



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

## Secure Coding

### Phase 5

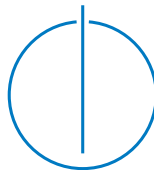
Team: 8

Members: Korbinian Würfl

Mai Ton Nu Cam

Vivek Sethia

Swathi Shyam Sunder



## Executive summary

In addition to fulfilling all the functional and non-functional requirements specified, SecureBank offers a secure solution for a Banking application, owing to the robust architecture.

The application is a one-stop solution featuring a web interface catering to the entire banking workflow, complete with a stand-alone Smart Card Simulator. Along with the feature-set of traditional banking, the application additionally comes with a set of dashboards for managers, summary reports for users and responsive design.

The application has faced extensive testing - both Black-box and White-box, in the hands of two able teams; as well as our own. The tests performed are prescribed by “Open Web Application Security Project (OWASP)”, which is a pioneer in the field of web application security. It has turned out that SecureBank sails through successfully. This has been confirmed by the absence of any deviations from application requirements, usability flaws and functional issues. The minor vulnerabilities that were discovered during the White-box testing phase have been fixed, with no stone left unturned.

True to its name, SecureBank is hence proven as a validated and verified product, that can be deployed in its current form as a real Banking application. It would also be hassle-free to incorporate any customizations to suit specific taste, given the ease of customizing the application.

# Contents

<b>Executive summary</b>	<b>ii</b>
<b>1 Time tracking</b>	<b>1</b>
1.1 Korbinian Würfl . . . . .	1
1.2 Mai Ton Nu Cam . . . . .	2
1.3 Vivek Sethia . . . . .	3
1.4 Swathi Shyam Sunder . . . . .	4
<b>2 Application Architecture</b>	<b>5</b>
2.1 Deployment . . . . .	5
2.2 Main Components . . . . .	6
2.3 External Libraries . . . . .	7
2.4 Smart Card Simulator . . . . .	8
2.4.1 TAN Generation and Working . . . . .	8
2.5 UML Design . . . . .	11
2.5.1 Class Hierarchy . . . . .	13
2.6 Database Schema . . . . .	14
<b>3 Security Measures</b>	<b>16</b>
3.1 SQL Injection . . . . .	16
3.1.1 Attack Description . . . . .	16
3.1.2 Countermeasure . . . . .	16
3.2 Integer Overflow . . . . .	17
3.2.1 Attack Description . . . . .	17
3.2.2 Countermeasure . . . . .	17
3.3 Privilege Escalation & Insecure Direct Object References . . . . .	18
3.3.1 Attack Description . . . . .	18
3.3.2 Countermeasure . . . . .	18
3.4 Cross-Site scripting . . . . .	19
3.4.1 Attack Description . . . . .	19
3.4.2 Countermeasure . . . . .	19

3.5	Cross-Site Request Forgery . . . . .	20
3.5.1	Attack Description . . . . .	20
3.5.2	Countermeasure . . . . .	20
3.6	Sensitive data sent via Unencrypted Channel . . . . .	21
3.6.1	Attack Description . . . . .	21
3.6.2	Countermeasure . . . . .	21
3.7	Session Variables, Fixation and Puzzling . . . . .	22
3.7.1	Attack Description . . . . .	22
3.7.2	Countermeasure . . . . .	22
3.8	Clickjacking . . . . .	23
3.8.1	Attack Description . . . . .	23
3.8.2	Countermeasure . . . . .	23
3.9	Brute Force Attacks . . . . .	24
3.9.1	Attack Description . . . . .	24
3.9.2	Countermeasure . . . . .	24
3.10	File-based Attacks . . . . .	25
3.10.1	Attack Description . . . . .	25
3.10.2	Countermeasure . . . . .	25
3.11	Directory Traversal/File Include Attacks . . . . .	26
3.11.1	Attack Description . . . . .	26
3.11.2	Countermeasure . . . . .	26
<b>4</b>	<b>Fixes</b>	<b>27</b>
4.1	SCS Exception due to incorrect Java version . . . . .	27
4.2	SCS Pin shown in Command Line . . . . .	28
4.3	Cookie Secure Flag . . . . .	29
4.4	Potential Buffer Overflow in C . . . . .	30
4.5	Guessable User Account . . . . .	31
4.6	Business Logic Data Validation . . . . .	31
4.7	Weak SSL/TLS Ciphers . . . . .	31
4.8	Error Codes . . . . .	32
<b>5</b>	<b>References</b>	<b>33</b>

# 1 Time tracking

## 1.1 Korbinian Würf

Task	Time in h
Diagrams	1.5
Video	15
Total	16.5

## 1.2 Mai Ton Nu Cam

Task	Time in h
Presentation Use Cases	2
Presentation Security Features	2
Attempt to fix Error Codes	2
Fixes in C Codes	2
Fixes for other vulnerabilites	1
Fixes report	1
Deliverables	1.5
Testing	1
Screenshots for video	0.5
<b>Total</b>	<b>13</b>

## 1.3 Vivek Sethia

Task	Time in h
Reporting Security Measure for SQL Injection	2.0
Reporting Security Measure for Integer Overflow	1.0
Reporting Security Measure for XSS Attacks	1.5
Reporting Security Measure for Session Fixation	2.0
Reporting Security Measure for Insecure Channel	1.5
Reporting components for deployment	2.0
Reporting UML Classes	2.0
Reporting Controller class hierarchy	2.0
Reporting usage of FPDF Library	1.00
Reporting Security Measure for File & Directory Traversal Attacks	1.0
Preparing screenshots for the fixes	0.5
Report checking & corrections	1.0
Final Testing of the application	1.5
<b>Total</b>	<b>19.00</b>

