DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Secure Coding

Phase 4

Team: 8

Members: Korbinian Würl

Mai Ton Nu Cam

Vivek Sethia

Swathi Shyam Sunder

Executive summary

Online Banking

Online Banking contains a large number of vulnerabilities, syntax errors and business logic flaws. One of the most severe of them is a widespread vulnerability for SQL-injection. Additionally, the application exhibits reflected XSS vulnerability. This way attackers can send poisoned links to unsuspecting persons which allows them to take control of their account. Checking the php error log it reveals a number of syntax errors. The consequence is unsuspected application behaviour. The file parser written in C also contains many SQL-injection and buffer overflow vulnerabilities. This yields the possibility of total control over the whole server.
These are only a few of all contained vulnerabilities. In addition to this, the application contains a number of deviations from the application requirements as well as usability flaws and functional issues.

SecureBank

While analyzing the SecureBank application we could only detect two minor vulnerabilities. One is, that the User Identities are not verified automatically. To fix this, email and address verification mechanisms have to be implemented. The second one is, that after 100 transactions no new tan codes are sent to the user and this way he cannot use the application any more. Both of these vulnerabilities are minor from a security aspect and are also contained within the Online Banking application.

Comparison

A comparison between SecureBank and Online Banking is difficult. SecureBank fulfills almost all functional and security requirements. Online Banking still has major deficiencies in almost all of the named disciplines.

Contents

1 Time tracking

1.1 Korbinian Würl

Task	Time in h
General Discovery	2
Test Role Definitions	0.5
Writing down Test Role Definitions	0.5
Test User Registration Process	0.25
Writing down Test User Registration Process	0.25
Testing for bypassing authentication schema	0.5
Writing down Testing for bypassing authentication schema	0.5
Testing for Privilege Escalation	0.5
Writing down Testing for Privilege Escalation	0.25
Testing for logout functionality	0.5
Writing down Testing for logout functionality	0.5
Test Session Timeout	0.5
Writing down Test Session Timeout	0.25
Testing for Command Injection	0.5
Writing down Testing for Command Injection	0.25
Analysis of Error Codes	0.5
Writing down Analysis of Error Codes	0.5
Analysis of Stack Traces	0.5
Writing down Analysis of Stack Traces	0.25
Test for Process Timing	0.5
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	1.5
Writing abstract	1
General report Time	2
Disassembly of C-Parser testting tools	1.5
Disassembly of C-Parser understand controll flow	2
Disassembly of C-Parser naming variables	2
Disassembly of C-Parser writing C-Parser	2
Disassembly of C-Parser writing C-Parser	2
Disassembly of C-Parser testing C-Parser	2
Presentation Time	1
Total	**28.75**

1.2 Mai Ton Nu Cam

Task	Time in h
General Discovery	2
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	0.5
Testing for Weak lock out mechanism	1
Writing down testing weak lock out mechanisms	0.5
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	0.5
Testing for Exposed Session Variables	1
Writing down testing exposed session variables	0.5
Testing for Cross Site Request Forgery	0.5
Writing down testing CSRF	0.5
Testing for HTTP Parameter pollution	0.5
Writing down testing HTTP parameter pollution	0.5
Testing for SQL Injection	2
Writing down testing SQL Injection	1
Testing for Code Injection	1
Writing down testing code injection	0.5
Writing down testing incubated vulnerabilities	0.25
Testing for HTTP Splitting/Smuggling	0.5
Writing down testing HTTP Splitting/Smuggling	0.5
Test Ability to Forge Requests	0.5
Writing down testing forging requests	0.5
Test Integrity Checks	0.5
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
Disassembly of C-Parser: testing tools	1.5
Disassembly of C-Parser: understanding control flow	2
Disassembly of C-Parser: writing C-Parser	2
Disassembly of C-Parser: testing C-Parser	2
Presentation Time	0.5
Total	**30.5**

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 & Reporting for XPath Injection	0.5
Testing & Reporting 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
Finding & Reporting vulnerabilities in SCS	1.00
General Report time & corrections	2
Reporting Analysis of Tools	1.50
Calculation of CVSS Score and adding score bars	1.00
Presentation for Demo	0.5
Total	**28.75**

1.4 Swathi Shyam Sunder

Task	Time in h
Test Account Provisioning Process	0.75
Reporting Test Account Provisioning Process	0.75
Testing for Account Enumeration and Guessable User Account	0.5
Reporting Account Enumeration and Guessable User Account	1.00
Testing for Browser cache weakness	0.5
Reporting Testing for Browser cache weakness	0.5
Testing for Weak password policy	1.00
Reporting Testing for Weak password policy	1.00
Testing for bypassing authorization schema	0.75
Reporting Testing for bypassing authorization schema	0.5
Testing for Insecure Direct Object References	0.5
Reporting Testing for Insecure Direct Object References	0.75
Testing for Bypassing Session Management Schema	0.75
Reporting Testing for Bypassing Session Management Schema	0.75
Testing for Session puzzling	0.5
Reporting Testing for Session puzzling	0.5
Testing for Reflected Cross Site Scripting	1.00
Reporting Testing for Reflected Cross Site Scripting	1.00
Testing & Reporting for XML Injection	0.5
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	0.75
Reporting Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	0.5
Testing & Reporting for Padding Oracle	0.5
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	0.50
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 & Reporting for CSS Injection	0.5
Testing for Client Side Resource Manipulation	0.5
Reporting Testing for Client Side Resource Manipulation	0.5
Testing & Reporting for Features & Functionality	1.50
Reporting Analysis of Tools	1.00
Decompilation & Code generation of Java SCS	1.00
Reporting Decompilation & Code generation of Java SCS	1.00
Studying & Reporting TAN generation algorithm in SCS	1.00
Writing & Reporting AllTanGenerationTests script for SCS	2.00
Calculation of CVSS score and adding score bars	0.5
Overall Report Time & Corrections	1.00
Creation of demo videos for presentation	0.5
Total	**32.75**

2 Overview of most important observations

2.1 Vulnerabilities of Online Banking

2.1.1 Sensitive Data Exposure

There is no implementation of HTTPS. So sensitive data is communicated without any encryption.

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

2.1.2 Session Hijacking

The session cookie is not set to Secure or HttpOnly, thus allowing manipulation from client-side.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-SESS-001 and OTG-SESS-003

2.1.3 Weak Password Policy

Passwords are stored in database using md5, thus easily traceable. No password policy during password reset.

- **Likelihood:** High
- **Impact:** High
- **Risk:** High
- **Reference:** OTG-AUTHN-007

2.1.4 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

2.1.5 SQL injection

For most of the queries, neither are prepared statements used nor are user inputs sanitized/validated.

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

2.1.6 Buffer overflow

The biggest issue with buffer overflow is the use of functions like `strcpy`, `strcat` and `sprintf` without checking for the lengths.

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

2.2 Vulnerabilities of SecureBank

2.2.1 Buffer overflow

The biggest issue with buffer overflow is the use of the function `strcpy` without checking for the lengths.

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

2.2.2 Test Number of Times a Function Can be Used Limits

after 100 transactions no new tan codes are sent to the user and this way he can not use the application any more.

- **Likelihood:** High
- **Impact:** Low
- **Risk:** Low
- **Reference:** OTG-BUSLOGIC-006

3 Tools

3.1 Distribution of Tools

3.1.1 Korbinian Würl

Tool

RATS (`https://code.google.com/p/rough-auditing-tool-for-security`)

IDA Pro Free (`https://www.hex-rays.com/products/ida/support/download_freeware.shtml`)

Google Chrome Developer Tools

cURL

3.1.2 Mai Ton Nu Cam

Tool

Valgrind (`http://valgrind.org/`)

IDA Pro Free (`https://www.hex-rays.com/products/ida/support/download_freeware.shtml`)

objdump

3.1.3 Vivek Sethia

Tool

RIPS (`http://rips-scanner.sourceforge.net`)

RATS (`https://code.google.com/p/rough-auditing-tool-for-security`)

FindBugs (`http://findbugs.sourceforge.net`)

EditThisCookie (Chrome Extension)

3.1.4 Swathi Shyam Sunder

Tool

Kiuwan (`https://www.kiuwan.com`)

JD-GUI - Java Decompiler (`http://jd.benow.ca`)

AllTanGenerationTests.java (Custom JUnit Script)

Advanced REST Client (Chrome Extension)

3.2 Analysis

3.2.1 RIPS

RIPS was used for static analysis of PHP code as the first step in finding vulnerablities in the application.

Online Banking

Online Banking showed 193 issues. Refer 3.1. The tool reported the following issues:

- Command Execution - Refer 3.2.
- File Inclusion - Refer 3.3.
- File Manipulation - Refer 3.4.
- Reflection Injection - Refer 3.5.
- Session Fixation - Refer 3.6.
- XSS - Refer 3.7.
- SQL Injection - Refer 3.8.

All the above issues were found to be false positives which was also confirmed by black box testing and manual code inspection.

SecureBank

SecureBank showed 331 issues. This is reasonable owing to the number of files. Refer 3.9. The tool reported the following issues:

- Command Execution - Refer 3.10.
- Code Execution - Refer 3.11.
- File Manipulation - Refer 3.12.
- Reflection Injection - Refer 3.13.
- HTTP Response Splitting - Refer 3.14.
- XSS. Refer - 3.15.

All the above issues were found to be false positives which was also confirmed by black box testing and manual code inspection.

