ErieGarbage Customer Manager
Coding and Testing Document
Version 1.1

TEAM MEMBERS

Name	Student ID
Allison Steinmetz	989643309
Mason Toy	920754627
Daniel Lopez	961382718

Revision History

Date	Version	Description	Author
11/29/16	1.0	Creating Base Templates	Mason
12/2/16	1.0	Set up document. Updated Class Diagram and section 3.2.	Allison
12/3/16	1.0	Static Analysis, Modifying sections and finishing sections.	Mason, Allison
12/4/16	1.0	Updated Class Diagram. Completed sections 1 and 3. Updated sections 2 and 4.1. Finishing Static Analysis	Allison, Mason, Daniel
12/5/16	1.0	Added information for section 4 and modified static analysis	Allison, Daniel
12/7/16	1.1	Updated Section 3.2 and added section 3.3	Allison
12/8/16	1.1	Completed section 3.3 and update class diagram	Allison, Mason, Daniel
12/9/16	1.1	Updated class diagram and added descriptions	Allison, Daniel

Table of Contents

Coding and Testing Document

1. Introduction

1.1 Purpose

The purpose of this document is to present in detail the coding and testing results for ErieGarbage Customer Manager. The coding part focuses on the static analysis of the code and documentation of the code including the class diagram and the documented security information on each class and its methods. The testing portion consists of the approaches used, the bugs found by testing and how these bugs were dealt with.

1.2 Scope

The Coding and Testing (CT) Document will detail the coding design and testing done on the ErieGarbage Customer Manager. The classes of the application are outlined with security analysis. The methods of testing and example test cases and their reports are shown.

1.3 Definitions, Acronyms, and Abbreviations

1.3.1 Definitions

Encrypt: To convert data into cipher text, with purpose of controlling access to it.
Decrypt: Convert cipher text to plain data, releasing access control.

1.3.2 Acronyms and Abbreviations

GUI – Graphical User Interface
SRS – Software Requirements and Specifications
Admin – Administrator
HTML – Hypertext Markup Language
PaS – Platform as a Service
CIA – Confidentiality, Integrity, Availability
RUP – Rational Unified Process
EGCM – ErieGarbage Customer Management
XSS – Cross Site Scripting
DoS – Denial of Service
MITM – Man in the Middle (Attack)

1.4 References

1.5 Overview

The introduction of the document provides an overview of the entire document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and an overview of this document.

The remainder of this document is split into three main parts: code static analysis, code documentation and testing results. The static analysis (section 2) give an overview of the results of the code's static analysis. The code documentation (section 3) outlines the class diagram consisting of all of the classes

with their variables and methods, as well as the security analysis for each class. Finally, the testing results (section 4) covers the methods of testing used for the EGCM and the results of the tests.

2. Static Analysis

After doing a bit of research on Static Security Analysis tools we found the application VisualCodeGrepper. This application supports a wide range of programing languages including C++, Java, SQL, C# and most importantly for this project PHP. After running a quick run on all of the PHP files there was one flaw generated for the code at this time, this is shown in Figure 2.1. The application tells the user what kind of security risks are associated and what line in what file the risk is generated from.

Priority	Severity	Title	Description
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
6	Suspicious ...	Comment Indicates Potentially Unfinished C...	The comment includes some wording which indicates that the developer regards it as unfinished or does not trust it to work correctly.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
4	Standard	fopen	
5	Low	Variable Used as FileName	The application appears to use a variable name in order to define a filename used by the application. It is unclear whether this variable can be controlled
4	Standard	fopen	
5	Low	Variable Used as FileName	The application appears to use a variable name in order to define a filename used by the application. It is unclear whether this variable can be controlled
5	Low	Variable Used as FileName	The application appears to use a variable name in order to define a filename used by the application. It is unclear whether this variable can be controlled
6	Suspicious ...	Comment Indicates Potentially Unfinished C...	The comment includes some wording which indicates that the developer regards it as unfinished or does not trust it to work correctly.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	md5	MD5 Hashing algorithm.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.
3	Medium	Potential XSS	The application appears to reflect data to the screen with no apparent validation or sanitisation. It was not clear if this variable is controlled by the user.

Figure 2.1

The static analysis resulted in a summary shown in Figure 2.1. The highest priority risks of level six, indicate potentially unfinished code given that the tool encountered a comment, yet there is no absence of needed code in the function. There are other warnings regarding utilizing a variable name for fopen, which can lead to security concerns. However, given the implementation, the function that provides that functionality is private and only accessible to the DatabaseController, which minimizes its risk.

The rest of warnings include the md5 hashing algorithm, which is utilized for hashing authentication values, and potential XSS for outputting unvalidated variables using echo. The hashing issue can be solved by utilizing a more secure hashing algorithm, such as SHA-1. Secondly, since the variables pointed out by VisualCodeGrepper are mostly static HTML templates for the layout, these should be declared immutable.