## 1.4 Swathi Shyam Sunder

Task	Time in h
Fixes for Smart Card Simulator	2.0
Reporting Fixes for SCS Java version flaw & Stack Trace	2.0
Reporting Smart Card Simulator	2.0
Preparing Class Diagram for Smart Card Simulator	0.5
Reporting Database Schema	2.0
Preparing ER Diagram for the Database	0.5
Reporting usage of external libraries	2.0
Reporting Security Measure for Brute Force attacks	1.0
Reporting Security Measure for Clickjacking	1.0
Reporting Security Measure for CSRF	1.5
Reporting Security Measure for Indirect Object References	1.0
Reporting Security Measure for File-based attacks	1.0
Preparing screenshots for the fixes	0.5
Writing Executive Summary for the Report	2.0
Overall Report corrections	1.0
<b>Total</b>	<b>20.00</b>



## 2 Application Architecture

### 2.1 Deployment

The main components of the application are the Web-based Banking component, the C parser and the Smart Card Simulator. Refer figure 2.1 for the deployment diagram, which depicts the architecture of the **SecureBank** application and its main components.

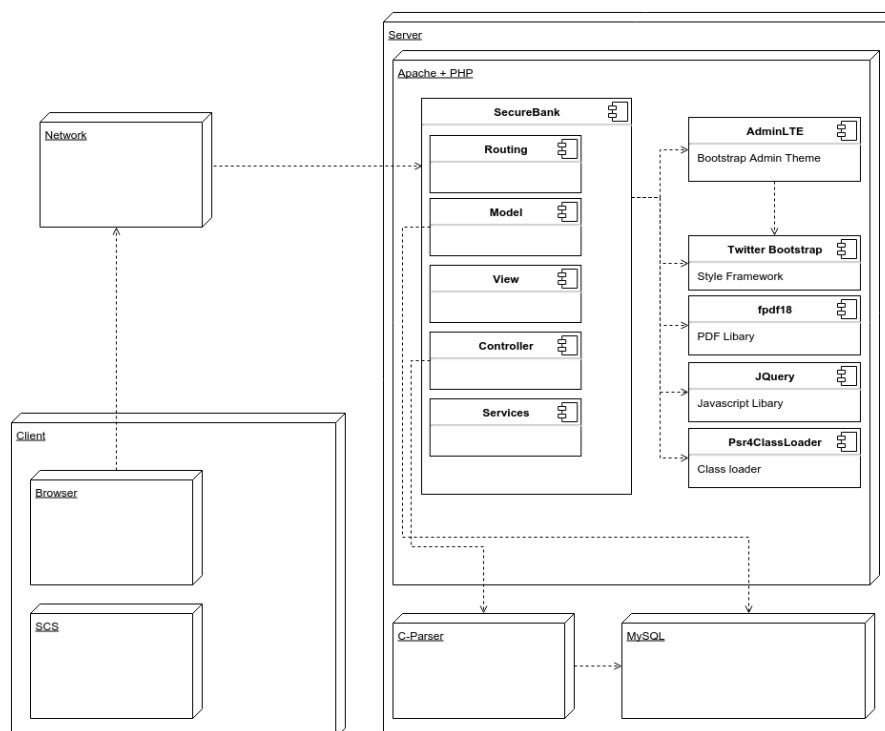


Figure 2.1: SecureBank Application Deployment Diagram

## 2.2 Main Components

The most important components of the application, as also depicted in figure 2.1 are detailed below.

- **MySQL** - This is the DBMS used in the application. Further details can be found in the section 2.6.
- **C Parser** - The C Parser is used in the application to support Batch transfers in the banking application, and it serves two main purposes.
  - **File parsing** - This part handles reading the file, basic data validation and memory allocation. Further, it invokes the functions for processing the transfers.
  - **Processing Bank Transfer** - This part is mainly responsible for executing database queries for processing account transfers. It also takes care of validations such as checking for exiting sender and recipient, sufficient funds and legitimate TAN codes.
- **Web Banking Application** - This is pre-dominantly the most important part of the application. It is the web interface, that provides all functions to Bank users - from Registration and Login to the complete Banking Work-flow. Extended functionality of the SCS can be taken advantage of as well, by downloading it from the web interface.
- **Smart Card Simulator** - This has been explained in complete detail in section 2.4.
- **External Libraries** - Section ?? describes in great detail, about the dependencies on external packages and frameworks.

## 2.3 External Libraries

1. **Admin LTE** - Admin LTE is a free Bootstrap dashboard theme from *almsaeedstudio*. It is fully responsive and built on top of Bootstrap 3. Typical interface features such as Registration & Login pages, Dashboard, Navigation components have been incorporated in the banking application.
2. **Twitter Bootstrap** - *Bootstrap* is a front-end open-source framework, that contains HTML and CSS-based templates for forms, buttons and other UI components. It aims to ease the development of dynamic websites and web applications. In the application, Bootstrap has been mainly used for styling purposes. Pre-defined styles for various components have been customized accordingly to suit our own requirements.
3. **jQuery** - *jQuery* is a JavaScript library which is used to simplify the client-side scripting of HTML. The usage in the application is minimal and is limited mainly to customization of properties of the *Data-Table* component and operations related to the *Modal Dialogs*.
4. **FPDF** - *FPDF* is a PHP class which allows to generate PDF files with pure PHP, that is to say without using the PDFlib library. The generation speed of the document is less than with PDFlib. Most features of the library are utilized effectively, be it for layouting, formatting or for capabilities such as file locking with password. This specific feature has been used to generate the PDF containing the secure TANs for the customer.
5. **PSR-4 ClassLoader** - This PHP component is provided by *Symfony*. The *ClassLoader* component provides tools to auto-load the classes and cache their locations for performance. Whenever a class that has not been required or included yet; is referenced, PHP uses the auto-loading mechanism to delegate the loading of a file defining the class.

