
  - Keeping a log on bugs that arises.
  - Use of Source code management system.

Bug Bars

Security Bug Bars

Servers (Within Bank Headquarters)	
Threat Levels	**Situations**
Critical	● Unauthorized access to servers ○ People outside of the bank (hackers) have unauthorized access to sensitive information. ○ Hackers take control of all of the servers, or parts of

	them.
Important	● Unauthorized access to servers ○ People inside of the bank (hackers or employees) have unauthorized access to sensitive information. ● Bypassing Security Features ○ When a firewall gets compromised and/or get deactivated/terminated.
Moderate	● Denial of Service ○ Temporary lack of access servers when resources are used up.
Low	● Information Disclosure (untargeted) ○ When servers are overloaded and random memory leaks are found. ● Tampering ○ When certain information about clients are changed temporarily due to a specific action/program within a specific scenario.

<u>Clients/Users (Bank Application Usage)</u>	
<u>Threat Levels</u>	**<u>Situations</u>**

Critical	● Unauthorized access to sensitive information ○ Viruses/worms/trojans that are injected into the application can override login requirements to gain access to sensitive personally identifiable information (PII) stored on servers.
Important	● Unauthorized access to sensitive information ○ A user is able to access sensitive PII with another client's credentials.
Moderate	● Denial of Service ○ User tries to enter his/her bank account with appropriate credentials in the bank application, but gets denied access due to possible compromise of client credentials.
Low	● Denial of Service ○ Bank application is not able to load due to technical problems on either the device or on the servers. ○ Keeps crashing during usage of bank application.

Privacy Bug Bars

End-User (Bank Customers/Users)	
Threat Levels	**Situations**

Critical	● Unauthorized access to sensitive information ○ While user is using the application, a virus/worm/trojan was injected in the application *during certain situations* and tracks all activities of the user within the application.
Important	● Lack of data protection ○ Data transmitted between the application and the bank is not encrypted properly, or not encrypted at all. ○ Possible interception within data transmission ("man in the middle").
Moderate	● Lack of user controls ○ Users are not able to delete sensitive personally identifiable information (PII) from the bank application when they want to. ● Data minimization ○ Data is not just transmitted between the application and the bank, but between third parties as well. ○ Data transmission to third parties when it is not necessary.
Low	● Lack of notification and consent

	○ Users are not notified when sensitive PII is hiddenly stored on user's device without consent.

Security and Privacy Requirements (Questionnaire/Checklist)

Identify The Project and Main Privacy Contacts

What is the name of the application?

____

Who is responsible for the privacy of the application?

____

Privacy Impact Rating

- ❑ The application continues to monitor the user after his/her use of the application
- ❑ Sensitive personally identifiable information (PII) is stored on the device and is not deleted after use
- ❑ Little or no consent is given to the user about storing sensitive PII on device
- ❑ None of the above

Detailed Analysis of Security and Privacy

- Describe types of data that will be stored on servers:
  - ____
    ____
- Describe types of data that will be stored on user devices:
  - ____
    ____
- Describe how a user get authorized access to sensitive PII/bank accounts:

- ____
  ____

- Describe how unauthorized access to sensitive PII/bank accounts is prevented:
  - ____
    ____

- Describe what can be done in an event where a user's privacy may have been compromised:
  - ____
    ____

Testing Out Security and Privacy on Application

Upon release of the application, the security and privacy of the application will be tested to see if there are any loopholes that compromises the integrity of the application. A detailed analysis of what happened during testing will be made.

Design Requirements

- **User Accounts**
  - Passwords must be a minimum of eight characters with combination of upper and lowercase alphanumeric characters and symbols.
  - Password must be hidden while user is logging in to avoid shoulder screening.
  - Passwords must be encrypted when storing on the database.
  - Must answer three security questions for security purposes when needed.
  - Three attempts is given to user before account is locked. If locked, must call customer service.

    - Notify account users of any changes in account by phone, email, or text. (withdrawal, deposit, etc).