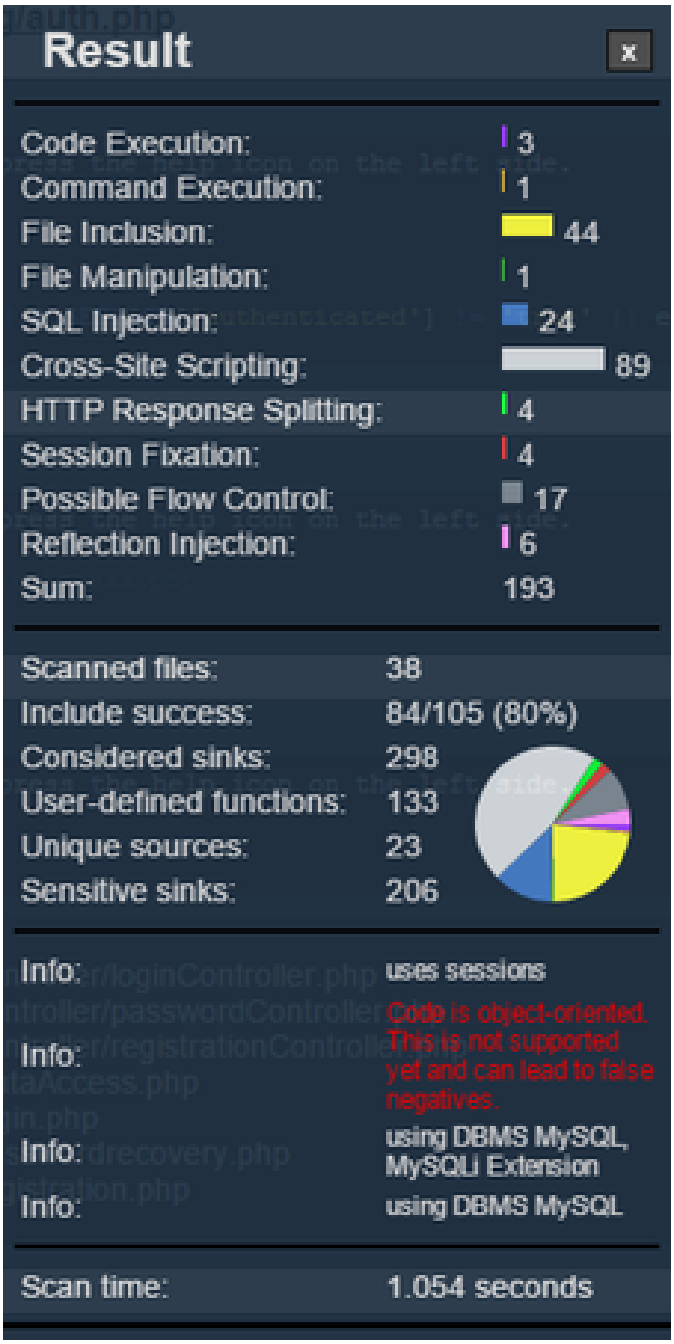


Figure 3.1: Overview of RIPS scan for Online Banking

```
Command Execution

Userinput reaches sensitive sink. For more information, press the help icon on the left side.

    67: shell_exec $out = shell_exec("./parser/exec " . $sessionuserid . " " . $target_file);  // clientFunctions.php
      • 38: function processfile($sessionuserid)
        43: $target_file = $target_dir . "form.txt";  // clientFunctions.php
            41: $target_dir = "/home/samurai/Documents/parse/";  // clientFunctions.php

        requires:
            62: if($uploadOk != 0)
            64: if(move_uploaded_file($_FILES['tranfile']['tmp_name'], $target_file))

Call triggers vulnerability in function processfile()

    91: $success = processfile ($sessionuserid);  // clientFunctions.php
      • 88: function processpost($sessionuserid)

        requires:
            90: if($_GET['action'] == 'file')

Vulnerability is also triggered in:
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/controller/clientController.php
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/controller/clientFunctions.php
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/index.php
```

Figure 3.2: RIPS: Command Execution vulnerability reported for Online Banking

```
File Inclusion

Userinput reaches sensitive sink. For more information, press the help icon on the left side.

    342: include_once include_once 'PEAR.php';  // auth.php

        requires:
            338: if(!session_id())
            340: if(!session_id())

Vulnerability is also triggered in:
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/employee.php
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/index.php

File Inclusion

Userinput reaches sensitive sink. For more information, press the help icon on the left side.

    468: include_once include_once 'Auth/Container/' . $driver . '.php';  // auth.php
      • 485: function s_factory($driver, $options = '')

Vulnerability is also triggered in:
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/employee.php
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/index.php
```

Figure 3.3: RIPS: File Inclusion vulnerability reported for Online Banking

```
File: C:\Users\seth\Desktop\Phase 4 source\InternetBanking/client.php

File Manipulation

Userinput reaches sensitive sink when function processfile() is called. (Blind exploitation)

    64: move_uploaded_file move_uploaded_file($_FILES['tranfile']['tmp_name'], $target_file))  // clientFunctions.php
        43: $target_file = $target_dir . "form.txt";  // clientFunctions.php
            41: $target_dir = "/home/samurai/Documents/parse/";  // clientFunctions.php

        requires:
            62: if($uploadOk != 0)
            38: function processfile($sessionuserid)

Call triggers vulnerability in function processfile()

    91: $success = processfile ($sessionuserid);  // clientFunctions.php
      • 88: function processpost($sessionuserid)

        requires:
            90: if($_GET['action'] == 'file')

Vulnerability is also triggered in:
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/controller/clientController.php
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/controller/clientFunctions.php
    C:\Users\seth\Desktop\Phase 4 source\InternetBanking/index.php
```