The *PSR-4 Class Loader* loads classes that follow the PSR-4 class naming standard. This PSR describes a specification for auto-loading classes from file paths. It is fully inter-operable, and can be used in addition to any other auto-loading specification, including *PSR-0*. This PSR also describes where to place files that will be auto-loaded according to the specification.

## 2.4 Smart Card Simulator

The Smart Card Simulator is a tool to generate transaction codes (also referred to as TANs), implemented in Java. This tool is available for download to users who choose SCS as the preferred way to receive TANs, at the time of their registration with the Bank. The SCS relies mainly on the PIN, that is sent to the user through an e-mail after registration is approved by an employee. The SCS is not personalized i.e., it is not unique based on the user. Also, there are no hard-coded passwords or tokens for user identification stored anywhere in the JAR. This is a good thing as an attacker cannot get hold of any user-specific information by just getting access to the SCS.

Once the SCS has been downloaded by the user, it can be used to generate TANs. The SCS works in two modes thereby supporting Single Transaction and Batch Transactions respectively.

- Single Transaction: The SCS takes 3 inputs - Account ID of the Recipient, Amount to be transferred and the PIN for the SCS.
- Batch Transactions: The SCS takes only 2 inputs - Batch file containing the transaction details in the prescribed format and the PIN for the SCS.

Refer figure 2.2 for the view of the Smart Card Simulator.

The SCS generates the TAN after performing basic validations such as entry of mandatory fields and format of inputs. Validations such as existent recipient, valid pin, sufficient balance etc. are handled at the PHP end at the time of actual transfer.

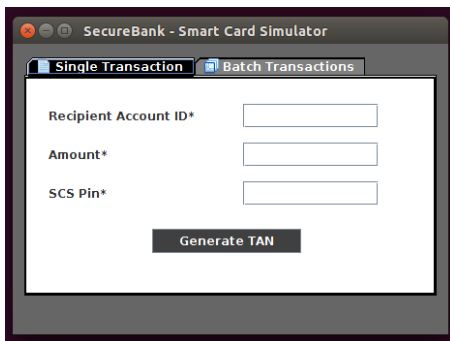
Java code is organized mainly in 2 classes - `SmartCardSimulator.java` and `TanGenerator.java`. **SmartCardSimulator** handles the display of the interface and the TAN generation logic is handled by the **TanGenerator**. Refer figure 2.3 for the complete Class diagram of the Smart Card Simulator.

### 2.4.1 TAN Generation and Working

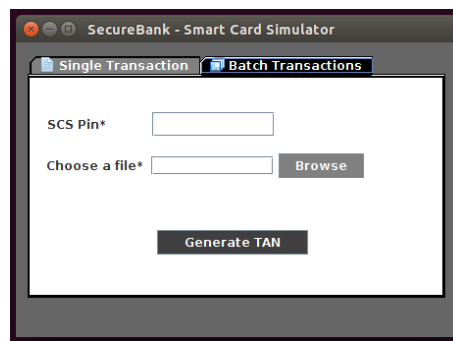
The TanGenerator removes all white-spaces from the inputs, and concatenates the current time in milliseconds divided by 10<sup>5</sup> (=100 seconds) to the input. This input string is then hashed with the `SHA512` hashing algorithm. The first 15 characters of this hash are the tan number for the corresponding transaction. The purpose of using the current time is to set an expiration time for the generated TAN. This makes the code unusable after 100 seconds. There is a minor possibility of client and server time not

being in sync which could lead to the TAN becoming unusable.

The TAN generated by the SCS is used in the online transfer form. At the PHP side, a TAN is generated using a similar algorithm as in Java, using the target and amount entered in the Web UI and the SCS pin of the user fetched from the database. If the TAN(from SCS) entered by the user matches this TAN, payment is processed successfully.