Overall code (2016 lines)

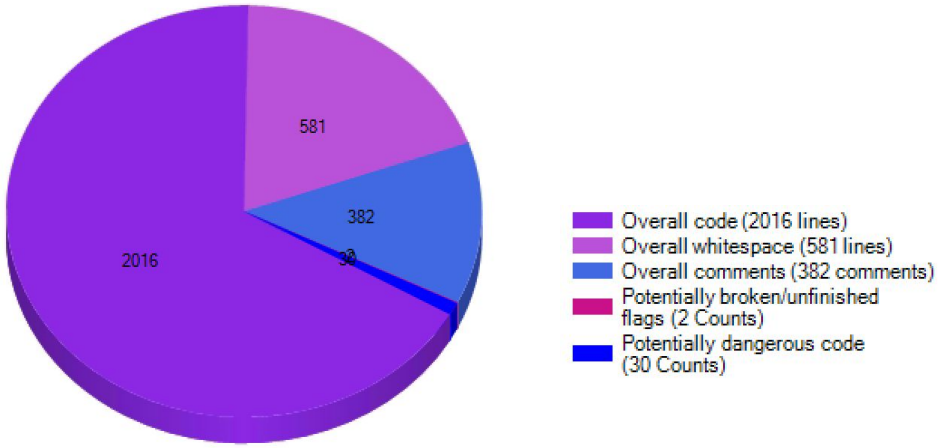

Figure 2.2: VisualCodeGrepper Visualization

As a whole, the software seems to be good in terms of static analysis. The chart, seen in Figure 2.2, visualizes the safety of all lines of code. It shows that only 30 out of 2016 lines are potentially dangerous. An additional two potentially broken/unfinished flags have been detected but upon further analysis, were seen to be a false alarm.

3. Code Documentation

3.1 Class Diagram

The class diagram shown in Figure 3.1 displays our view class. Each view has its own class that inherits the header and footer from the AbstractView class.

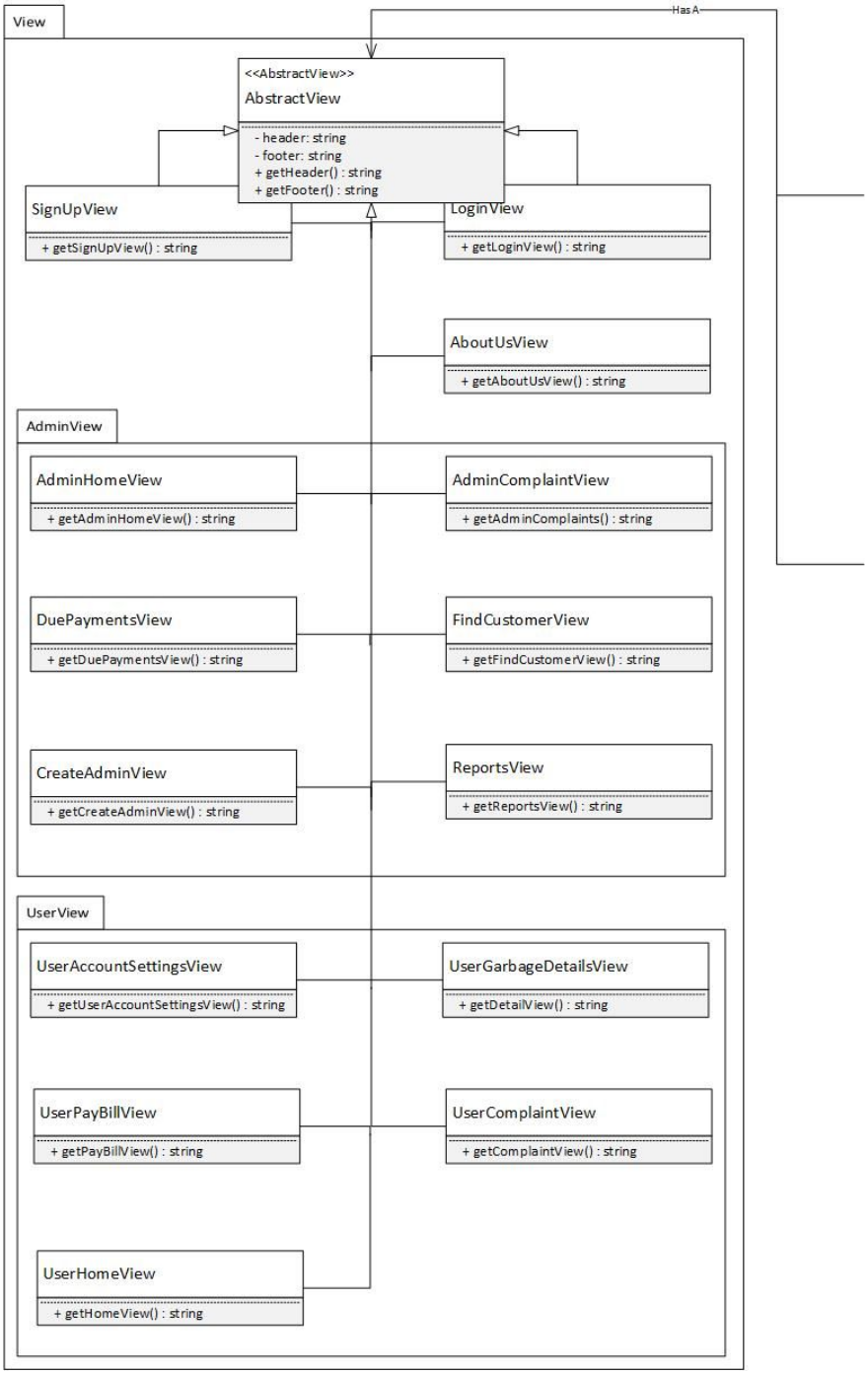

Figure 3.1: View

Figure 3.2 shows the controller classes. The ClientController and the AdminController will communicate with the DatabaseController to create and retrieve data in the database.