Figure 3.4: RIPS: File Manipulation vulnerability reported for Online Banking

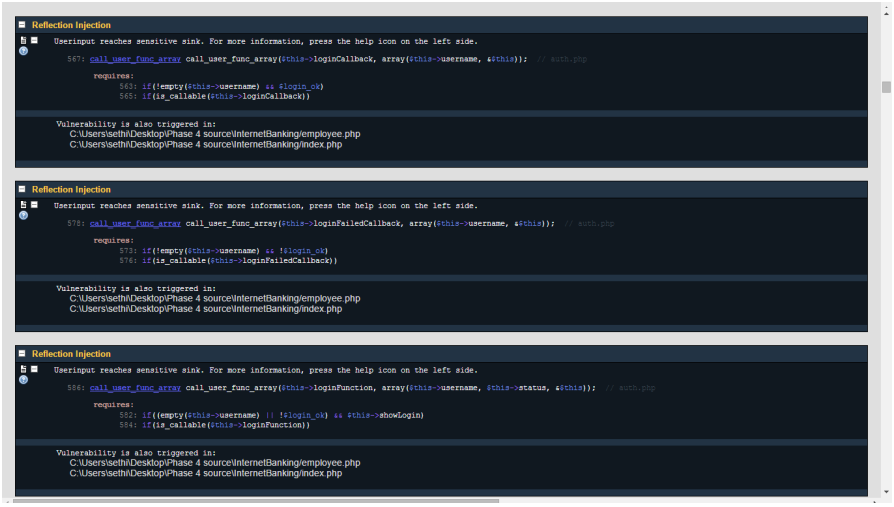


Figure 3.5: RIPS: Reflection Injection vulnerability reported for Online Banking

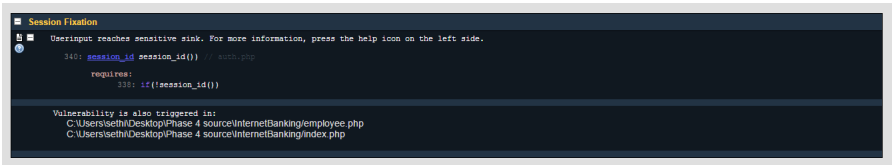


Figure 3.6: RIPS: Session Fixation vulnerability reported for Online Banking

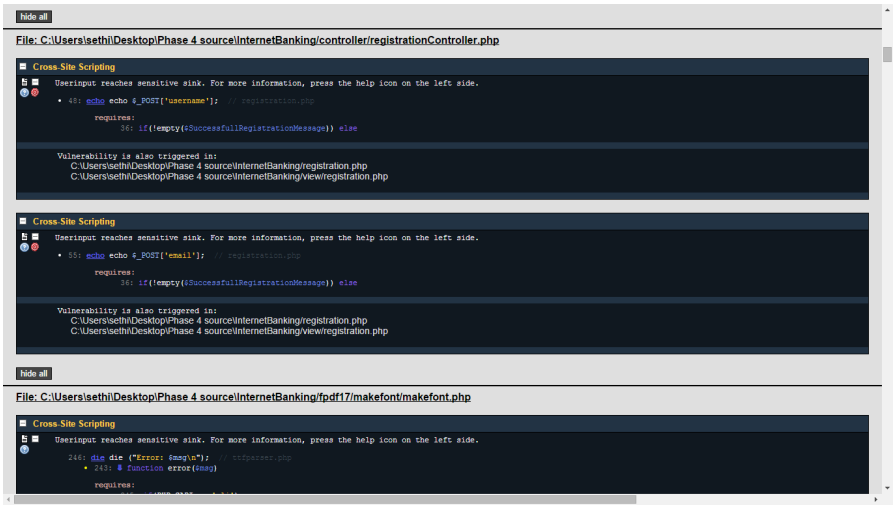


Figure 3.7: RIPS: Cross Site Scripting vulnerability reported for Online Banking

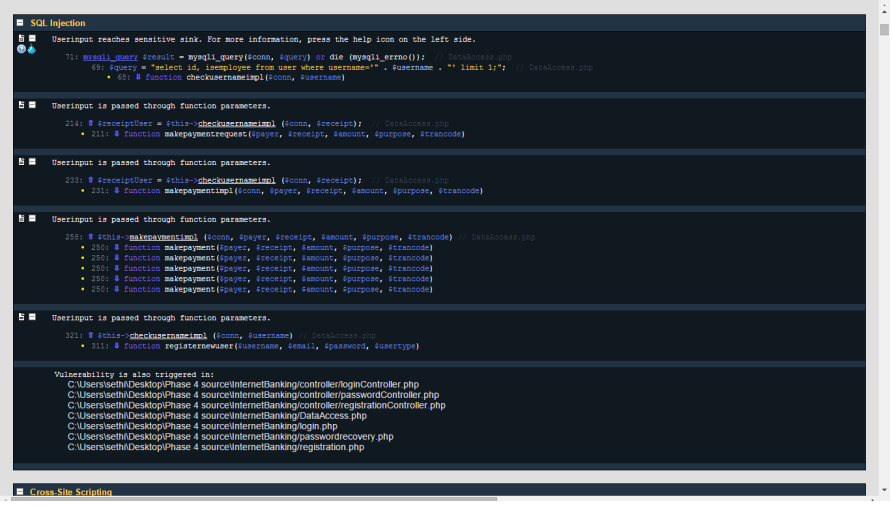


Figure 3.8: RIPS: SQL Injection vulnerability reported for Online Banking

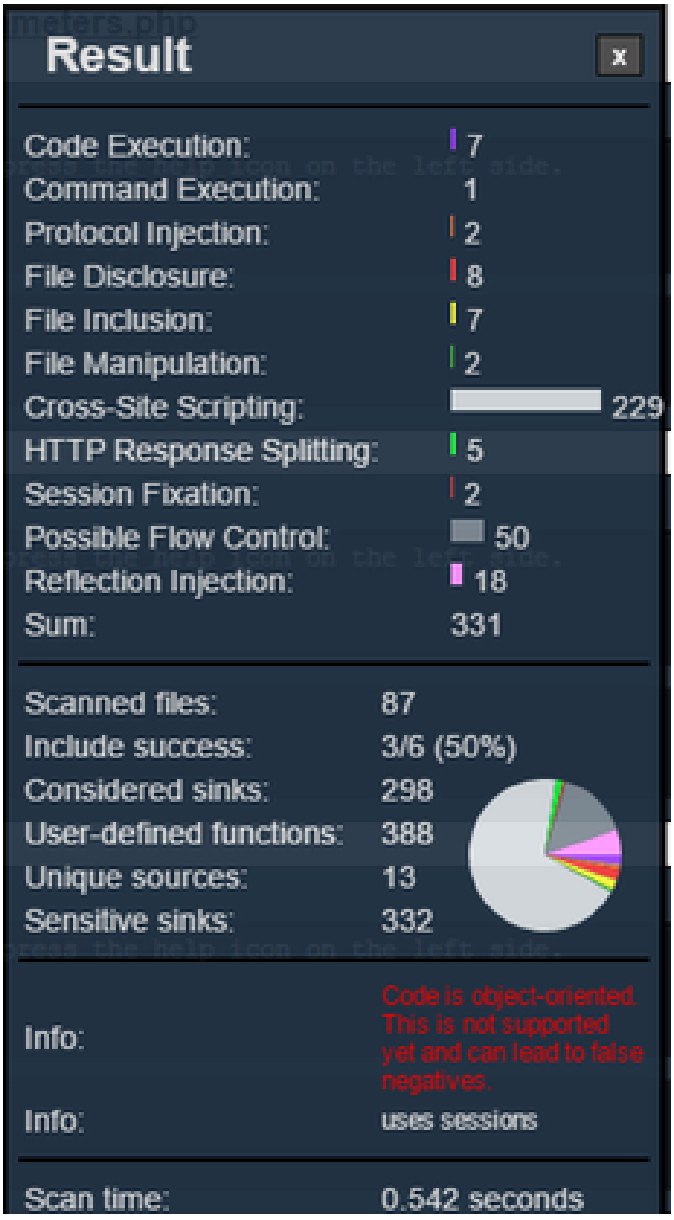


Figure 3.9: Overview of RIPS scan for SecureBank

```
Command Execution
Userinput reaches sensitive sink when function processbatchtransfer() is called.

293: exec exec($shell_command, $output, $return_var);
     282: $shell_command = $_SERVER['DOCUMENT_ROOT'] . "/../textparser/textparser " . escapeshellarg($uploaded_file_name) . " " . escapeshellarg($customer_id) . " " . escapeshe
          230: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, strlen((string)$i) + 1);  // if($i == 1) else ,
               227: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, 0);  // if($i == 1),
                    220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                         204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                         219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
                    223: $i = 1;
                    226: $pos = strrpos($uploaded_file_name, ".txt");  // if($i == 1),
                         220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                              204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                              219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
               223: $i = 1;
             • 229: $pos = strrpos($uploaded_file_name, "_" . ($i - 1) . ".txt");  // if($i == 1) else ,
               227: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, 0);  // if($i == 1),
                    220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                         204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                         219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
                    223: $i = 1;
                    226: $pos = strrpos($uploaded_file_name, ".txt");  // if($i == 1),
                         220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                              204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                              219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
               223: $i = 1;
               226: $pos = strrpos($uploaded_file_name, ".txt");  // if($i == 1),
                    220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                         204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                         219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
               223: $i = 1;
          227: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, 0);  // if($i == 1),

     requires:
          200: # function processbatchtransfer($request, $helper, $helper2, $customer)
```

Figure 3.10: RIPS: Command Execution vulnerability reported for SecureBank

```
File: C:\Users\sethi\Desktop\Phase 4 source\src/Helper/StringHelper.php

Code Execution
Userinput reaches sensitive sink. For more information, press the help icon on the left side.

     22: preg_replace_callback preg_replace_callback('/_([a-z])/', # function ($i){

hide all
```

Figure 3.11: RIPS: Code Execution vulnerability reported for SecureBank

```
File: C:\Users\sethi\Desktop\Phase 4 source\src/Controller/TransactionController.php

File Manipulation
Userinput reaches sensitive sink when function processbatchtransfer() is called. (Blind exploitation)

     236: move_uploaded_file move_uploaded_file($file['tmp_name'], $uploaded_file_name))
          205: $file = $request->getfile("make_transfer_via_file_upload", "file");
          230: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, strlen((string)$i) + 1);  // if($i == 1) else ,
               227: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, 0);  // if($i == 1),
                    220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                         204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                         219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
                    223: $i = 1;
                    226: $pos = strrpos($uploaded_file_name, ".txt");  // if($i == 1),
                         220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                              204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                              219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
               223: $i = 1;
             • 229: $pos = strrpos($uploaded_file_name, "_" . ($i - 1) . ".txt");  // if($i == 1) else ,
               227: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, 0);  // if($i == 1),
                    220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                         204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                         219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
                    223: $i = 1;
                    226: $pos = strrpos($uploaded_file_name, ".txt");  // if($i == 1),
                         220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                              204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                              219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
               223: $i = 1;
               226: $pos = strrpos($uploaded_file_name, ".txt");  // if($i == 1),
                    220: $uploaded_file_name = $upload_dir . $random_file_name . ".txt";
                         204: $upload_dir = $_SERVER['DOCUMENT_ROOT'] . '/tmp/';
                         219: $random_file_name = str_replace("/", "", $this->get("random")->getstring(10));
               223: $i = 1;
          227: $uploaded_file_name = substr_replace($uploaded_file_name, "_" . $i, $pos, 0);  // if($i == 1),

     requires:
          200: # function processbatchtransfer($request, $helper, $helper2, $customer)
```

Figure 3.12: RIPS: File Manipulation vulnerability reported for SecureBank

```
hide all

File: C:\Users\sethi\Desktop\Phase 4 source\src/Helper/FormHelper.php

Reflection Injection
Userinput reaches sensitive sink. For more information, press the help icon on the left side.

     108: call_user_func_array $field_raw = call_user_func_array(__NAMESPACE__ . '\SanitizationHelper::' . $sanitization, array($field_raw));
          102: foreach($config['sanitizations'] as $sanitization)
               91: foreach($this->field_configs as $name=>$config)

     requires:
          104: if($sanitization_helper_reflec->hasmethod($sanitization))
          106: if($sanitization_helper_reflec_method->ispublic() && $sanitization_helper_reflec_method->isstatic())
```

Figure 3.13: RIPS: Reflection Injection vulnerability reported for SecureBank

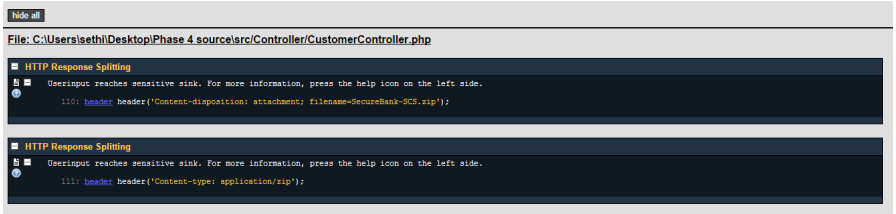


Figure 3.14: RIPS: HTTP Response Splitting vulnerability reported for SecureBank

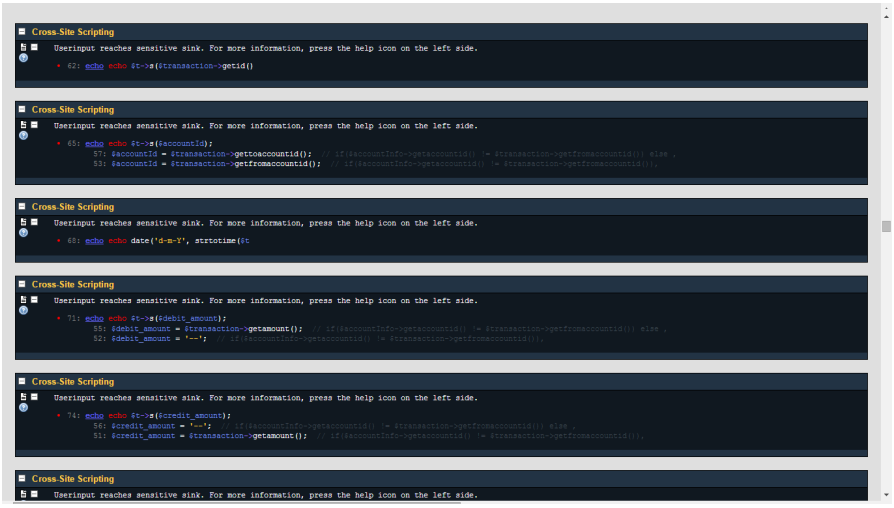


Figure 3.15: RIPS: Cross Site Scripting vulnerability reported for SecureBank

3.2.2 Kiuwan

Kiuwan was used for static analysis of PHP code for finding vulnerablities in the application.