(a) View for Single Transaction



(b) View for Batch Transaction

Figure 2.2: Smart Card Simulator

## 2 Application Architecture

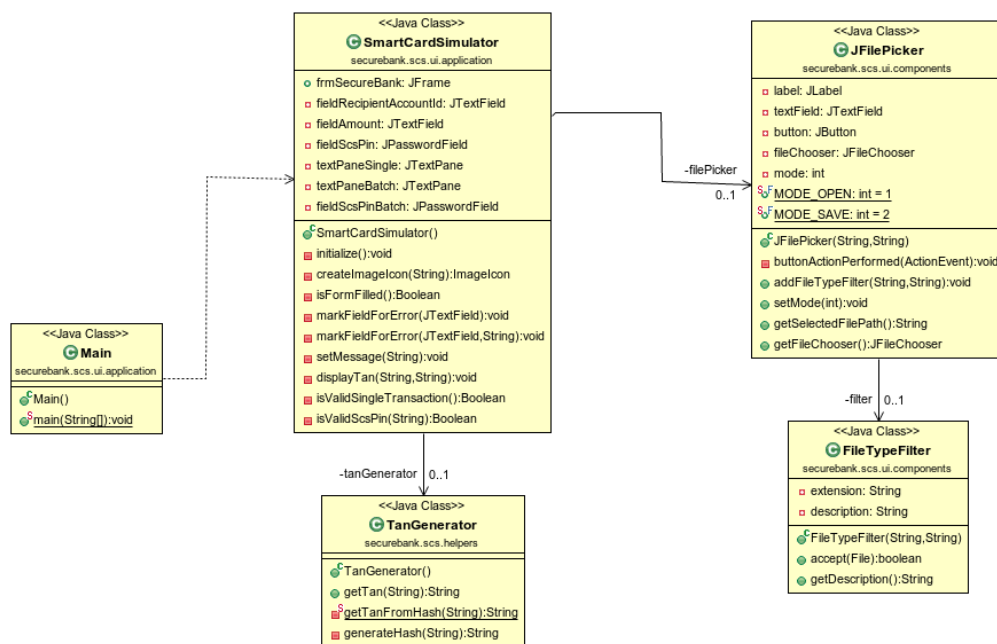


Figure 2.3: Class Diagram of Smart Card Simulator

## 2.5 UML Design

The UML Diagram for the web application is depicted in the figure 2.4 below. The diagram presents an extension of the **Model-View-Controller** architecture pattern.

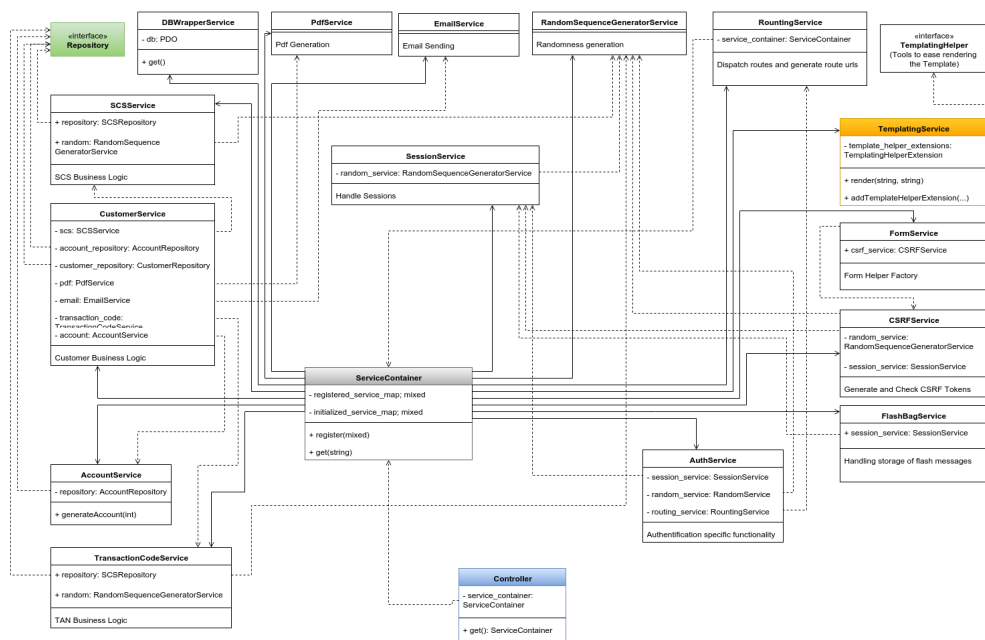


Figure 2.4: UML Diagram for the SecureBank application

The main categories of classes in the application are:

- **Controller** - These classes are responsible for all operations concerning the working. *Example: CustomerController handles all customer-related functionalities.* A main Controller is defined and all other controllers inherit from the base controller, adding additional and specific functionality.
- **Model** - These classes represent the format of specific data objects. *Example: Customer Model defines the structure of a Customer object.*
- **Repository** - These classes handle the interaction with the database and contain the queries. Similar to the case of controllers, a base Repository is defined and all other repositories inherit from it.

All common functionality such as *add*, *update*, *get* and *find* are implemented in the base repository. So queries for these operations for any tables, need not be explicitly written. Rather, the functions need to be merely invoked with the required parameters. Extensive queries such as **JOINS** are written in the respective repositories, as per requirement.

- **Service** - The *Service* classes offer common functionality to be used across various other classes. Examples include *CSRFService*, *SessionService*, *EmailService*, *PdfService*, *RandomSequenceGeneratorService* etc.
- **Template** - *Template* represents the **View**. So all templates are files containing HTML code, to be rendered on the web. Adhering to good coding practices, all client-side scripting is kept out of the HTML and handled separately in a JavaScript file.



### 2.5.1 Class Hierarchy

As described in section 2.5, the *Controller*, *Repository*, *Model* classes extensively adhere to Inheritance. The concept has been depicted in the figure 2.5 for the **Controller** classes.

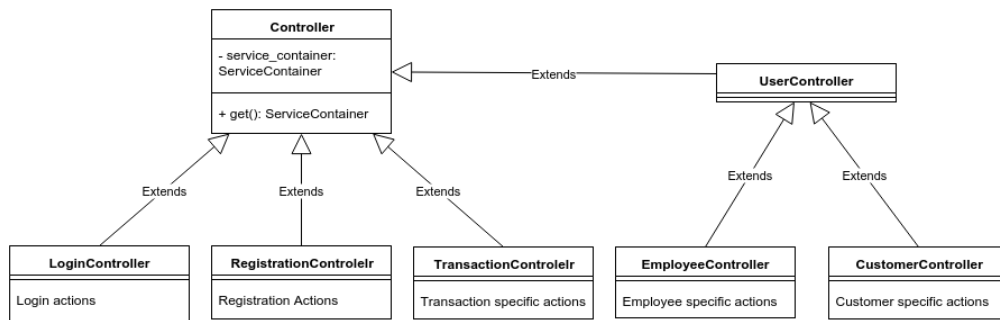


Figure 2.5: Hierarchy of Controller classes

**Controller** is the Base class for all controllers in the application. All other controllers such as the *UserController*, *LoginController*, *RegistrationController* and *TransactionController* inherit from the base class.

