A Secure Banking System

Atin Singhal
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
asing274@asu.edu

***Abstract*—This paper discusses the system design, functionalities and security features for the Secure Banking System (SBS) developed to fulfil the requirements for CSE 545 Software Security. The SBS allows secure online transactions, which are recorded in the Hyperledger Blockchain[1], and management of user accounts.**

***Index Terms*—secure banking system, security, hyperledger, blockchain, public key infrastructure**

I. Introduction

Billions of transactions happen daily and they are prone to security threats. A small security vulnerability in these transactions could potentially lead to a loss of millions of dollars. The Secure Banking System (SBS) developed, tackles these security threats by taking proactive measures and provides the users with a secure gateway for transactions. Multiple layers of security are required to make sure the banking systems are impervious to malicious attacks. For the same purpose, this SBS is equipped various security features such as Public Key Infrastructure, virtual keyboard to enter One Time Password (OTP), one active session per user[2], SQL Injection Prevention[3], role based access control, SSL/TSL(HTTPS), data encryption[4], capturing transactions in blockchain[1], etc.

Paper starts with the overview and functionalities that the SBS has, then it moves on to security features that were implemented. Next, it discusses my contributions to the project in detail and conclusion as an end note.

II. System Overview

SBS has various functionalities and caters to types of users:

- External users
- Internal users

External users (Individual Customers) each have at least one of the following types of accounts: checking, savings and credit card with common functions, such as fund transfer, debit and credit from user accounts. These users can also request for creation of additional accounts or schedule appointments with the bank.

Internal users further divide into Employee (Tier-1), Manager (Tier-2) and Administrator (Tier-3). Internal users perform operations on behalf of external users (Customers) and maintain their account. Tier-1 Employees have access to functionalities like viewing, creating and authorizing non-critical transactions, i.e. transactions below \$1000. They can also view customer accounts, issue cashier checks and handle fund transfer requests. Tier-2 Employees can view, create, modify and delete customer accounts, authorize transactions above \$1000 and initiate account modifications. Tier-3 Employee is the administrator and can view, create, modify & delete employee accounts. They can also authorize or decline employee requests and access system logs.

A. Implementation Details

1) The application is written in Java using Spring Framework[5].
2) Amazon Web Services (AWS) EC2 instance is used for deployment.
3) MySQL is used the database and was hosted as RDS using AWS.
4) Google SMTP servers are used to send OTP via e-mail.
5) JavaScript should be enabled for the application to be functional.

Fig.1 explains how information flows within the system. It indicates how data is utilized, processed and stored after each user action.

B. Security Features

This section discusses about the various security features that were implemented to make the banking system secure.

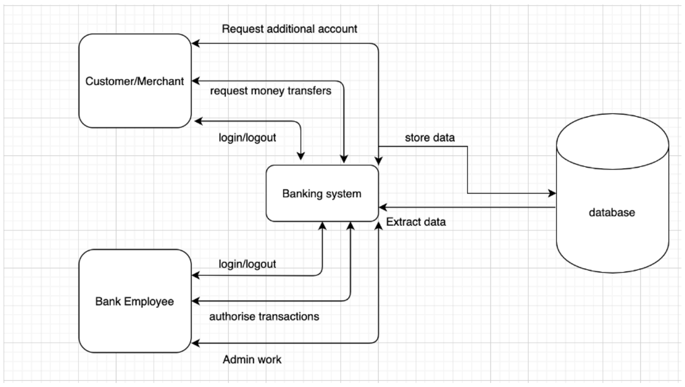

Figure 1: System Decomposition

1) One Time Password (OTP): OTP, also known as one-time pin or dynamic password is good for only one login session or action, on a computer system or other digital device. Our implementation incorporates 2FA by ensuring that the OTP requires access to an email a user has registered in our system with. To validate highly sensitive requests/transactions, the system must make sure that the request is originating from the real user. The assumption here is that only the real user would have access to the email to which the banking system sends the OTP. This OTP is tied to a user session and an action.

2) Secure Sockets Layer (SSL)[6]: SSL is an encryption-based Internet security protocol. This was implemented in our application by enforcing HTTPS only traffic to the AWS EC2 Instance. SSL encrypts data that is transmitted across the web. If anyone intercepts this data, they only see a meaningless sequence of characters that is extremely difficult to decrypt.

3) Public Key Infrastructure (PKI)[7]: A self-signed certificate is used by the server to identify itself to the clients. This stops other malicious attackers from impersonating as our banking application.

4) Malicious Login Controls: The app enforces locking of user accounts if there are more than 5 failed login attempts. If a user account gets locked, a Tier 2 Employee must be involved to unlock the users account. If 10 failed login attempts are detected by the system, the IP from which the requests originate will be blocked for 24 hours.

5) CSRF Tokens: A CSRF token is a unique, secret, unpredictable value that is generated by the server-side application and transmitted to the client in such a way that it is included in every form in the website. When a form is submitted, Spring-Security[2] verifies if this CSRF token is valid, and forwards the request to the controller only after confirmation. CSRF tokens can prevent CSRF attacks[3] by making it impossible for an attacker to construct a fully valid HTTP request suitable for feeding to a victim user.