- **Handling inactivity of Users**
    - Account inactivity for five minutes would result to automatic logging off an account.
    - Exiting the browser without logging off would result to automatic logging off an account.
- **Handling unrecognizable device**
    - Application should ask user to answer three security questions when logging in from an unrecognizable device. Three attempts is given before account is locked.
- **Handling suspicious activities**
    - Application must keep a log where the user is located while logging into the account for suspicious activities.
    - All suspicious activities must be informed to the user by the help desk people.
- **Handling payments, money transfer and other money transactions**
    - Payments and other transactions must be completed with password authentication. Three attempts is given before account is locked.
- **Secure Server and Database settings**
    - Make sure all informations that are put on the database are encrypted.
    - Network/Server should have highest security settings possible.

<u>Attack Surface Analysis/Reduction</u>

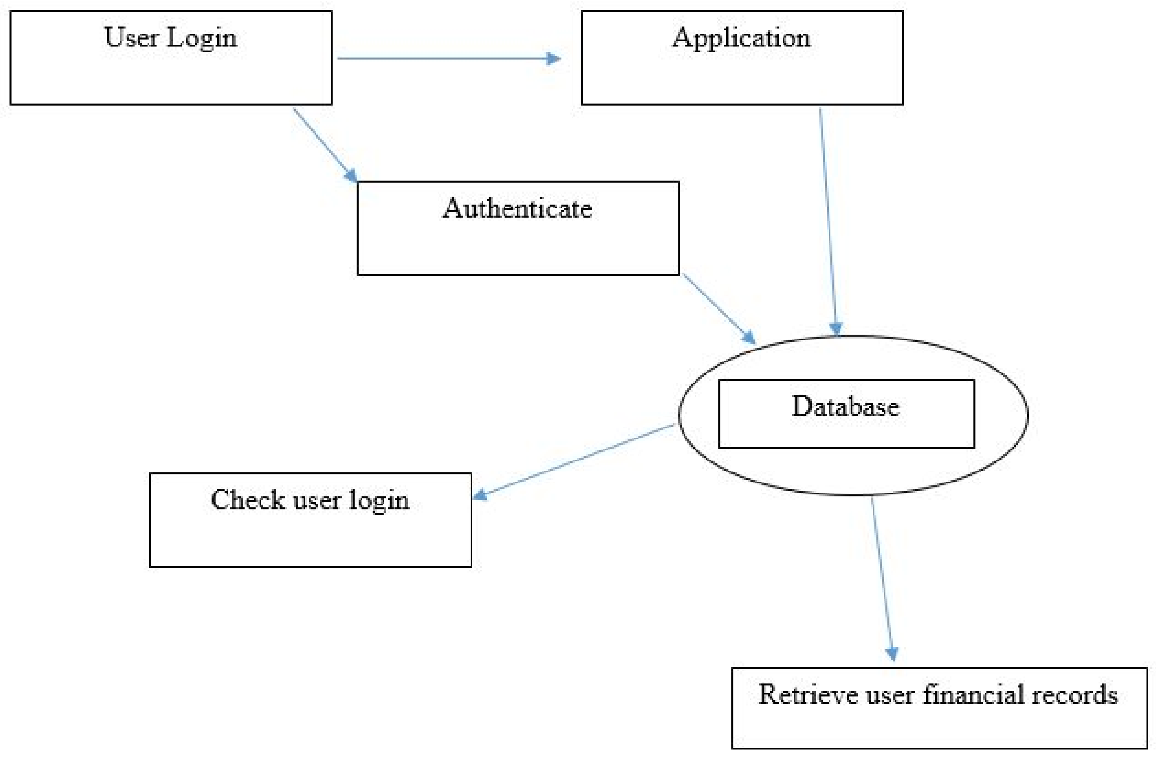
For our application, the type of user will be split into two categories, basic users and admins. Basic users will have access to their own personal information such as, bank records, bank messages, account information, balance, etc. They will also be able to use the standard login page for access to their accounts. Admins will be able to view and assist basic users with their own portals if needed. However, admins do not have access to the personal information that a user has access to. The only thing that admins can view for personal information is the password and history logs of what the basic user did while logged on.