On a top level, Customer and Employee objects are Users. So all functionality that is applicable to a *User* is also applicable to a *Customer* and an *Employee*. Hence the *CustomerController* and *EmployeeController* inherit directly from the *UserController*.

## 2.6 Database Schema

The DBMS used for the **SecureBank** application is MySQL. The schema consists of 6 entities or tables. They are

1. **TBL\_CUSTOMER** - This table contains all information about the Customers. These include Profile information, details about login attempts and status of the Customer account.
2. **TBL\_EMPLOYEE** - This table contains all information about the Employees, such as profile information and status.
3. **TBL\_ACCOUNT** - This table consists of details about the Bank Account details of the Customers, such as *Account number* and *Balance*. The referential constraints are implemented in a way that a Customer can hold multiple Accounts in the Bank.
4. **TBL\_TRANSACTION** - This table holds details of the Transactions performed by all Customers in the Bank. Complete details about the payer and payee(recipient) are recorded here. The constraints are defined so that both parties are account holders of the *SecureBank*.
5. **TBL\_TRANSACTION\_CODE** - This table contains the Transaction codes (also referred to as *TANs*) corresponding to each Customer. The information about whether the code is already used is also present here.
6. **TBL\_SCS** - This table consists of the PINs for the Smart Card Simulator, corresponding to each Customer.

Data-types and lengths for all the fields in the tables are carefully chosen, taking into account the purpose and usage in the application. Refer figure 2.6 for the complete Entity-Relationship diagram of the **SecureBank** application.

In the web application, all queries to the MySQL database are through the PDO interface and in the C part, MySQL prepared statements are used. These mechanisms ensure better performance, safer & cleaner code along with protection against injection vulnerabilities.

In addition to the usual *Insert*, *Update* and *Update* statements, *Transactions* are also used, mainly in the processing of account transfers. This is due to the fact that a single transfer is associated with multiple actions such as deduction of funds from the sender and depositing funds in the recipient, which involves operations on multiple tables. *Commit* and *Rollback* features of Transactions are employed in such scenarios.



## 3 Security Measures

### 3.1 SQL Injection

#### 3.1.1 Attack Description

A SQL injection attack consists of insertion of a SQL query via the input data from the client to the application. This is done in order to affect the execution of predefined SQL commands. A successful SQL injection exploit can read sensitive data from the database, modify database data i.e., **Insert**, **Update** or **Delete** table data, execute administrative operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases, even issue commands to the operating system.

#### 3.1.2 Countermeasure

In order to safeguard the application from SQL injections, following measures are undertaken in the **SecureBank** application.

- **Sanitization of user inputs before execution of queries** - *Helper/ValidationHelper.php* and *Helper/SanitizationHelper.php* implement functions for validation and sanitization of user input. These functions are used whenever forms are rendered in the web interface.
- **Usage of PDO statements on the PHP side** - All SQL queries are executed via a base class called *Model/Repository.php*, which uses the *PDO::prepare* function. All other repositories inherit this base repository and also use *PDO::prepare* for any additional functions. This ensures that all queries on the database are escaped.
- **Usage of MySQL Prepared statements in the C Parser** - A similar code structure is followed in the C code, where all main database operations are handled in *repository.c*. All other repositories make use of functionality from the repository and add extended functionality, also implemented using MySQL prepared statements.

## 3.2 Integer Overflow

### 3.2.1 Attack Description

An integer overflow condition exists when an integer, which has not been properly sanity checked, is used in the determination of an offset or size for memory allocation, copying, concatenation, or similarly. If the integer in question is incremented past the maximum possible value, it may wrap to become a very small, or negative number, therefore providing a very incorrect value.

### 3.2.2 Countermeasure

In order to safeguard the application from Integer overflows, following measures are undertaken in the **SecureBank** application.

- **Sanitization of user inputs before further processing** - *Helper/ValidationHelper.php* and *Helper/SanitizationHelper.php* implement functions for validation and sanitization of user input. These functions are used whenever forms are rendered in the web interface.
- **Data validation** - This is done by enforcing appropriate checks to validate data during specific operations such as Account Balance Initialization by an employee, Single Transaction and also Batch transfers. This has been done both in PHP and C code.

## 3.3 Privilege Escalation & Insecure Direct Object References

### 3.3.1 Attack Description

Privilege escalation occurs when a user gets access to more resources or functionality than they are normally allowed, and such elevation or changes should have been prevented by the application. Vertical escalation is when it is possible to access resources granted to more privileged accounts like acquiring administrator privileges and Horizontal escalation is when it is possible to access resources granted to a similarly configured account like accessing information related to a different user.

Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability, attackers can bypass authorization and access resources in the system directly, for example database entries belonging to other users, files in the system and more. This is caused by the fact that the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks.

### 3.3.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **Authorization checks** - This is implemented by defining a clear distinction of privileges among the users. Operations are distinguished as Administrator, Employee or Customer actions. Each controller function starts with an access right check, that checks for the static and contextual privileges of the current user. In scenarios where the current user ID is used in code, it is taken from the session and not from GET or POST parameters. This is a countermeasure against vertical privilege escalation.
- **Page access independent from User input** - User related pages, e.g. account information page, do not depend on user input. Specifically page access does not depend on GET or POST parameters, e.g. `userid=1` or something similar. This is a countermeasure against horizontal privilege escalation.