Online Banking

Online Banking showed 2266 defects, out of which 76 were of high priority. Refer 3.16. The tool reported issues in the following categories:

- Security
- Efficiency
- Reliability
- Portability
- Maintainability

We analyzed the issues in the Security category and found some false positives. However, few issues turned out to be actual bugs.

- **Use of weak cryptographic hash** The user passwords are encrypted using `md5` before storing in the database. Since `MD5` has several weaknesses when used to generate cryptographic hashes, this is a serious vulnerability. This has been discussed in more detail in section 5.3.7. See 3.17 for the Kiuwan report.
- **Use of hardcoded password** The database credentials are stored in the file `DataAccess.php`. Also, these values are not defined at one place and used throughout code. Rather, the same hard-coded credentials can be found in several functions in the entire file. See 3.18 for the Kiuwan report.
- **Improper neutralization of input during web content generation (XSS)** It was found that it is possible to perform XSS attacks from the `Make Payment` interface. However, since it does not affect any functionality or other users, it could be considered low in priority. Refer 5.6.1 for more detail. See 3.19 for the Kiuwan report.

SecureBank

SecureBank showed 3010 defects, out of which 18 were of high priority. Refer 3.20. The tool reported issues in the following categories:

- Security
- Efficiency
- Reliability
- Portability
- Maintainability

We analyzed the issues in the Security category and found that all of them were false positives which was also confirmed by black box testing and manual code inspection.

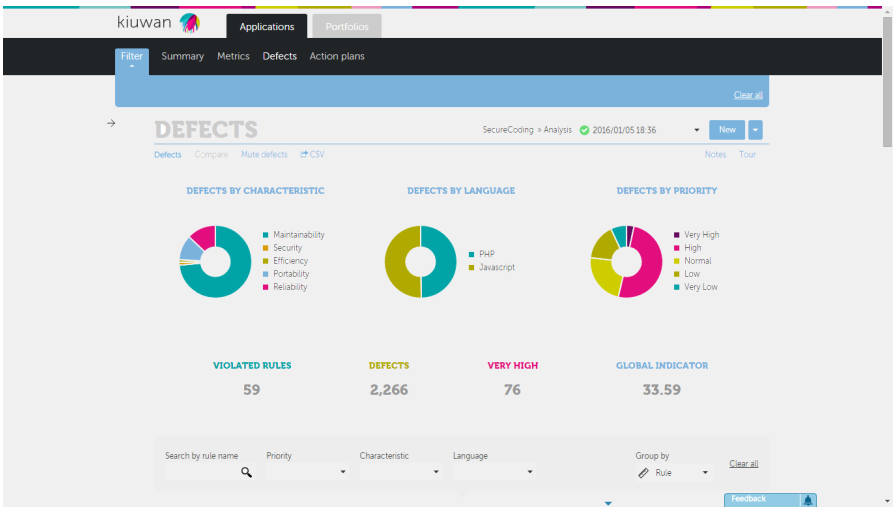


Figure 3.16: Overview of Kiuwan scan for Online Banking

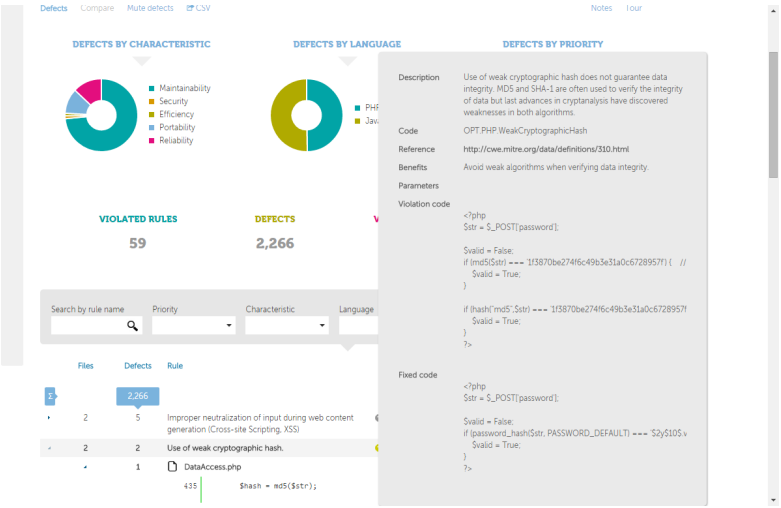


Figure 3.17: Kiuwan: Weak Cryptographic hash vulnerability reported for Online Banking

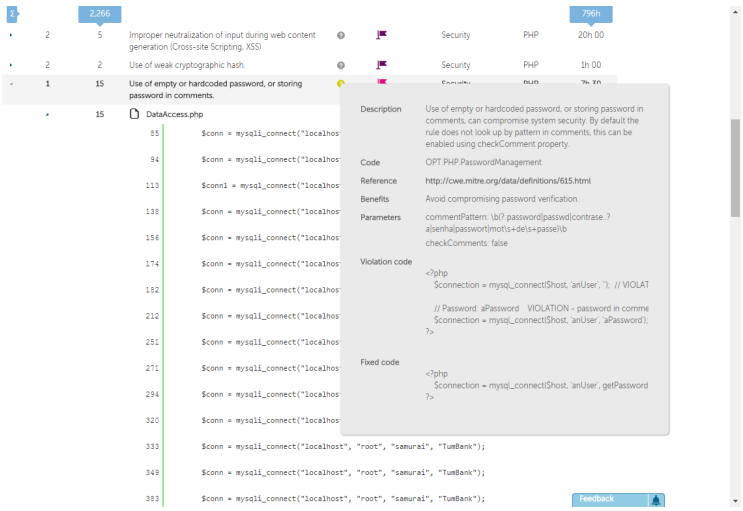


Figure 3.18: Kiuwan: Hardcoded Password vulnerability reported for Online Banking