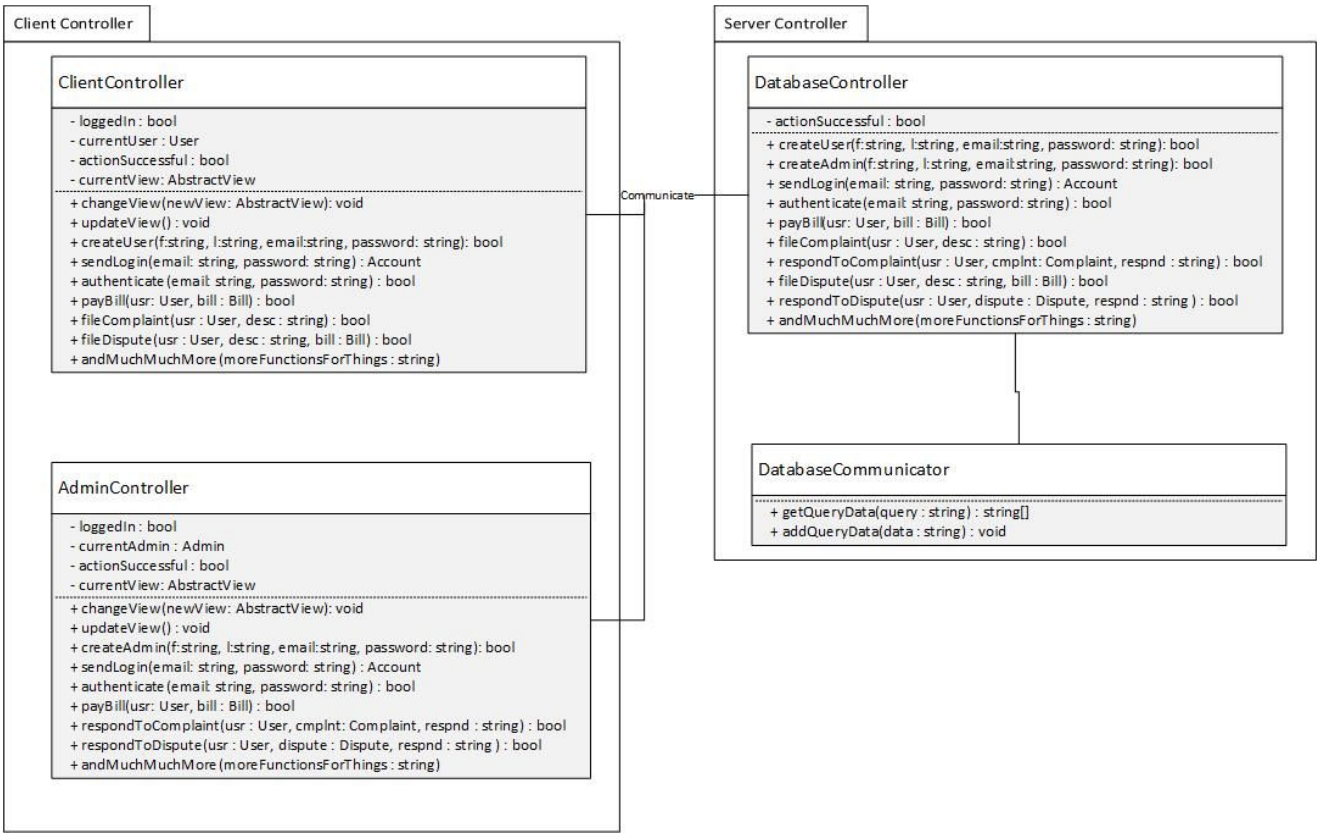

Figure 3.2: Controllers

Figure 3.3 displays the model package. It contains all of the data model classes needed in the system. Each class consists of the private needed variables and public methods to create and modify the values.