## 3.4 Cross-Site scripting

### 3.4.1 Attack Description

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different unsuspecting end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

### 3.4.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **Sanitization of user inputs before further processing** - *Helper/ValidationHelper.php* and *Helper/SanitizationHelper.php* implement functions for validation and sanitization of user input. PHP functions such as *htmlspecialchars*, *filter\_var* and *preg\_match* are employed for input sanitization; leading to HTML tags being rendered as plain text. These *Helper* classes are used whenever forms are rendered in the web interface.

## 3.5 Cross-Site Request Forgery

### 3.5.1 Attack Description

Cross-Site Request Forgery (CSRF) is an attack that forces an authenticated end user to execute unwanted actions on a web application. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. By sending a link via email, an attacker may trick the users of the application into executing malicious actions. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds etc. If the victim is an administrator or employee, CSRF can compromise the functionality of the entire web application.

### 3.5.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **CSRF Tokens** - This has been implemented in all HTML forms through a hidden input field for a CSRF token. Furthermore, the CSRF token is saved in the PHP session. The files *Service/CSRFService.php* and *Helper/templatingForm Extension.php* handle the logic for generation of CSRF tokens and their inclusion in every HTML form. The CSRF token is unique for each PHP session.



## 3.6 Sensitive data sent via Unencrypted Channel

### 3.6.1 Attack Description

Sensitive data must be protected when it is transmitted through the network. Some examples for sensitive data are Credentials, PINs, Session identifiers, Tokens, Cookies etc. If the application transmits sensitive information via unencrypted channels like **HTTP**, it is considered a security risk.

### 3.6.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **Usage of HTTPS** - It has been found that all accesses to the application are via *HTTPS* i.e., secure **HTTP** connections. Unencrypted traffic over *HTTP* is also redirected to the encrypted version of the webpage.
- **Usage of HSTS** - The HSTS (HTTP Strict Transport Security) header for *Strict-Transport-Security* is set to `max-age= 60000; includeSubDomains`. The configuration details for SSL and HSTS are found in the file *secure-bank.conf*.

## 3.7 Session Variables, Fixation and Puzzling

### 3.7.1 Attack Description

Session Fixation is an attack that permits an attacker to hijack a valid user session. The attack explores a limitation in the way the web application manages the session ID. When authenticating a user, it doesn't assign a new session ID, making it possible to use an existent session ID. The attack consists of obtaining a valid session ID, inducing a user to authenticate him/herself with that session ID, and then hijacking the user-validated session by the knowledge of the used session ID. The attacker has to provide a legitimate Web application session ID and try to make the victim's browser use it.

This is generally exploitable due to exposed session variables. The session tokens such as Cookie, SessionID, Hidden Field; if exposed, will usually enable an attacker to impersonate a victim and access the application illegitimately. It is important that they are protected from eavesdropping at all times, particularly while in transit between the client browser and the application servers.

### 3.7.2 Countermeasure

In order to safeguard the application from vulnerabilities related to Session variables, following measures are undertaken in the **SecureBank** application.

- **Usage of single Session cookie** - By using the same session cookie *PHPSESSID* everywhere in the application, Session overloading is prevented. The session variables are set in the file *SessionService.php* which is invoked once during login.
- **Secure Session Cookie** The session cookie has the *secure* and *httponly* flags set in the file *Service/SessionService.php*. This eliminates the possibility of setting the cookie from client side Javascript or other methods.
- **Session Timeout** - The Session is terminated by PHP after the standard value of 1440s inactivity. This has been implemented in the file *Service/SessionService.php* where no explicit Session timeout is specified.

## 3.8 Clickjacking

### 3.8.1 Attack Description

Clickjacking is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is “hijacking” clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both. An attacker could also make a user transfer money to the attacker without the user noticing it.

Using similar techniques, a user can be led to believe that the credentials are being typed in the bank account, but are instead typing into an invisible frame controlled by the attacker.

### 3.8.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **Prevention of site embedding** - Clickjacking is denied by setting the header *X-Frame-Options* to *DENY* in the *src/.htaccess* file. This ensures that the website cannot be embedded in an *iframe*.

## 3.9 Brute Force Attacks

### 3.9.1 Attack Description

A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until the one correct combination that works is discovered. Though it is always possible for an attacker to discover a password through a brute-force attack, the downside is that it could take years to find it. The possible combinations of passwords could increase enormously depending on the length and complexity.

### 3.9.2 Countermeasure

In order to safeguard the application from Brute force attacks during login, following measures are undertaken in the **SecureBank** application.

- **Enforcement of Strong Password** - There is a restriction on the choice of passwords during registration. User needs to enter minimum 6 characters which should contain atleast one uppercase character, one lowercase and one number. This reveals that passwords of users cannot be cracked easily.
- **Password Encryption** - Before saving the password in the database, it has been encrypted using the *crypt* function based on a random salt. So even for users having the same password, the database entries will be different.
- **Lockout Mechanism** - Logging in with a false password can be repeated 5 times before the account is locked. In the file *Auth/DBAuthProvider.php*, the account is checked for the number of login attempts. If it is greater than or equal to 4, it is set to 5 and the account is locked for 60 minutes.

## 3.10 File-based Attacks

### 3.10.1 Attack Description

File based attacks could be due to upload of malicious files or unexpected file types. The option to upload information is common to most applications. To reduce the risk, we may only accept certain file extensions and reject most others, but attackers are able to encapsulate malicious code into inert file types.

Vulnerabilities related to the uploading of malicious files is unique as these files can easily be rejected through including business logic that will scan files during the upload process and reject those perceived as malicious.