Points of Attack

- User Interface
    - Sequence of unordinary actions that leads to exploit in interface
- User login
    - Brute Force
    - Social Engineering
    - Use of Cookies
    - Keylogging
- TCP/IP Ports
    - Ports compromised through virus or attack
- Database/Server
    - SQL injections
- File Transfer
    - Corrupted files that could compromise the database and network
- Message system

    - Phishing messages that would compromise personal information
- Cookies
    - Misuse of cookies to gain informations that are not encrypted.

Threat Modeling: Data Flow Diagram



Approved Tools

Note: *These approved tools will include those that will work with Windows, Mac OS X, or both.*

This list may be updated later to reflect changes in development and/or compatibility.

Compilers/Tools

Compiler/Tool	Approved Version(s)	Comments
Java Compiler	Java Development Kit	Any version that is released

	(JDK) 8 Update 51	after the approved version may or may not be compatible.
MySQL Server	MySQL Community Server 5.6.26	
MySQL Workbench	6.3.4	
MySQL Connector/J	5.0.8; 5.1.36	Other versions may be used, but they may not work properly.

Integrated Developer Environments (IDEs)

IDE	Approved Version(s)	Comments
Eclipse	Luna Service Release 2 (4.4.2); Mars Release (4.5.0)	Both versions are compatible, but choose only one version to work with.
IntelliJ IDEA	Community Edition 14.1.4; Ultimate Edition 14.1.4	Other versions may be used, but they may not work properly.

Deprecated Unsafe Functions

Deprecated Functions in Java are functions that might or will be obsolete in future Java versions. Below is a list of what we think are useful deprecated functions to use in our project and their counterparts that can be used instead.

List of deprecated functions in Java that might be used:

- **java.awt.Component.action(Event, Object)**
  - **Instead, we can use actionListeners**
- **java.awt.List.addItem(String)**
  - **Instead, we can use add(String)**
- **java.awt.TextArea.appendText(String)**
  - **Same as above**
  - **Instead, we can use append(string)**
- **java.awt.Choice.countItems(), java.awt.Menu.countItems(), java.awt.List.countItems(), java.awt.MenuBar.countItems()**
  - **Instead, we can use getItemCount()**
- **java.awt.List.delItem(int)**
  - **Instead, we can use remove(String) or remove(Int)**
- **java.sql.Time.getYear() , java.util.sql.Time.getYear()**
  - **Instead, we can use *Calendar.get(Calendar.YEAR) - 1900***
- **java.sql.Time.getMonth() , java.util.Time.getMonth()**
  - **Instead, we can use *Calendar.get(Calendar.MONTH***
- **java.sql.Time.getDate() , java.util.Date.getDate()**
  - **Instead, we can use *Calendar.get(Calendar.DAY_OF_MONTH)***

- **java.sql.Time.getDay(),java.util.Date.getDay()**
    - **Instead, we can use *Calendar.get(Calendar.DAY_OF_WEEK)***
- **java.sql.Date.getHours(),java.util.Date.getHours()**
    - **Instead, we can use *Calendar.get(Calendar.HOUR_OF_DAY)***
- **java.sql.Date.getMinutes(),java.util.Date.getMinutes()**
    - **Instead, we can use *Calendar.get(Calendar.MINUTE)***
- **java.sql.Date.getSeconds() , java.util.Date.getSeconds()**
    - **Instead, we can use *Calendar.get(Calendar.SECOND)***

Static Analysis Tool

For our project, our group decided to use the IntelliJ IDE which comes with built in static analysis tools for the developer to use. IntelliJ tells the developer where certain flaws within the program are located and can even suggest fixes for the problem. The interface of the analysis tool is very well put together. The interface groups certain bugs into different categories and allows the user to see them in neat rows for easy access.

After experimenting with the tool during each compilation, the results were very good. The static analysis tool caught several instances where we should have made the variable private. While coding, we also ran into a problem where we could not figure out why an incrementer was not working properly. The tool spotted the error in our program and we were able to fix it. The only bad thing about the tool was that it had some trouble recognizing some hierarchy in catch statements. The cleanup that it suggested made the program unrunnable. Other than that, the tool is great and was a lot easier to use than Checkstyle for Eclipse, which ultimately did not install anyway.