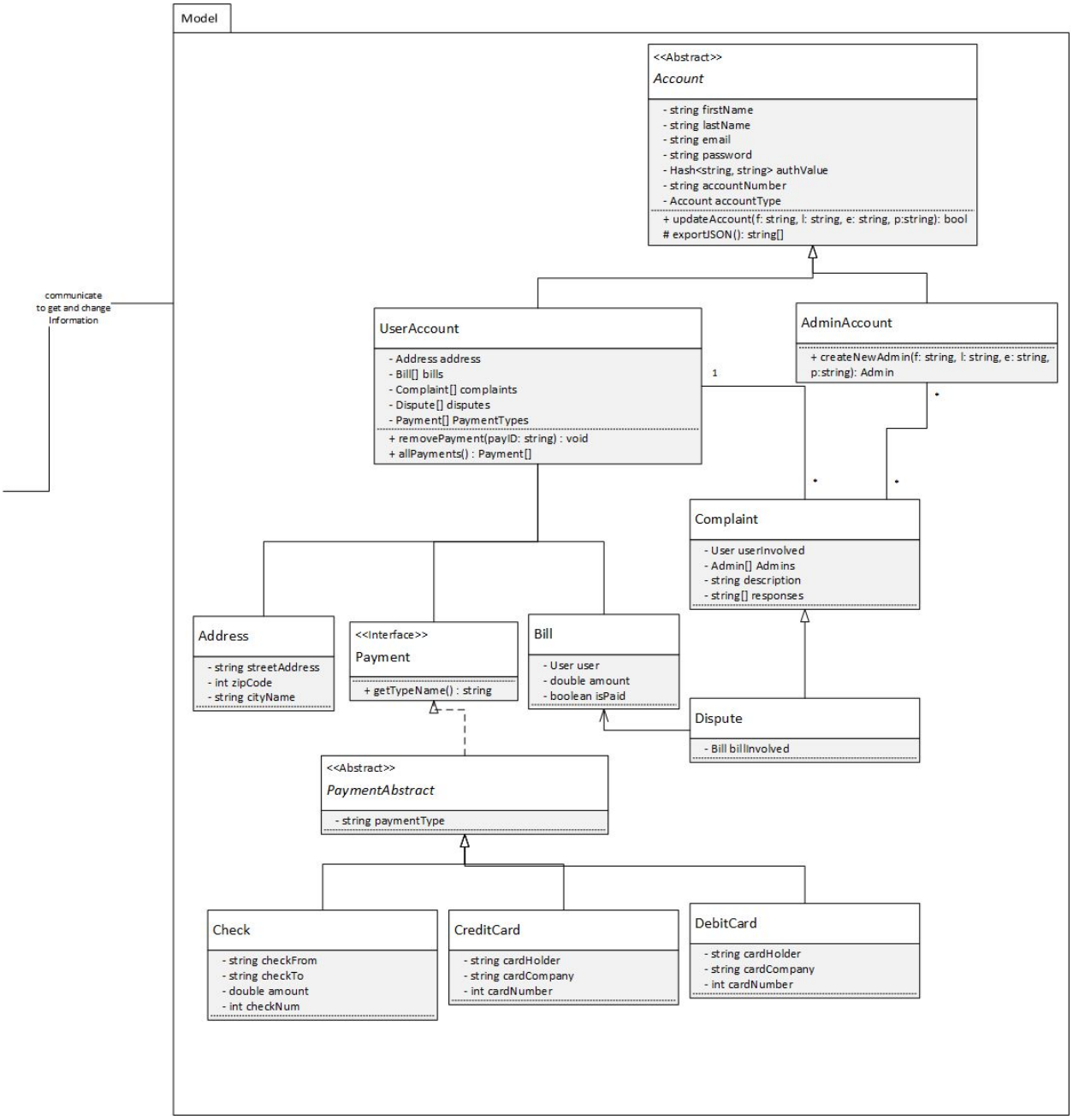

Figure 3.3: Model (server)

3.2 Class Description

3.2.1 View Subclasses

The view subclasses do not have any variables because they inherit all the variables necessary from the abstract View superclass a view that does not contain any data.
For each of the view classes, they will have the following values:

Protected Data

Class does not contain any protected data (Except for the abstract View. The abstract View class has a header and footer variable.)

Authentication Procedures

Upon construction, every view instantiates a ClientController, who calls its authenticateToken function upon construction, which automatically attempts to authenticate the client using a cookie stored token to determine what content should be displayed.

Functions that Change Values of Data

The only variables are the header and footer and they do not change.

Functions that Reveal Data

The functions getHeader and getFooter will reveal the header and footer for the view.

Minimum Guarantee of Security

The view can only make calls to functions on the ClientController to which they are authorized.

3.2.2 Controllers

3.2.2.1 ClientController

Protected Data

DatabaseController databaseController
Bool authenticated
User activeAccount
String permissions;

Authentication Procedures

Upon construction, the internal authenticateToken function is called, which attempts to authenticate the client by its authentication token stored in a cookie. Functionality is limited by permissions.

Functions that Change Values of Data

authenticateToken authenticates the user and sets the local protected activeAccount member to the account returned from the databaseController's authenticateToken function.

Functions that Reveal Data

authenticateToken: the currentUser will be returned, revealing the User object to the class

loadPickupTimes: the pickup times will be loaded from a json file and return by the databaseController

getActiveAccount: returns the protected activeAccount local variable

Minimum Guarantee of Security

No data will be revealed or changed unless authenticated.

3.2.2.2 AdminController

Protected Data

All data is inherited from the ClientController superclass

Authentication Procedures

All authentication procedures are inherited from the superclass.

Functions that Change Values of Data

createAndRegisterAdminAccount: triggers a data change in DatabaseController

createBillForUser: triggers a data change in DatabaseController

Functions that Reveal Data

Functions inherited by ClientController

findCustomer: reveals the customer account object associated with first name and last name or email

Minimum Guarantee of Security

No data will be revealed or changed unless authenticated. No functionality is available for non-administrators.

3.2.2.3 DatabaseController

Protected Data

A series of private static functions containing constant file paths is stored in this class.

Array failedLogins

Bool authenticated;

Account activeAccount

String permissions

Authentication Procedures

Low level authentication functions called by the ClientControllers are available publically. These also authenticate the DatabaseController to only allow functionality for the relevant permission level, as the DatabaseController is an instance of the ClientController.

Functions that Change Values of Data

authenticateToken: sets the local permission variable based on the client's authentication

markInvalidLogin: marks a client's invalid login onto the json file for request limiting

setUserPermission: sets the currently authenticated user's permission to user in the permissions json file

setAdminPermission: sets the currently authenticated admin's permission to admin in the permissions json file

deleteAccount: delete account associated with account number if invoked by an admin

deleteActiveAccount: deletes the actively authenticated account

overwriteFile: overwrites a variable file with variable data (private)

Functions that Reveal Data

authenticateToken: returns the user account associated

readFile: reads a private file (private)

accountExists: returns if the account number corresponds to an active account

Minimum Guarantee of Security

No data will be revealed or changed unless proper authentication provides relevant permissions..

3.2.2.4 DatabaseCommunicator

This class contains a lot of queries to retrieve the needed data for each view.

Protected Data

This class does not contain any protected data.

Authentication Procedures

There are no authentication procedures for this class.

Functions that Change Values of Data

The queries will not change the value of any of the data.

Functions that Reveal Data

The queries will return the searched for data from the database.

Minimum Guarantee of Security

The class does not own data, and therefore does not require a minimum guarantee of security to access it.

3.2.3 Models

3.2.3.1 Account

Protected Data

String firstName
String lastName
String email
Hash<string, string> authValue
String accountNumber
Account accountType

Authentication Procedures

The authValue will be used to authenticate the user or admin. Once authenticated, the protected data can be revealed.