The application may allow the upload of malicious files that include exploits or shell-code without submitting them to malicious file scanning. Malicious files could be detected and stopped at various points of the application architecture.

### 3.10.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **Rejection of unexpected File Types** - The application only allows *.txt* files to be uploaded while performing Batch Transactions using the web interface. The file type is checked from the meta-data of the file and non-matching types are immediately rejected.
- **Restriction on File Size** - The maximum size of the uploaded file is restricted to *1MB*. This way, the file is neither read nor processed any further.
- **Random naming of uploaded files** - Before processing the uploaded files further, they are copied to the server with a unique randomly generated name, in order to prevent any attacks through file name. This also handles cases where same file names could cause concurrency issues.
- **Deletion of uploaded files** - After processing the uploaded file, the file is deleted from the server, without persisting it. This ensures that the application is safeguarded from any extended attacks due to the presence of the file.

## 3.11 Directory Traversal/File Include Attacks

### 3.11.1 Attack Description

Using and managing files is common to web applications. Traditionally, web servers and web applications implement authentication mechanisms to control access to files and resources. Web servers try to confine files related to users inside a “root directory” or “web document root”, which represents a physical directory on the file system. Users have to consider this directory as the base directory into the hierarchical structure of the web application.

The purpose of defining privileges is to identify which users or groups are supposed to be able to access, modify, or execute a specific file on the server. These mechanisms are designed to prevent malicious users from accessing sensitive files or to avoid the execution of system commands.

By exploiting this kind of vulnerability, an attacker is able to read directories or files which are normally inaccessible, access data outside the web document root, execute arbitrary code or system commands or include scripts and other kinds of files from external websites.

### 3.11.2 Countermeasure

In order to safeguard the application from this vulnerability, following measures are undertaken in the **SecureBank** application.

- **Disabling Directory Listing** - This has been done by setting *Options -Indexes* in the *src/.htaccess* file. Hence it is not possible to access any source code or download files by accessing files/folders. The detailed configuration options can also be found in the configuration file for the application - *secure-bank.conf*.

## 4 Fixes

In phase 4, Team 13 performed White-box testing of our application. In their findings, they mentioned six issues to be fixed. They are as stated below.

### 4.1 SCS Exception due to incorrect Java version

This was a major flaw, as the Smart Card Simulator worked on Java version 1.8 and the version of Java on the Virtual Machine was 1.7. Though the SCS works without any issue after update of Java version, we considered to fix it anyway. The fix involved making changes to the project properties for compatibility to version Java 1.7, before exporting the JAR. This was a minor change and required no modifications in code. Refer figure 4.1.

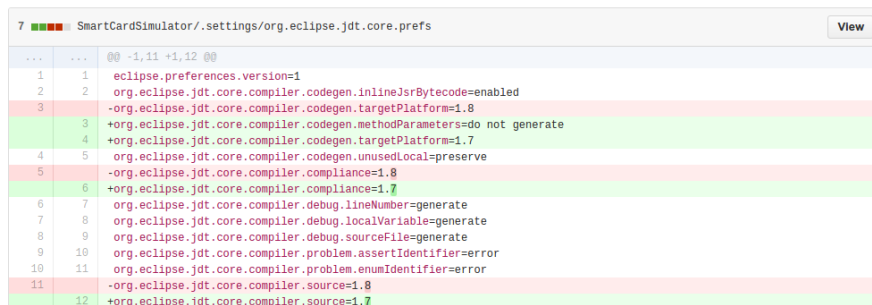


Figure 4.1: Project properties change to fix SCS Java version issue

## 4.2 SCS Pin shown in Command Line

While using the Smart Card Simluator in Linux environments, the SCS Pin was being printed to the command line, after clicking the *Generate Tan* button. This is a security flaw and can compromise privacy. This has been fixed and all the stack traces have been modified to simple print statements. Refer figure 4.2 for the code change.