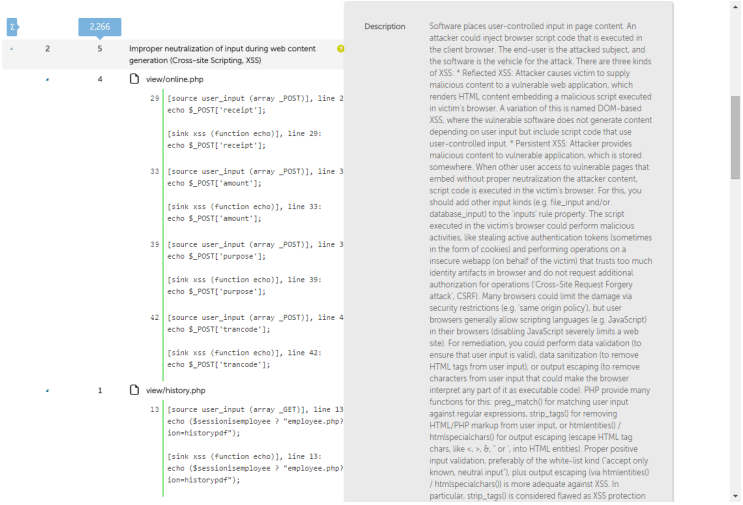


Figure 3.19: Kiuwan: XSS vulnerability reported for Online Banking

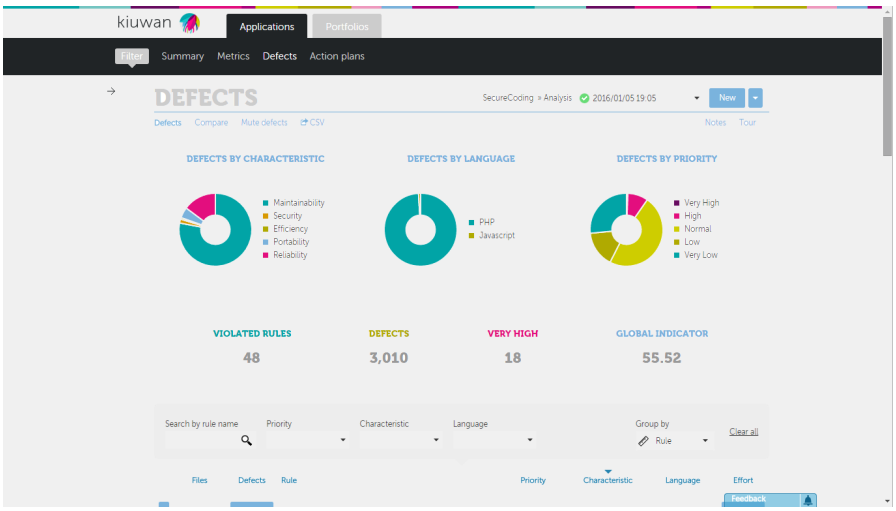


Figure 3.20: Overview of Kiuwan scan for SecureBank

3.2.3 RATS

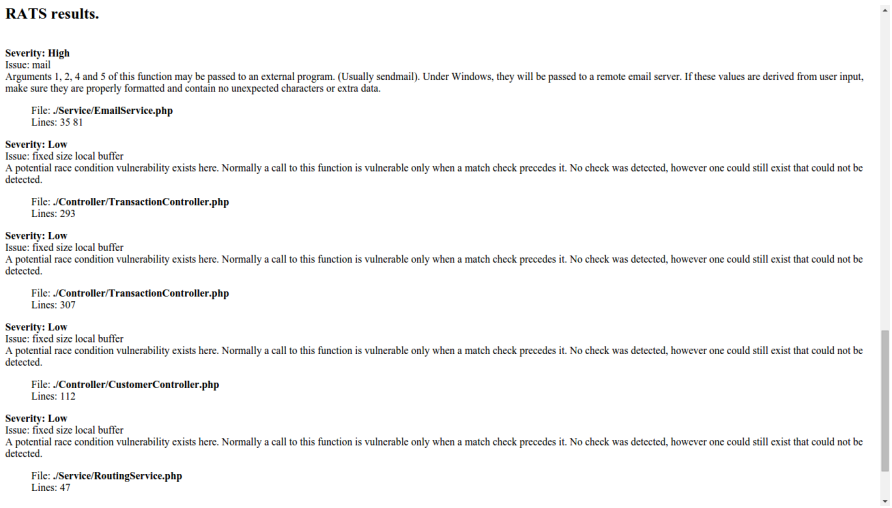
RIPS was used to scan PHP code, mainly for security-related programming errors.

Online Banking

RATS did not show any problems with the PHP code of Online Banking application.

SecureBank

RATS reported 5 issues in the PHP code of SecureBank, out of which 1 was of high severity. See 3.21. The issue was with the usage of the `mail` function of PHP. However, upon analyzing the code, it was found that the parameters to the function were not derived from user input. Hence this turned to be a false positive. The other low severity suspicions were also confirmed to be false positives after manual code inspection.



RATS results.

Severity: High
Issue: mail
Arguments 1, 2, 4 and 5 of this function may be passed to an external program. (Usually sendmail). Under Windows, they will be passed to a remote email server. If these values are derived from user input, make sure they are properly formatted and contain no unexpected characters or extra data.

File: **./Service/EmailService.php**
Lines: 35 81

Severity: Low
Issue: fixed size local buffer
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

File: **./Controller/TransactionController.php**
Lines: 293

Severity: Low
Issue: fixed size local buffer
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

File: **./Controller/TransactionController.php**
Lines: 307

Severity: Low
Issue: fixed size local buffer
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

File: **./Controller/CustomerController.php**
Lines: 112

Severity: Low
Issue: fixed size local buffer
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

File: **./Service/RoutingService.php**
Lines: 47

Figure 3.21: Overview of RATS scan for SecureBank

3.2.4 FindBugs

FindBugs was used for static analysis of Java code as the first step in finding vulnerablities in the Smart Card Simulator.

InternetBanking

FindBugs reported 3 issues in the Java code of InternetBanking, out of which 1 was categorized as `High confidence` and the other 2 as `Low confidence`. See 3.22 for the scan report. The issues are mostly related to coding practices & conventions and are listed below, while one of them could be of concern.

- Use of default encoding instead of specifying an explicit character encoding when converting to bytes
- Catch of an unthrown exception
- Call to swing method outside the swing thread

As seen, they are easy to correct and do not require much time or effort. The last one refers to the invocation of the `pack` function. The Swing methods `pack()` will create the associated peer for the frame. This causes the system to create the event dispatch thread. This makes things problematic because the event dispatch thread could be notifying listeners while pack and validate are still processing. This situation could result in two threads going through the Swing component-based GUI. This s a serious flaw that could result in deadlocks or other related threading issues.

SecureBank

FindBugs reported 9 issues in the Java code of SecureBank, out of which 1 was categorized as `Troubling`. See 3.23 for the scan report. The issue was about a possible null pointer dereference on exception path. Upon analyzing the code, it was found that the variable can be null only if an excpetion is thrown while trying to get an instance of `MessageDigest`, which is an issue, though very rare. However, this does not lead to any security problems. The other issues are related to coding practices & conventions and are listed below.

- Use of == or != instead of equals to compare strings
- Use of platform default encoding instead of specifying an explicit character encoding when opening a file
- Redundant null check of a variable, which is known to be non-null

- Use of non-localized `String.toUpperCase()` or `String.toLowerCase()`

As seen, they are easy to correct and do not require much time or effort. The last one is not relevant as the application does not support Internationalization.

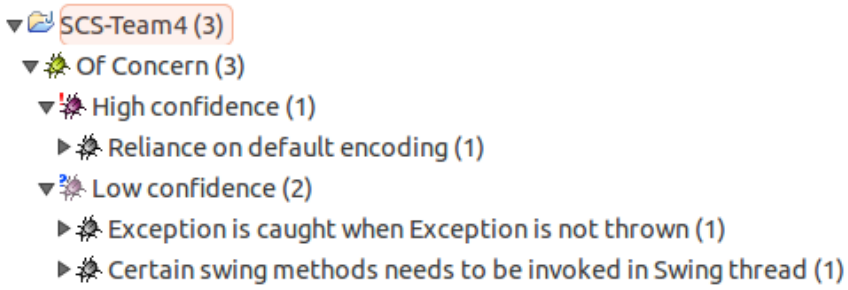


Figure 3.22: Overview of FindBugs scan for InternetBanking

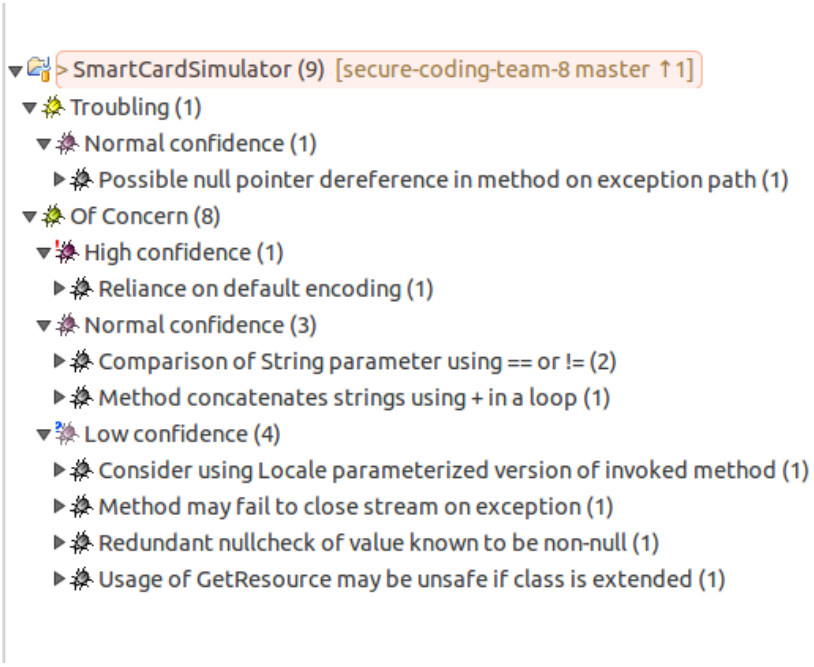


Figure 3.23: Overview of FindBugs scan for SecureBank

3.2.5 AllTanGenerationTests.java

AllTanGenerationTests.java is a custom JUnit script to check for vulnerabilities in the Java code of the SCS. It tests for basic validations such as blank or invalid inputs and duplicate TANs.

InternetBanking

The test reported 7 failures in the Java code of InternetBanking and passed 1 test successfully See 3.24 for the test report. 6 of the failed tests were related to validation of inputs. It was found that the SCS does not perform any validations and generates TAN irrespective of inputs. However, there are some checks on the PHP side, due to which these may be considered as false positives. However, 1 test `shouldNotGenerateDuplicateTan` reveals that the probability of generating duplicate TANs is extremely high and calculated to be `13%`. The number of duplicates were found to be 4 in 100, 232 in 1000, 1222 in 10000 TANs. Though there is check for duplicate TAN on the PHP side, if frequency of duplicate TANs generated by the SCS is high, the user will have to re-generate TANs multiple times for a successful transfer.

SecureBank

The test reported 1 failure in the Java code of SecureBank and passed 7 tests successfully. See 3.25 for the test report. The failed test `shouldNotGenerateDuplicateTan` revealed that the probability of duplicate TANs generated is high. However, this is due to the TAN generation logic that generates the same TAN upto 100 seconds. This is done to achieve expiration of the TAN and make it unusable after 100 seconds. Also, this prevents TANs generated at extremely short intervals. Also, after 100 seconds, the probability of duplicate TANs is found to be low. Hence, the test failure can be considered as a false positive.