Functions that Change Values of Data

updateAccount can update the values of firstName, lastName, email and password. There are also setter methods for accountNumber and accountType. The wipe function will reset all the values in the class.

Functions that Reveal Data

There are get methods to get the values for accountNumber, firstName, lastName, and email. exportJSON will return a JSON revealing firstName, lastName, email, accountNumber and accountType.

Minimum Guarantee of Security

No data will be revealed externally or changed until authenticated.

3.2.3.2 User

Protected Data

Address address
Bill[] bills
Complaint[] complaints
Dispute[] disputes
Payment[] paymentTypes
It also inherits protected data from Account.

Authentication Procedures

The authValue will be used to authenticate the user. Once authenticated, the protected data can be revealed.

Functions that Change Values of Data

The constructor sets firstName, lastName, email, password and address. It also creates empty arrays for bills, complaints, disputes and paymentTypes. removePayment will update the paymentTypes by deleting a payment from the array. It also inherits the ability to updateAccount to update the user's firstName, lastName, email or password. The wipe function will reset all the values in the class.

Functions that Reveal Data

The inherited get methods from Account will still work to get the values of the inherited variables. exportJSON will return a JSON revealing firstName, lastName, email,

accountNumber, accountType, bills, address, complaints and disputes. AllPayments will return the list of all the paymentTypes.

Minimum Guarantee of Security

No data will be revealed externally or changed until authenticated.

3.2.3.3 Admin

Protected Data

Inherits protected data from Account.

Authentication Procedures

The authValue will be used to authenticate the admin. Once authenticated, the protected data can be revealed.

Functions that Change Values of Data

The constructor sets firstName, lastName, email and password. updateAccount will also affect these values. It also inherits the ability to updateAccount to update the admin's firstName, lastName, email or password. The wipe function will reset all the values in the class.

Functions that Reveal Data

The inherited get methods from Account will still work to get the values of the inherited variables.

exportJSON will return a JSON revealing firstName, lastName, email, accountNumber and accountType.

Minimum Guarantee of Security

No data will be revealed externally or changed until authenticated.

3.2.3.4 Complaint

Protected Data

User userInvolved
Admin[] admins
String description
String[] responses

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal or modify the data.

Functions that Change Values of Data

The constructor sets the userInvolved and the description. A response can be added with an admin with an addResponse method.

Functions that Reveal Data

There are getter methods for all the data so that both the user and admin clients can view the complaint descriptions and responses with their corresponding user and admins.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated. The complaint can only be created by a user.

3.2.3.5 Address

Protected Data

String streetAddress
Int zipCode
String cityName

Authentication Procedures

The user must be authenticated and have a loggedIn value in its controller of true to reveal or modify the data.

Functions that Change Values of Data

Values of variables can be set with setter methods or in the constructor. These methods can only be accessed by the user class.

Functions that Reveal Data

There are getter methods for streetAddress, zipCode and cityName.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated. The address data can only be modified by a user.

3.2.3.6 Bill

Protected Data

User user
Double amount
Boolean isPaid

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal or modify the data.

Functions that Change Values of Data

The constructor will set the user and amount. isPaid has a setter method to update its value.

Functions that Reveal Data

Each variable has a getter method to retrieve that value.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated.

3.2.3.7 Dispute

Protected Data

Bill billInvolved

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal or modify the data.

Functions that Change Values of Data

The billInvolved is set in the constructor for the dispute. It also inherits all of the variables of a complaint, meaning the constructor sets the userInvolved and the description and a response can be added with an admin with an addResponse method.

Functions that Reveal Data

The billInvolved can not change after the bill is created. There are getter methods for all of the inherited data so that both the user and admin clients can view the dispute descriptions and responses with their corresponding user and admins.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated. The dispute can only be created by a user.

3.2.3.8 PaymentAbstract

Protected Data

String paymentType

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal the data.

Functions that Change Values of Data

Constructor will set the paymentType.

Functions that Reveal Data

getPaymentType will return paymentType.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated.

3.2.3.9 Check

Protected Data

String checkFrom
String checkTo
Double amount
Int checkNum

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal the data.

Functions that Change Values of Data

The constructor sets all of the protected variables and can not be changed afterwards.

Functions that Reveal Data

There are getter methods for all of the protected variables.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated. Only a user account can modify the data, however both admin and users can view the data.

3.2.3.10 CreditCard

Protected Data

String cardHolder
String cardCompany
Int cardNumber

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal the data.

Functions that Change Values of Data

The constructor sets all of the protected variables and can not be changed afterwards.

Functions that Reveal Data

There are getter methods for all of the protected variables.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated. Only a user account can modify the data, however both admin and users can view the data.

3.2.3.11 DebitCard

Protected Data

String cardHolder
String cardCompany
Int cardNumber

Authentication Procedures

The user or admin must be authenticated and have a loggedIn value in its controller of true to reveal the data.

Functions that Change Values of Data

The constructor sets all of the protected variables and can not be changed afterwards.

Functions that Reveal Data

There are getter methods for all of the protected variables.

Minimum Guarantee of Security

No data will be revealed or changed until authenticated. Only a user account can modify the data, however both admin and users can view the data.

3.3 Method Descriptions

3.3.1 ClientController

3.3.1.1 changeView

Minimum Expectation of Input

The new AbstractView that they would like to change to.

Output Information

currentView will be updated with the inputted view.

Error Output

An error will appear stating “Unable to change view.”

External Resources

The method to get the view of the newView will be called.

Fail State Result

The view will remain on the same page and not change to the newView.

Required Variable Names

newView

Required Variable Data Types

AbstractView newView

Optional Variable Names

None

Optional Variable Data Types

None

Access Type of Variables

newView - copy

3.3.1.2 updateView

Minimum Expectation of Input

None

Output Information

The information in the view will be updated with the latest information.