Dynamic Analysis

The tool that we have chosen to do our dynamic analysis is JUnit. Each of our team members has some experience with it, so it is suitable for us to use this tool. In addition, JUnit is available on the IDEs that we use, so it helps us to keep some kind of consistency in our dynamic analysis. When we used JUnit several times, we encounter a few problems. One was that we did not know how to call methods so that we could test them. We were able to write a test to test the method that we want, but it did not work properly/the way we wanted it to work.

In this case, due to the limited nature of our program thus far, we were only able to test out the databases if there is any data stored in the SQL tables. As we continue to work on our program, we intend to update any files used for JUnit testing to be suitable for testing our program.

Fuzz Testing

- Shoulder Screening
  - Since the passwords are not censored when users are logging in their accounts, hackers could easily see what users are typing on the screen. They could plan on remotely controlling the user's computer and watch the screen while the user is logging in or they could just watch them in person while the users are logging in.
  - A good prevention for this is to censor the password.
- Wrong input on some fields
  - Inputs like birthdate should be inputted in the correct format in order for the program to run correctly. Birthdate should be inputted in the format

    YYYY-MM-DD. If input is wrong, the program would stop and leave it vulnerable for an attack by hackers

    - A good prevention for this is by using if statements to force users to input correct input and by handling exceptions.

- Sensitive Data Exposure
    - Information such as user accounts are exposed right now.
    - A good prevention would be to encrypt this sensitive information.
- SQL Injections
    - Our application could be attacked with SQL injections, since we are using a SQL database to store in data.
    - Using parameterized statements for queries is a good technique for preventing SQL injections, like the use of prepared statements in creating a query, which we have done in our program.

Attack Surface Review

For this iteration of the project, our group continued to use the built in tools that are included in the IntelliJ IDE. Compared to the last project, not much has changed in terms of what errors the tools are reporting. A few errors are always reoccurring such as private variables, unused statements and others. Nothing drastic is being reported by the program. As for the unused statements, these are just placeholders for future features that we are planning to implement at a later stage during our development phase.

<u>Incident Response Plan</u>

Escalation Manager - Chase Matayoshi

Legal Representative - Chris Caoagdan

Public Relations Representative (PR) - Jaimar John Sison

The escalation manager's job is to assess the risk of the incident and take a course of action based on the severity of the problem. Once the problem is analyzed, the escalation manager will assemble colleagues to mitigate the damage of the threat and attempt to patch up any security vulnerabilities in the code.

The legal representative handles all legal matters that rise from the incident. This can include employee representation, lawsuits from customers, and any other legal matters that concern the company. If sensitive information was leaked, the legal representative must work with the users to achieve a common ground of compensation.

The public relations representative's main job is to deliver the message of the incident to the public. The main goal is to explain the facts of the incident and what steps have been taken to fix the problem. As a public relations representative, they will need to make sure to not cause a mass wave of panic in the user base.

Group Email: chasem7@hawaii.edu

Incident Response:

- Analyze threat
- Determine necessary course of action (Real or fake problem)
- Assemble team needed to fix the problem

- Gather facts about the problem and have PR make an announcement once problem is stabilized
- Inform legal representative of problem
- Assess who is aware of the situation internally
- Construct a plan to compensate and recover from and losses caused by the intrusion

Final Security Review

FSR Rating: Passed FSR with exceptions

Based on several tests with the program, there are areas of the program that should be addressed to increase its security. With one test, a user tries to register for an online bank account and enters a password that does not meet the password requirements. In this case, even though the password was not acceptable, when the user tries to change the password within the registration page, he/she would find out that the username already exists. This flaw is also shown with the birth date. With our bank application, though, since new users do not have money in their accounts, there is not much to find out if the new account was hacked. After doing this test, we were able to figure out how to overcome this flaw in the program.

Upon further analysis using the tools built in with the IntelliJ IDE, the rating decision is further strengthened due to the fact that are code is not clean. The code achieves the necessary security measures, but the way that they are achieved is done in an inefficient manner. Though this poses no immediate risk, if the application were to grow, the code would have to be evaluated again to make sure that no overlapping bugs come into play which could possibly make the program vulnerable to intrusion or break altogether.

Release and Archive

Link to Final Release: https://github.com/UHrocks7/AnonymousBankApp/releases/tag/v1.0