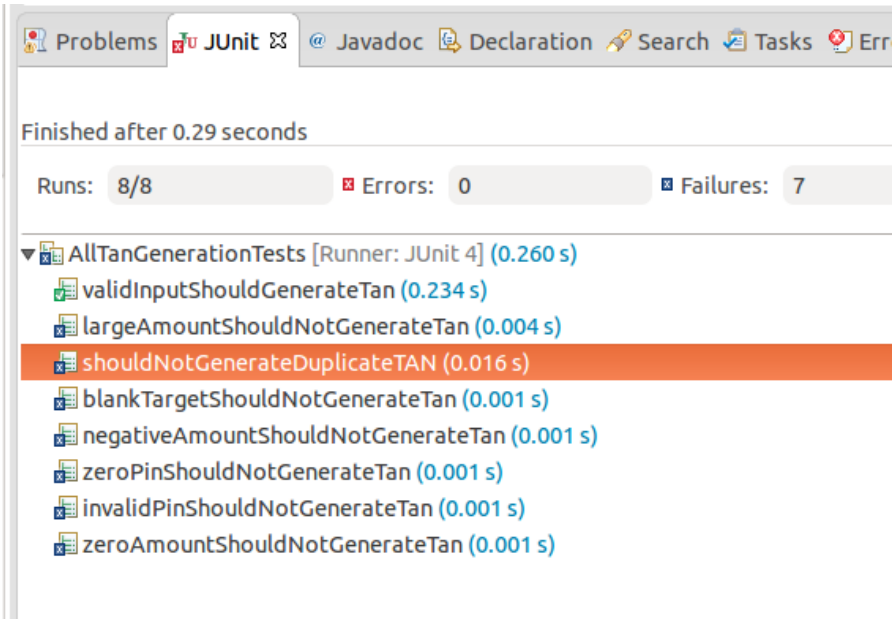


Figure 3.24: Overview of JUnit scan for InternetBanking

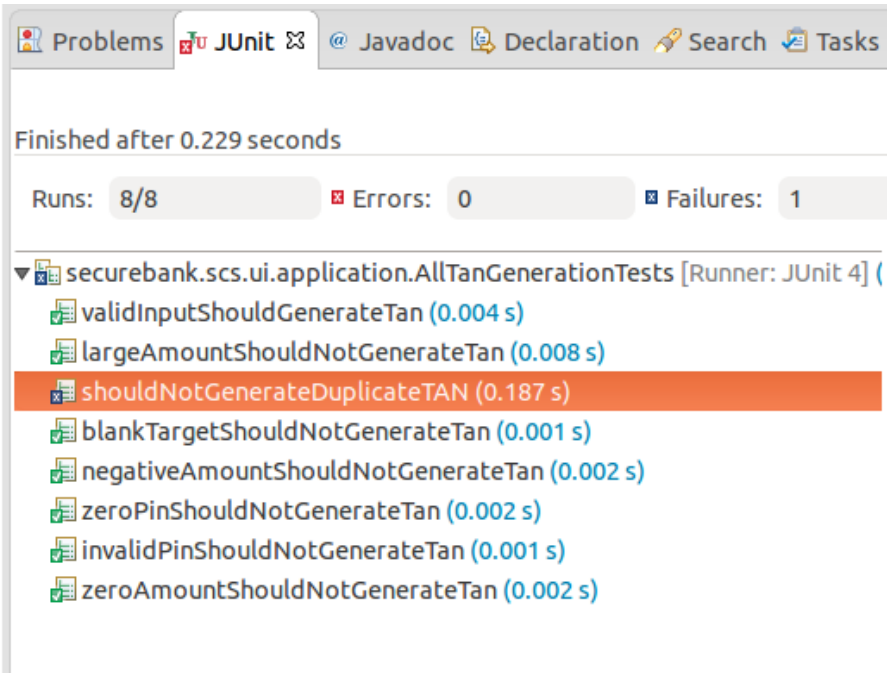


Figure 3.25: Overview of JUnit scan for SecureBank

3.2.6 IDA PRO Free

IDA PRO Free is a disassebmler and debugging tool for binary programms.
It produces assembler code and controll-flow-graphs of given binarys.
Additionally it can extract strings and calls to system functions or external libaries as well as programm entry points.
It is very well documented and intuitive to use.
We used this tool to disassemble the binary file `exec` we obtained from Internet Banking. It enabled us to examine the assembler code and functions used in the file. We coud rename certain variables to structurize and explain the code better. Additionally the possibility to add comments enabled us to add further informations to the code.
Using the informations gained this way we could reconsturct the C-code of the binary file.

```
loc_8048C44:
mov     eax, [ebp+arg_0]
mov     eax, [eax]
mov     [esp+28h+var_C], 0
mov     [esp+28h+var_10], 0
mov     [esp+28h+var_14], 0
mov     [esp+28h+var_18], offset aTumbank ; "TumBank"
mov     [esp+28h+var_1C], offset aSamurai ; "samurai"
mov     [esp+28h+var_20], offset aRoot ; "root"
mov     [esp+28h+var_24], offset aLocalhost ; "localhost"
mov     [esp+28h+connection], eax
call    _mysql_real_connect
mov     eax, 0
```

Figure 3.26: IDA PRO Free: Extracted Secrets

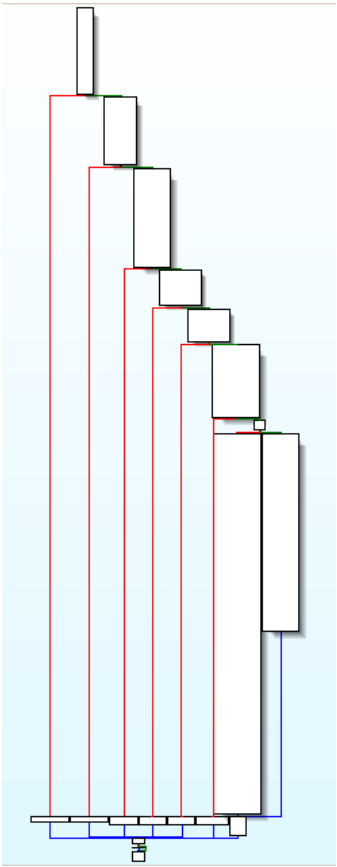


Figure 3.27: IDA PRO Free: Controll flow graph of a function

3.2.7 Valgrind

Valgrind was used for automatically detecting memory management bugs in the C executable. The command used was `valgrind -leak-check=full <command for parser>`.

It was used primarily to detect the number of allocations and freed blocks. For both application it showed that more heaps were allocated than freed. Refer to section 5.6.14 for more information.

```
HEAP SUMMARY:
    in use at exit: 125,848 bytes in 45 blocks
  total heap usage: 98 allocs, 53 frees, 164,610 bytes allocated
```

(a) Valgrind output for Online Banking

```
HEAP SUMMARY:
    in use at exit: 73,872 bytes in 21 blocks
  total heap usage: 119 allocs, 98 frees, 228,414 bytes allocated
```

(b) Valgrind output for SecureBank

Figure 3.28: Partial Valgrind output: heap management

3.2.8 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
```

3.2.9 Google Chrome Developer Tools

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

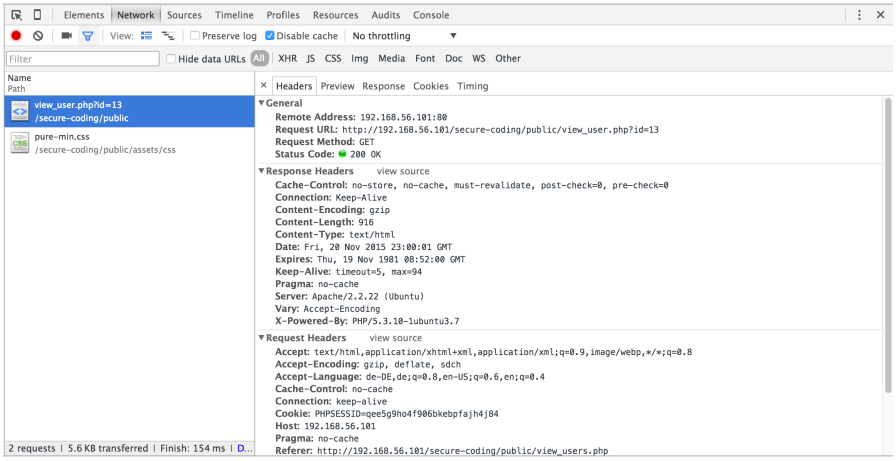


Figure 3.29: Google Chrome Developer Tools Network Analysis

3.2.10 objdump

objdump is a GNU binary utility to display information about one or more object files. It was used for disassembling the C executables. The command used was `objdump -D <C executable>`. It was used for a general overview of the assembly code.

4 Reverse Engineering

4.1 Java Smart Card Simulator

4.1.1 Decompilation

The Smart Card Simuator JAR file was obtained by downloading it from the Online Banking website. The JAR was decompiled using `JavaDecompiler (JD)`. It was found that it contained 3 files - `Main.class`, `Main$1.class` and `Main$2.class`. Upon decompilation, the Java code was obtained. However, there were errors in code as the output of JD is not re-compilable code. There were multiple instances of variables with `this.val$` prefix. Analyzing this, it was found that there could be inner classes. When a local final variable is accessed from an inner class, the decompiler shows it with a `this.val$` prefix. Observing the context and going through the code, appropriate modifications were made and executable code was generated. Refer 6.1 for the complete code.

4.1.2 Analysis of Working

- Firstly, there is no distinction between users of TAN by PDF or by SCS. All users have the option of using both options. Secondly, the PIN to the SCS can be found in the TAN containing the PDFs; which is a serious flaw. If an attacker gets hold of this PDF and has the user credentials, then all possible attacks can be perfomed. The attacker can not only use the 100 TANs from the PDF but also use the SCS for infinite TAN generation.
- The PIN for the SCS is generated in the PHP code using the code `$pin = rand(100000, 999999);`. So the PIN will always be a 6-digit number. The number of possibilities are `900000`. This is not very difficult to crack for an attacker using a Brute-force attack, given that the SCS does not have any lockout mechanism. There is no restriction on the number of times it can be used in an hour or even a day.
- The SCS takes 3 inputs - `PIN`, `Target` and `Amount` to generate the TAN. However, TANs are generated even without entering any of these details. Since it is possible to download the SCS from the website, without even having an account, this may be used by attackers to analyze the algorithm used for TAN generation.

- The SCS is not personalized i.e., it is not unique based on the user. Also, there are no hard-coded passwords or tokens for user identification stored anywhere in the JAR. This is a good thing as an attacker cannot get hold of any user-specific information by just getting access to the SCS.
- The SCS generates the TAN without any checks and all validations are handled at the PHP end. This includes valid target, amount and correct PIN. It would be recommended to have basic checks such as blank target, negative or zero amount and blank pin checks in the SCS itself.
- At the PHP side, a TAN is generated using a similar algorithm as in Java, using the target and amount entered in the Web UI and the SCS pin of the user fetched from the database. If the TAN(from SCS) entered by the user matches this TAN, payment is processed successfully.
- There is no expiration of the TAN. So it can be used anytime in the future. This could be vulnerable especially if a TAN has been compromised to an attacker.

4.2 Batch processing tool based on C

4.2.1 Disassembly

The C-Binary was optained by downloading it from the VM. We used `IDA PRO Free` to disassemple the binary code. We found out, that the file was composed in two important functions: The `main`-function and the function `mysql_query_function`. In the `main`-function the input text file gets parsed and in a loop the `mysql_query_function`-function gets called to make the changes in the database and apply the transactions. We used the flow diagram and disassebled code generated by `IDA PRO Free` to reconstruct the C-code the binary was generated from.

4.2.2 Analysis of Working

- Using the original binary with the provided test file does not work and exists with "Negative transactions not allowed"
- All query executions are vulnerable to SQL-Injection.
- The application repeatedly uses `strcpy()` and `strcat()` without length checks. This makes it vulnerable to buffer overflow attacks.

5 Detailed Test Report

5.1 Configuration and Deploy Management Testing

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

	Online Banking
Observation	It was found that the application allowed only text files. No other files were accepted.
Discovery	By manually inspecting the code, appropriate checks for text files were found. Refer 5.1 for the related code.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	It was found that the application allowed only text files. No other files were accepted.
Discovery	By manually inspecting the code, appropriate checks for text files were found. Refer 5.2 for the related code.
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

Comparison

Both applications handle file extensions properly and neither of them poses any vulnerability.

Listing 5.1: PHP code for checking file type from clientFunctions.php