Error Output

An error will appear stating “Unable to update view.”

External Resources

Fail State Result

The view will remain the same without updating the data in the view.

Required Variable Names

None

Required Variable Data Types

None

Optional Variable Names

None

Optional Variable Data Types

None

Access Type of Variables

None

3.3.1.3 createUser

Minimum Expectation of Input

String inputs of a firstname, lastname, email, and password are the minimum expected input.

Output Information

The function returns a boolean value of whether or not the new user was created.

Error Output

If a user already exists with the entered email, the system will display an error stating “The email entered already has an account.” If the user enters a password that does not meet the criteria, the system displays an error stating “Please enter a valid password”.

External Resources

The constructor method of the user class will be called to create the new user.

Fail State Result

The system displays “The user account was not created” and the user is not created.

Required Variable Names

F, l, email, password

Required Variable Data Types

All variables are strings

Optional Variable Names

None

Optional Variable Data Types

None

Access Type of Variables

F - copy
L - copy
Email - copy
Password - copy

3.3.1.4 sendLogin

Minimum Expectation of Input

The email and password.

Output Information

The user that has the email and password matching the inputs. LoggedIn value is set to true.

Error Output

If no user has a matching email and password combination, the system displays an stating “Incorrect username or password.”.

External Resources

Must use authenticate method to check if the user exists

Fail State Result

User is not logged in and remains at the LoginView. LoggedIn value remains false.

Required Variable Names

Email and password

Required Variable Data Types

String email
String password

Optional Variable Names

None

Optional Variable Data Types

None

Access Type of Variables

Email - copy

Password - copy

3.3.1.5 Authenticate

Minimum Expectation of Input

An email and password.

Output Information

Returns a true if a user is found with a matching email and password combination.

Error Output

If no user has a matching email and password combination, the system displays an stating “Incorrect username or password.”.

External Resources

Will check to see if a user object in the database that matches with the email and password combination.

Fail State Result

Returns false

Required Variable Names

Email and password

Required Variable Data Types

String email

String password

Optional Variable Names

None

Optional Variable Data Types

None

Access Type of Variables

Email - copy

Password - copy

3.3.1.6 payBill

Minimum Expectation of Input

The user and the bill they would like to see.

Output Information

Updates the actionSuccessful value to true to signify that the bill has been paid. Updates bill as paid in the database.

Error Output

The system displays an error stating “The bill was unable to be paid at this time. Please try again later.”

External Resources

The isPaid value is set to true for the bill in the database.

Fail State Result

The function will return false and set the actionSuccessful value to false. The isPaid value for the bill will remain unchanged.

Required Variable Names
User and bill
Required Variable Data Types
User user
Bill bill
Optional Variable Names
None
Optional Variable Data Types
None
Access Type of Variables
User - reference
Bill - reference

3.3.1.7 fileComplaint

Minimum Expectation of Input
The user and description.
Output Information
Creates a new complaint stored in the database. Updates the actionSuccessful value to true to signify that the complaint was filed successfully.
Error Output
The system displays an error stating “The complaint was unable to be filed at this time. Please try again later.”
External Resources
The constructor for the complaint class will be called to create a new complaint and store it to the database.
Fail State Result
The function will return false and set the actionSuccessful value to false. The complaint will not be added to the database.
Required Variable Names
User and desc
Required Variable Data Types
User user
String desc
Optional Variable Names
None
Optional Variable Data Types
None
Access Type of Variables
User - reference
Desc - copy

3.3.1.8 fileDispute

Minimum Expectation of Input
The user, description and the bill involved.
Output Information
Creates a new dispute stored in the database. Updates the actionSuccessful value to true to signify that the dispute was filed successfully.
Error Output

The system displays an error stating “The dispute was unable to be filed at this time. Please try again later.”

External Resources

The constructor for the dispute class will be called to create a new dispute and store it to the database.

Fail State Result

The function will return false and set the actionSuccessful value to false. The dispute will not be added to the database.

Required Variable Names

User, desc, and bill

Required Variable Data Types

User user
String desc
Bill bill

Optional Variable Names

None

Optional Variable Data Types

None

Access Type of Variables

User - reference
Desc - copy
Bill - reference

4. Testing Results

4.1 Testing Approaches

The three types of testing we used were unit, integration and penetration testing. Unit testing exercises individual functions, methods, classes, or stubs. This is testing each individual component to ensure that it does what it is supposed to and is done so securely. Integration testing focuses on a collection of subsystems, which may contain many executable components. So this is testing how different pieces of the code interact with one another and if they do so in a secure way. Penetration testing allows project managers to assess how an attacker is likely to try to subvert a system. This means testing the security of a computer system and/or software application by attempting to compromise its security.

4.1.1 Unit Testing

All model classes’ functionalities were tested individually before focus on the DatabaseController. Each model class provided the proper functionality free of errors. The DatabaseController cannot be properly unit tested as it's most basic functionality involves interacting with multiple objects. The validator class, which deals with validating all external input, should be further tested with more elabore unit testing to assure no malicious scripts can sneak past the filters and exploit the server.

4.1.2 Integration Testing

A custom class was designed for simple integration testing of the main controller's core functionality. Although no frameworks were utilized to enhance the testing results, the class has simple outputs comparing the functions expected output versus actual output. Through this, an issue regarding the storage and reading of associative PHP arrays came to light, as when json objects were loaded from the disk, they no longer contained the class functions or even belonged to the class. Through this testing, unrecognizable buggy behavior became clear and a fix was put in place to prevent calling a function from

a standard class object. More integration testing is necessary to be able to assure the integrity of the system as a whole.