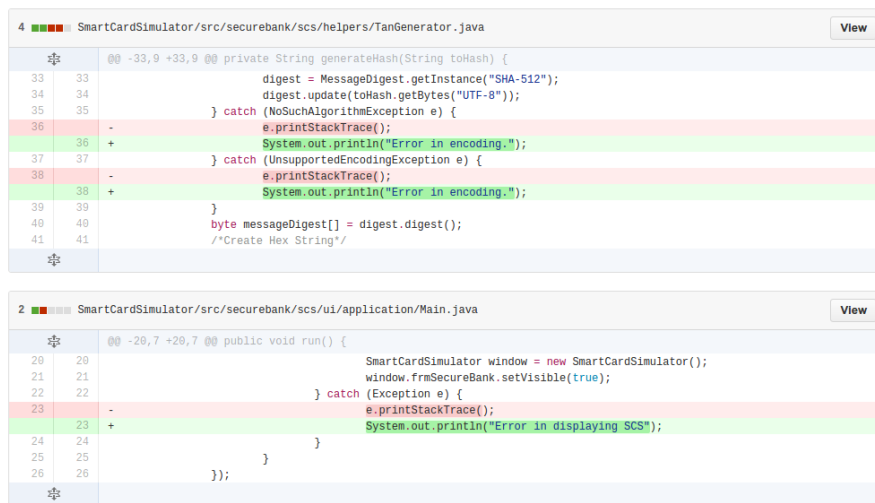



Figure 4.2: Code change to fix SCS Pin shown



### 4.3 Cookie Secure Flag

In their findings from the White-box testing, Team 13 had pointed out that the secure flag for the PHP session cookie was dependent on whether *HTTPS* was used or not. Since we enforce the use of *HTTPS*, the issue was a minor one. Nevertheless, we fixed it by always setting the secure flag. The fix is in the file `src/Service/SessionService.php`. Refer 4.3 for the code change.



```

8  src/Service/SessionService.php
@@ -17,9 +17,8 @@ class SessionService {
17 17      * @param int $limit The lifetime of the Session Cookie
18 18      * @param string $path The domain path where the session is active
19 19      * @param string $domain The domain where the session is active
20 20      * @param boolean $secure Only set the Session Cookie via https
21 20      */
22 21      function __construct(RandomSequenceGeneratorService $random_service, $name="Main", $limit = 0, $path = '/', $domain="") {
23 22          $this->random_service = $random_service;
24 23          // Prevent a session from being initialized twice
25 24          if(in_array($name, self::$session_lock)) throw new \Exception("Session with '$name' cannot be started twice");
@@ -28,11 +27,8 @@ function __construct(RandomSequenceGeneratorService $random_service, $name="Main
28 27          // Set the cookie name
29 28          session_name($name . '_session');
30 29
31 30          // Set SSL level
32 31          $https = isset($secure) ? $secure : isset($_SERVER['HTTPS']) ? 'on' : 'off';
33 32
34 33          // Set session cookie options
35 34          session_set_cookie_params($limit, $path, $domain, $https, true);
36 35          session_set_cookie_params($limit, $path, $domain, true, true);
37 36          session_start();
38 37
39 38          // Make sure the session hasn't expired, and destroy it if it has

```

Figure 4.3: Code change to fix issue with cookie attributes

## 4.4 Potential Buffer Overflow in C

There was a potential buffer overflow in our text-parser due to the use of `strcpy` without checking the length. This has been fixed in `textparser/transaction_repo.c`. Refer ?? for the code change.

```

24 ■■■■■ textparser/transaction_repo.c View
128 128      int_data[3] = is_rejected;
129 129      int_data[4] = is_closed;
130 130
131 131      strncpy(str_data[0], customerName, strlen(str_data[0]));
132 132      if (strlen(customerName) < sizeof(str_data[0])) {
133 133          strcpy(str_data[0], customerName);
134 134      } else {
135 135          strncpy(str_data[0], customerName, sizeof(str_data[0]));
136 136          str_data[0][-1] = '\0';
137 137
138 138      parameter_length[2] = strlen(str_data[0]);
139 139      strncpy(str_data[1], toAccountName, strlen(str_data[1]));
140 140
141 141      if (strlen(toAccountName) < sizeof(str_data[1])) {
142 142          strcpy(str_data[1], toAccountName);
143 143      } else {
144 144          strncpy(str_data[1], toAccountName, sizeof(str_data[1]));
145 145          str_data[1][-1] = '\0';
146 146
147 147      parameter_length[4] = strlen(str_data[1]);
148 148      strncpy(str_data[2], remarks, strlen(str_data[2]));
149 149
150 150      if (strlen(remarks) < sizeof(str_data[2])) {
151 151          strcpy(str_data[2], remarks);
152 152      } else {
153 153          strncpy(str_data[2], remarks, sizeof(str_data[2]));
154 154          str_data[2][-1] = '\0';
155 155
156 156      parameter_length[6] = strlen(str_data[2]);
157 157
158 158      float_data[0] = amount;
159 159
160 160      /*set the current date-time in the time-stamp structure*/
  
```

Figure 4.4: Code change to fix Buffer overflow

## 4.5 Guessable User Account

This was a minor issue, where-in different error messages were displayed to the user upon failed login. The error message is now made consistent to show “Either the e-mail or the password is wrong”. So it is not possible to guess whether the account does not exist or the password is wrong. The code change for this fix happened to just be modification of the message, as can be seen in the figure 4.5.



```
2 src/Controller/LoginController.php
View

function processLogin ($request) {
    break;

    case 'UserNotFoundException':
        $this->get("flash_bag")->add("Login failed", "There is no account with this e-mail.", "error");
        $this->get("flash_bag")->add("Login failed", "Either the e-mail or the password is wrong.", "error");
        break;

    case 'LoginFailedException':
```

Figure 4.5: Code change to fix Guessable User Account

## 4.6 Business Logic Data Validation

We do not consider this as an issue, as the current implementation is similar to the real-world working. In real working, the account name does not matter and does not have to match the account number used at the time of transaction. The only relevant factor is the *Account Number* of the recipient. If the transfers were performed through the addition of beneficiaries, then the concept of *Recipient Name* would make more sense. However, since we do not have this implementation, this could be termed as an improvement for the future.

## 4.7 Weak SSL/TLS Ciphers

This was resolved by prohibiting the use of old SSL versions. For that the line `SSLProtocol all -SSLv2 -SSLv3` was added to the virtual host configuration.

## 4.8 Error Codes

This would have been resolved by using custom error pages. We tried to fix this by adding `ErrorDocument <code> <custom error page>` to our `.htaccess` file, but it did not produce the expected result. Therefore, this issue remains unfixed.

## 5 References

- **OWASP Top 10 Vulnerabilities** - [https://www.owasp.org/index.php/Top\\_10\\_2013-Top\\_10](https://www.owasp.org/index.php/Top_10_2013-Top_10)
- **Admin LTE** - <https://almsaeedstudio.com/>
- **Bootstrap** <http://getbootstrap.com/>
- **jQuery** <https://jquery.com/>
- **FPDF** - <http://www.fpdf.org/>
- **PSR-4 Class Loader** - <http://www.php-fig.org/psr/psr-4/>