```
if ($_FILES["tranfile"]["size"] > 500)
{
    $uploadOk = 0;
    throw new Exception("File␣too␣large!");

}

if($file_type != "txt")
{
    $uploadOk = 0;
    throw new Exception("Only␣txt␣files␣are␣accepted!");
}
```

Listing 5.2: PHP code for checking file type from TransactionController.php

```
if ($file['type'] != "text/plain") {
    $this->get("flash_bag")->add(_OPERATION_FAILURE,
    "The␣uploaded␣file␣must␣be␣a␣plain␣text␣file",
    "error");
    $this->get("routing")->redirect("make_transfer_get",
    array("form" => $helper, "form2" => $helper2));
    return;
} else if ($file['error'] == 2) {
    $this->get("flash_bag")->add(_OPERATION_FAILURE,
    "The␣uploaded␣file␣size␣exceeds␣the␣maximum␣of␣1␣MB",
    "error");
    $this->get("routing")->redirect("make_transfer_get",
    array("form" => $helper, "form2" => $helper2));
    return;
}
```

5.1.2 Test HTTP Methods - OTG-CONFIG-006

	Online Banking
Observation	It was found that there is no reference to methods other than GET and POST, in the source code.
Discovery	The PHP source code was examined using `grep` command and this was confirmed using the Nmap tool which revealed that the server allows only 4 methods : HEAD, GET, POST, OPTIONS. See Figure 5.1.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	It was found that there is no reference to methods other than GET and POST, in the source code.
Discovery	The PHP source code was examined using `grep` command.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

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

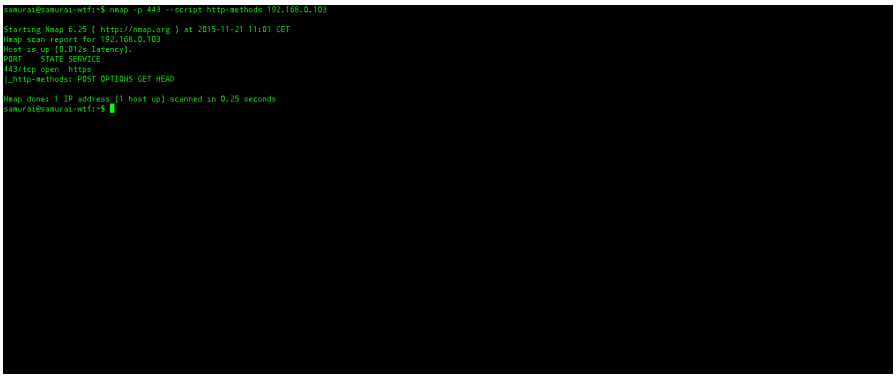


Figure 5.1: Nmap - Check for allowed HTTP methods

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

	Online Banking
Observation	Online Banking does not use HTTPS and therefore does not implement HSTS.
Discovery	Accessing the site with `https://IP_ADDRESS/Online Banking/` shows that HTTPS is not used. Furthermore, using the command `curl -s -D- http://IP_ADDRESS/Online Banking/ | grep Strict` did not yield any results and therefore the header for `Strict-Transport-Security` is not set.
Likelihood	N/A
Impact	N/A
Recommendations	Use and enforce HTTPS for a secure communication. If using HTTPS, the HSTS header for `Strict-Transport-Security` has to be set to `max-age=60000; includeSubDomains`.
CVSS	N/A

	SecureBank
Observation	The HSTS header for `Strict-Transport-Security` is set to `max-age=60000; includeSubDomains`.
Discovery	Using the command `curl -s -D- https://IP_ADDRESS/ | grep Strict` shows that the header for `Strict-Transport-Security` is set.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking does not use HTTPS and therefore the header for `Strict-Transport-Security` is not set. SecureBank enforces HTTPS and the header for `Strict-Transport-Security` is set to `max-age=60000; includeSubDomains`.

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

	Online Banking
Observation	Online Banking 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
Observation	Online Banking 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.

5.2 Identity Management Testing

5.2.1 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	✓	✓	✓	✗
Download SCS	✓	✓	✓	✗
Reset own Password	✓	✓	✓	✗
Logout	✓	✓	✓	✗
Register as Employee	✗	✗	✗	✓
Register as Customer	✗	✗	✗	✓
Approve/Disapprove Customer	✓	✓	✗	✗
Approve/Disapprove Employee	✓	✗	✗	✗

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

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	✓	✓	✗
Download SCS	✓	✓	✗
Reset own Password	✓	✓	✗
Logout	✓	✓	✗
Register as Employee	✗	✗	✓
Register as Customer	✗	✗	✓
Approve/Disapprove Customer	✓	✗	✗
Approve/Disapprove Employee	✓	✗	✗

Online Banking does not use a Admin user. We find it critical to differenciate between Employee and Admin to reduce the possibilities of abuse.

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

Likelihood	Employees can abusively invite non-authorized persons and give them administrative rights.	
Impact	Foreign persons gain administrative privileges.	
Recommendations	Have the Role Admin for Approval of Employees	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	High
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	High
	Availability Impact	High

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

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

	No Abnormalities could be determined.
Discovery	We examined the Application by manually following the provided links and UI elements
Likelihood	N/A

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

Comparison

Online Banking does not use an Admin user and gives every Employee the right to invite other Employees.

5.2.2 Test User Registration Process - OTG-IDENT-002

	Online Banking CVSS Score: 4.3
Observation	User Registration proccess and constraints: • Anyone can register as Employee or Customer. The Registration uses one Form for both Registrations • Identification Requirements are "Username", "Email", "Password" • Customers have to be verified by an Employee • Employees have to be verified by an Employee • The same Email address cannot be used twice for a user or employee. • The same Username address cannot be used twice for a user or employee. • Email adresses are not checked for identity Problematic: • Email adresses are not checked for identity • Address, City or Postal code are not verified
Discovery	To test the Registration proccess we manually registered some customer and employee accounts with different and same email addresses.
Likelihood	There is the possibillity that a Person registers with a foreign identity. To prohibit this the manual verification proccess has to ensure that idientities are properly verified. As there is not much information provided about the applicant this can be difficult. Therefore the Likelihood of an atack is increased.
Impact	If someone registers as another person he can possibly use his account to commit crimes and the actions could not be traced back to him or would be blamed on the persion that the account was registered on.

Recommendations	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 CVSS Score: 4.3
Observation	User Registration proccess and constraints: • Anyone can register as Employee or Customer. The Registration forms are seperated • Identification Requirements are “First name”, “Last Name”, “Email”, “Password” for Employee • Identification Requirements are “First name”, “Last Name”, “Address”, “Postal code”, “City”, “Email”, “Password” for Customer • Customers have to be verified by an Employee • Employees have to be verified by an Admin • The same Email address cannot be used twice for a user and cannot be used twice for a employee. The same email can be used for a customer account and a employee account • When the same email is used for an employee and a customer one allways get logged in as Customer • Email adresses are not checked for identity • Address, City and Postal code are not verified Problematic: • When the same email is used for an employee and a customer one allways get logged in as Customer • Email adresses are not checked for identity • Address, City and Postal code are not verified
Discovery	To test the Registration proccess we manually registered some customer and employee accounts with different and same email addresses.

Likelihood	There is the possibillity that a Person registers with a foreign identity. To prohibit this the manual verification proccess has to ensure that idientities are properly verified. This is a lot easier with the additionally provided informations address, city, postal code, but as these values are not automatically verified it has be done by a human. This fact increases the likelihood of an atack.	
Impact	If someone registers as another person he can possibly use his account to commit crimes and the actions could not be traced back to him or would be blamed on the persion that the account was registered on.	
Recommendations	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 Online Banking, SecureBank asks more informations on user registration but as the informations are not properly verified by the system this advantage is only superficial.

5.2.3 Test Account Provisioning Process - OTG-IDENT-003

	Online Banking
Observation	A customer cannot approve or reject pending registrations of other employees or customers. This can only be done by authorized employees in the application and is not exposed to other users. However, an employee can reject already approved customers or employees, with no further effect to the system.
Discovery	This flaw has been exposed using the Advanced Rest Client. Steps are as follows: • Login as an Employee and enter the URL - `http://<IP-address>/Online Banking/employee.php`. The details of all registered users are shown. • Note the User Id of one of the users. Consider this value is `xyz`. • Open the Advanced Rest Client and enter the URL - `http://<IP-address>/ Online Banking/employee.php ?action=approveregistrations`. • Select the POST method and in the payload, enter `requestid=xyz&approve=reject`. Select the "Content-type" header to `application/x-www-form-urlencoded`. • Click on "Send" and observe that the response contains the text "Account request rejected successfully". • However, upon trying to login as user `xyz`, there are no failures and the user can function as earlier.

	• Upon analyzing the database, it was found that there are 2 tables - `userrequest` and `user`. All registrations are stored in `userrequest` and approved users are stored in `user`. Upon approval or rejection, `approveddate` or `rejecteddate` column of the corresponding entry in the `userrequest` table is updated. In case of approval, a new record is also inserted into the `user` table. In the above scenario, `approveddate` and `rejecteddate` both get set for the user `xyz`. But login continues to work as it is based on the `user` table.
Likelihood	N/A
Impact	N/A
Recommendations	Approval or rejection operations should be restricted on already approved users.
CVSS	N/A

	SecureBank
Observation	A customer cannot approve or reject pending registrations of other employees or customers. This can only be done by authorized employee and is not exposed to other users. Also, no operations can be performed on already approved users, not even by authorized employees.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Though no vulnerability is exposed in Online Banking application, it is still possible to modify the database, thus exposing a flaw. Considering this, SecureBank is more secure in this regard.

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

	Online Banking CVSS Score: 6.5
Observation	It was found that the application responds with the same error messages for every client request that produces a failed authentication. This has been tested in the Login page.
Discovery	This test was performed manually by trying various combinations of email and password. Steps are as follows: • **Scenario 1 -** Testing for Valid user with right password – Open the Login page and enter a valid email and password. Click on the Submit button. – The user is redirected to the Transactions page without any success message. • **Scenario 2 -** Testing for Valid user with wrong password – Open the Login page and enter a valid email with an incorrect password. Click on the Submit button. – An error message is displayed that reads `Login and/or password not correct.`, and the user stays on Login page. Also, the data entered in the form is cleared off. • **Scenario 3-** Testing for non-existent User – Open the Login page and enter an incorrect email and password. Click on the Submit button. – An error message is displayed that reads `Login and/or password not correct.`, and the user stays on Login page. Also, the data entered in the form is cleared off.

	• **Scenario 4-** Registering a new User – Open the Registration page and enter thr details. Click on the Submit button. – An error message is displayed that reads `Username already in use, please choose another` or `Email-id is already in use, please choose another.` and the user stays on the same page. Refer 5.3 for the related code excerpt.	
Likelihood	Likelihood is low as some skills are required to perform dictionary attcks or to use password-cracking softwares and the time to guess a valid user.	
Impact	Impact is high because if a user account is hacked, all permitted operations can be performed.	
Recommendations	It is recommended to have consistent messages on failed attempts. Also, a lockout mechanism should be implemented.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	Low
	Availability Impact	None

	SecureBank
Observation	It was found that the application responds with different error messages for different incorrect login requests. However, it is not possible to enumerate users. There is also a lockout mechanism which locks the account after 5 failed login attempts.
Discovery	This test was performed manually by trying various combinations of email and password. Steps are as follows: • **Scenario 1 -** Testing for Valid user with right password – Open the Login page and enter a valid email and password. Click on the Submit button. – The user is redirected to the Home page without any success message. • **Scenario 2 -** Testing for Valid user with wrong password – Open the Login page and enter a valid email with an incorrect password. Click on the Submit button. – An error message is displayed that reads `Login failed - Either the e-mail or the password is wrong.`, and the user stays on Login page. Also, the data entered in the form is retained. • **Scenario 3-** Testing for non-existent User – Open the Login page and enter an incorrect email and password. Click on the Submit button. – An error message is displayed that reads `Login failed - There is no account with this email.`, and the user stays on Login page. Also, the data entered in the form is retained.
Likelihood	N/A
Impact	N/A

Recommendations	It is recommended to have consistent messages on failed attempts.
CVSS	N/A

Comparison

In Online Banking, it is possible to enumerate usernames and is hence vulnerable, also owing to the non-existent lockout mechanism. In case of SecureBank, though the error messages differ, no vulnerability has been detected owing to the absence of username-baseed login and the presence of a lockout mechanism. Hence, SecureBank is more vulnerable than Online Banking.

Listing 5.3: PHP code for checking user from registrationController.php