4.1.3 Penetration Testing

In order to prepare the web application for real world attackers, one must test the system as one. Therefore, a series of penetration tests were performed on the ErieGarbage system to assess its strengths and weaknesses.

4.1.3.1 Brute-force Attacks

Brute-force attacks were the largest weakness in this application. A user was able to make an infinite amount of requests with no lockout limit. However, a framework is now in place to handle excessive subsequent invalid requests. This removes the possibility of an attacker trying as many email and password combinations as desired, with hopes of finding a match.

An attacker, however, can attempt to enumerate all directories and/or files in the server, with hopes of finding hidden information in the server. The mitigation in place only applies to user requests to PHP code, not to the Apache web server hosting the site.

On the default configuration of apache, if a client visited a directory URL, apache would list the contents of the entire directory, and allow looking at any of the files. A client would be presented with the interface on Figure 4.1 to and be able to traverse every directory within the server root. Via this interface, an attacker can obtain internal knowledge of system organization, the version of apache that is currently running, and even access any of the sensitive credentials file much like credentials.json on Figure 4.2, containing all account's hashes and ids in an associated array.

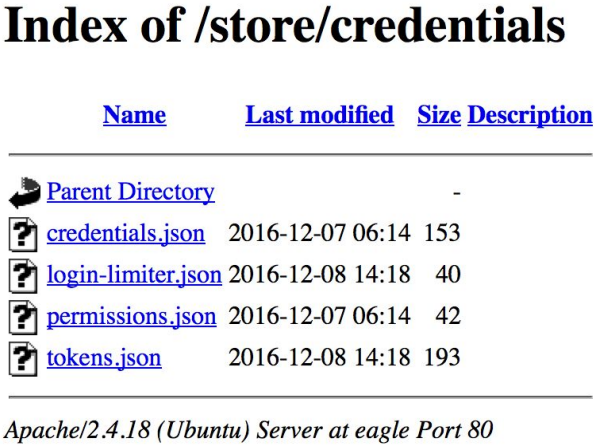

Figure 4.1: Apache Directory Indexing

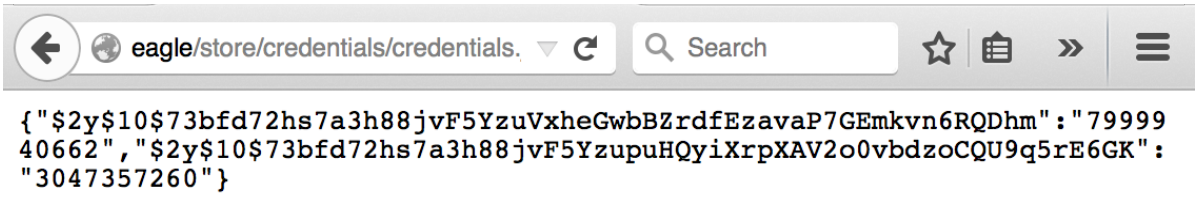

Figure 4.2: Reading Sensitive Information from Browser

Thankfully, the directory indexing issue was very easily solved by correctly configuring the Apache configuration files to disable the option.

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>
```

Figure 4.3: Apache Configuration Changes

Figure 4.3 shows the highlighted option to be deleted to disable indexing for the root of the web server (/var/www). This modification redirects users to a forbidden page whenever they attempt to visit a directory.

Now, in order to restrict access to the store files, the file structure of the project was modified to keep the store folder outside of the server hosted files, disallowing direct client access to critical files that should only be accessed by the server. Only one line of code in the database controller was updated.

4.1.3.2 Traffic Analysis

In order to assure the message exchange between server and client is as expected, BurpSuite by PortSwigger was utilized to analyze the requests sent back and forth. For this tool, HTTPS has to be disabled, rendering all exchanged data as plain text. Through this, the plain-text password and email for a login request and the authentication cookie are clearly visible, as shown in Figure 4.1.

```
POST /login.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
Accept: text/html,application/xhtml+xml,applicati
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://localhost/login.php
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 40

email=hi%40hotmail.com&password=password
```

Figure 4.4: Unencrypted Client Login HTTP Request

The server then retrieves than information, and upon successful authentication, sends back an authentication token object to be stored in the user's browser as a cookie. We can assert there is no sensitive information other than a randomized token and an account number in the token by taking a look at the server's response in Figure 4.5, with the authentication token's parsed values in Figure 4.6.