6) Form Validations: There are limited type of forms in our banking application that input data into the database. The goal of validating the inputs of a form is to avoid XSS attacks on the victim. White-listing is easier than black- listing patterns. There are strict regex patterns for every kind of input the banking system accepts to avoid any kind of XSS inputs to the database. These checks are done both at the front-end and the back-end.

7) Hyperlegder[1] Integration: The secure banking system utilizes the blockchain technology to capture all valid transactions and record them on an immutable ledger. This immutability protects the users by preventing any form of tampering with the transaction records. The ledger also serves as a source of truth due to its immutable characteristics.

8) URL Protection: Various end points in our application are to be viewed by certain authorized users. So, these endpoints need to be protected. Roles are assigned to each user based on the type of the user. Spring-Security[2] is configured to declare the roles a URL can be accessed by.

9) Hashed Password Storage: Even with strong security measures in place, it is possible that the database could fall into the wrong hands. Hashing and salting sensitive data in the database can ensure that even if the database is acquired by a malicious attacker, the database is useless since the hashed data is meaningless to anyone without the knowledge of what the data actually is. The system uses BCrypt[4] hashing algorithm with 10 rounds. Even if this hashed password falls into the wrong hands, decryption is near impossible and gives the users and the bank enough time to take corrective measures.

III. My Contributions

Being the group leader, I was responsible for organizing meetings, submitting meeting minutes, drafting the work breakdown structure and having regular discussions with group members regarding the project status and implementations. Played a primary role in designing the misuse cases for the application.

I was mainly responsible for the security implementations, hyperledger[1] blockchain along with a few functional implementations. Worked on making skeleton web pages to perform easy navigation between pages, added style sheets in the login page and designed the error page for the application. Primarily responsible for parts of Web Security Configuration file which include configurations about user sessions, role-based access management etc.

Implemented functionalities including system logs, user-wise IP logging, maintaining user login history, virtual keyboard[8] for the OTP page. For security side, added PKI[7] applications, responsible for creating Public Key Certificate, enforcing HTTPS redirection[9]. Also created scripts for added security. Added blocking of back buttons, right click, copy/ cut/ paste functionality, text selection, etc. Used Hyperledger JavaSDK for integration with our application, which was further optimized by a few other group members, to make sure the application is light-weigth and can smoothly run on limited resources on an EC2 AWS instance. Responsible for creating smart contract for hyperledger platform and docker setup for the same.

I was also actively involved in the testing phase where we tested the application for functional as well as security bugs. This involved automated as well as manual testing. Tools used for automated testing include: Acunetix Web Vulnerability Scanner, Arachni, Postman, Burpsuite, Kali Linux Security tools, etc. Snyk and Github were actively used throughout the project to collaborate and continuously find & fix the dependency-related vulnerabilities.

IV. Knowledge & Skills Acquired

1) Figured out how to lead a group of eight members, break complex tasks into parts and steps & effectively plan and manage time.
2) Learned about Spring Framework and Spring Security.
3) Learned how to build a secure web application.
4) Learned about SSL certificates, XSS Scripting, SQL Injection and CSRF tokens during the development and testing phase.
5) Acquired knowledge on Software Development Life Cycle (SDLC).
6) Gained insights on application security testing.
7) Gained hands-on experience on Java development, AWS deployment, Hyperledger blockchain & Docker.
8) Learned to implement various security features and techniques.
9) Learned how to use collaboration tools like GitHub and Slack effectively.

V. Conclusion

While developing financial websites like banking system, it is very important to consider security from the first stage of the development cycle. It might not be possible to add security features afterwards and may costs lots of resources, including money and time. Secondly, the application must not rely on just one layer of security. We must incorporate multiple layers to make sure the application is secure even if one of the layer gets compromised. Lastly, role based authentication is very important in sensitive applications like banking systems.

VI. Team Members

- Atin Singhal (Leader)
- Shivank Tiwari (DL)
- Raj Buddhadev
- Ayush Ray
- Kangjian Ma
- Uttam Bhat
- Sriprashanth Ramamoorthy
- Amitabh Das

Acknowledgment

I would like to thank Professor Stephen S.Yau for such a wonderful opportunity and his valuable guidance throughout this project.

References

[1] Hyperledger. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-2.0/.
[2] Spring Security. [Online]. Available: https://spring.io/projects/spring-security.
[3] Open Web Application Security Project (OWASP). [Online]. Available: https://owasp.org/www-project-top-ten/.
[4] BCrypt Hashing Algorithm. [Online]. Available: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCrypt.html.
[5] Spring Boot Reference Documentation. [Online]. Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/.
[6] Secure Socket Layer. [Online]. Available: https://www.ssl.com/faqs/faq-what-is-ssl/.
[7] Public Key Infrastructure (PKI). [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/seccertenroll/public-key-infrastructure?redirectedfrom=MSDN.
[8] Virtual Keyboard- Mottie. [Online]. Available: https://mottie.github.io/Keyboard/.
[9] Post Redirect Get. [Online]. Available: https://en.ryte.com/wiki/Post-Redirect-Get.