```
GET /home.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:39.0) Gecko/20100101
Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://localhost/login.php?fail=true
Cookie:
eg-auth=%7B%22id%22%3A%227999940662%22%2C%22token%22%3A%229567ea5871a445a041e47f1081
cb61be%22%2C%22expiry%22%3A1481228792%7D
Connection: close
```

Figure 4.5: Server Successful Login HTTP Response

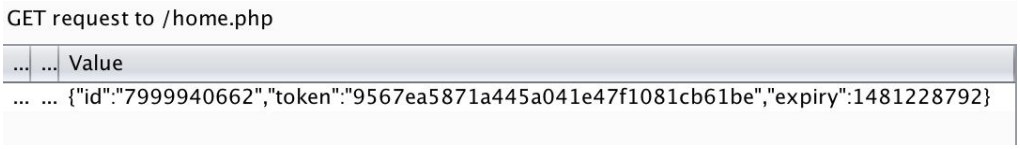

Figure 4.6: Authentication Token Values

If an attacker managed to intercept the value of this cookie, he could successfully hijack the client's session. However, with the final configuration of the system and the HTTPS protocol in place using SSL, public key encryption is utilized, rendering attackers' sniffing attacks void.

An attacker could still compromise the client's system, and steal the token, hijacking the session, but current expiry date of (15) minutes for each token minimizes the risk. Because of this, clients are redirected to the login page every fifteen minutes.

4.1.3.3 Fuzz Testing

The most probable place for code injection to take place is the login screen to which anyone has access. However, fuzz testing resulted in no erroneous behavior. The user is simply let know that authentication failed, and can try again.

4.2 Found Bugs

Many bugs were encountered throughout the coding and testing process. The major ones are documented in Figure 4.7 below, along with an assessment of risk and the mitigation plan or action taken.

#	Bug	Risks Assessment	Mitigation
1	Insecure usage of MD5 hashing algorithm in controllers for credentials	A leakage of hashed credentials can be easily cracked using brute-force, especially weak passwords..	Use PHP's built-in password_hash function using the secure bcrypt algorithm (as of PHP version 5.5).
2	Broken invalid authentication limit for request origin IP address	A user can bruteforce any inputs in the server, including credentials.	The authenticator limiter's implementation has been fixed.
3	After a client is timed out for sending (5) incorrect login requests, login failed is shown on following requests, rather than timeout.php.	Client can actively fail to log in with no knowledge of being locked out, further getting locked out for even longer, or even believe his credentials are wrong.	The login method verifying request limits was modified to redirect to timeout.php whenever the limit is exceeded.
4	Directory indexing is enabled, clients can and list and access all files	Clients can enumerate all files, better understand the system, and directly attack weak point or files.	Modify apache's configuration to disable indexing.

5	Credentials store accessible via known url	Clients can read credentials, permissions, and account files, resulting in potential information and/or password hash leaks	Re-locate the store directory to be outside of the server root.
6			

Figure 4.7: Encountered Bugs

4.3 Logged Error Messages

The ErieGarbage system logs system behavior via the Logger class in the /store/management/log.txt file. Error messages within are described in more detail in Figure 4.8 below.

#	Error Message	Cause	Potential Solutions
1	Failed to load account	Account number from authenticated file does not correspond to an existing account JSON file, or the entry in the credentials file corresponds to a non-existing account number.	1. Create the account entry in **/store/accounts/{accountNumber}.json** 2. Remove the user credentials from **/store/accounts/credentials.json**
2	Failed to read credentials	Credentials file could be missing or read/write privileges for the system could be wrong	1. Validate the credentials.json file in **/store/credentials/credentials.json** 2. Fix execution system privileges
3	Failed to read failed logins	Failed logins file (for request timeouts) can be missing, corrupt, or system privileges are misset.	1. Validate the login-limiter.json file in **/store/credentials/login-limiter.json** 2. Fix execution system privileges
4	Failed to read tokens file	Tokens file might be corrupt/missing or read/write privileges for system can be off	1. Validate the tokens.json file in **/store/credentials/tokens.json** 2. Fix execution system privileges
5	Invalid input provided	The client provided illegal input, no risk, just informational logging	No solution, already solved.
6	Request limit for account exceeded	The client attempting to authenticate exceeded their invalid request limit.	Client should wait (15) minutes for the timeout to expire.
7	Failed to load permissions file	The server failed to load the permissions file, permissions for users cannot be verified	1. Validate the tokens.json file in **/store/credentials/permissions.json** 2. Fix execution system privileges
8	Invalid account type loaded	Tampering with account file store took place, system may be compromised.	Validate the associated account .json object and its account type property.
9	Account file attempted to delete does not exist	Tampering with account file may have taken place, system may be compromised	Validate the credentials file, accounts file, and permissions file for ghost accounts. (accounts not existing in all files)
10	Permission denied	Client attempted to access unauthorized functionality	No solution, already solved.
11	Token mismatch	The auth token supplied in the client browser's cookie does not match the one in store. Can be caused by a client tampering with the cookies.	If not caused by an attacker, or a user tampering with cookies, must be an implementation issue.

Figure 4.8: Error Messages

5. Team Members Log Sheets

5.1 Mason Toy

Date	task	duration
11/29/16	Creating Base Templates for home and headers	2.5hrs
12/2/16	Class Diagram redesign part 1	1.5 hrs
12/3/16	Static Analysis & coding	5.5 hrs
12/4/16	New Static Analysis test	1.5 hrs
12/8/16	Fixing Class Diagram Testing Implementation	2.5 hrs
	Total :	13.5 hrs

5.2 Allison Steinmetz

date	task	duration
12/3/16	Set up document. Updated Class Diagram and section 3.2.	6 hrs
12/4/16	Added sections 1, 2 and 4. Completed sections 1 and 3. Updated Class Diagram.	9 hrs
12/5/16	Minor modifications to sections 2 and 4 descriptions.	.5 hrs
12/7/16	Updated section 3.2	1.5 hr
12/7/16	Added section 3.3	1.5 hr
12/8/16	Completed section 3.3	2.5 hr
12/9/16	Updated class diagrams and added descriptions	1 hr
	Total :	22 hrs

5.3 Daniel Lopez

date	task	duration
12/x/16	Implementation	11 hrs
12/4/16	Work on (4.1) Testing section, (2) Static Analysis	2 hr
12/6/16	4.2: Bugs	0.5 hrs
12/6/16	Implementation + 4.3: Errors	5 hrs
12/7/16	Implementation	6 hrs
12/8/16	Implementation + Traffic Analysis (4.1.3 Pentesting)	3 hrs
12/8/16	Brute-force Attacks (4.1.3 Pentesting)	1 hr
12/8/16	Environment setup, HTTPS, 3.3.2	2 hrs
	Total :	30.5 hrs