```
$user = $db->checkUsername($_POST['username']);

if($user!=NULL){
        $error .= 'Username already in use, please choose another.<br>';
}
if(!filter_var($email,FILTER_VALIDATE_EMAIL)){
        $error .= 'Email is invalid<br>';
}
else{
        $user = $db->checkEmail($_POST['email']);
        if($user!=NULL){
                $error .= 'Email-id is already in use,
                                please choose another.<br>';
        }
}
```

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

	Online Banking	CVSS Score: 6.5
Observation	It has been observed that authentication is based on a combination of User-name & password and that user-names are provided by the users themselves at the time of registration. It is possible to trigger dictionary attacks for user-names and passwords. Though the error messages are consistent for all failed login attempts, it is possible to perform the attack infinite number of times due to the absence of a lockout mechanism.	
Discovery	This has been tested in the Login and Registration pages.	
Likelihood	Likelihood is low as some skills are required to perform dictionary attcks or to use password-cracking softwares and the time to guess a valid user, given that the error messages are consistent is also high.	
Impact	Impact is high because if a user account is hacked, all permitted operations can be performed.	
Recommendations	A lockout mechanism should be implemented.	
CVSS	Attack Vector	Network
	Attack Complexity	High
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	Low
	Availability Impact	None

	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 5.2.4.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking application is vulnerable to user-name and password attacks and there is no lockout policy. In case of SecureBank, authentication is solely based on e-mail & password combination and there is also a lockout mechanism in-place. Thus, SecureBank is more secure in this regard.

5.3 Authentication Testing

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

	Online Banking CVSS Score: 8.6
Observation	It has been found that the forms in the application submit to an insecure HTTP target. Parameters such as User name, Password, TAN number etc. are not encrypted.
Discovery	In the observed source code, there was no hint to encryption. Neither is Apache configured to provide any SSL/TLS encryption. This was found by examining the configuration file `/etc/apache2/sites-enabled/000-default`. To confirm whether the application works on HTTP or HTTPS via a black-box test, cURL was used. Steps are as follows: • Open the terminal and type `curl https://<IP-address>`. The response states unknown protocol. • To get a detailed response, use `curl -verbose https://<IP-address>`. • Now try with `curl http://<IP-address>`. The response indicates a successful connection and the output of the request. 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. Any attacker; even one with no experience will notice that there is no encryption available, upon visiting the bank's website for the first time.
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 or even transactions being hijacked. It becomes very easy for a Man-In-The-Middle attack.

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	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	Low
	Availability Impact	Low

	SecureBank
Observation	It has been found that all accesses to the banking application are via `https` i.e., secure http connections. Unencrypted traffic over `http` is also redirected to the encrypted version of the webpage.
Discovery	This has been confirmed by examining the configuration file `secure-bank.conf` where the SSL and HSTS details can be found.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is secure owing to the secure connection over encrypted channel whereas Online Banking application exposes serious vulnerability.

5.3.2 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 shows the registration forms for both banks. It is obvious that there are no default credentials. Therefore there is no vulnerability regarding default credentials.

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

	Online Banking	CVSS Score: 7.5
Observation	Logging in with a false password can be repeated numerous times without being locked out.	
Discovery	The file `DataAccess.php` has the function `Authenticate`, which checks whether there is a user with the username and the hashed password. If that user exists, the function returns the user, otherwise it returns `null`. There is no code for counting the number of times a user has tried to login, therefore no lock out mechanisms are implemented.	
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
Observation	Logging in with a false password can be repeated 5 times before the account is locked.
Discovery	In `Auth/DBAuthProvider.php` the account is checked for the number of login attempts. If that number is bigger or equal to 4, the login attempt number is set to 5 and the account is locked for 60 minutes.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking does not have any lock out mechanisms, whereas SecureBank locks the account after several unsuccessful login attempts.

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

	Online Banking
Observation	There were several observations: • Session ID Prediction is not possible • Parameter modification is not possible • SQL Injection is not possible in the Login form • Direct Page access is not possible
Discovery	As Online Banking uses the standard PHP 5.3 way of generating Session IDs it is not vulnerable to Session ID Prediction. See 5.4. Parameter modification was eliminated by manually looking at the url parameters in the app. For SQL Injection please refer to OTG-INPVAL-005. Direct page access is not possible as the app uses checks to determine if the applications is bootstrapped and the user is logged in. See 5.5
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	There were several observations: • Direct Page access is not possible • Parameter modification is not possible • Session ID Prediction is not possible • SQL Injection is not possible in the Login form
Discovery	As SecureBank uses the standard PHP 5.3 way of generating Session IDs it is not vulnerable to Session ID Prediction. Parameter modification was eliminated by manually looking at the url parameters in the app. For SQL Injection please refer to OTG-INPVAL-005 Direct page access has no effect as the Application uses Object a router that blocks direct access.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

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

Listing 5.4: PHP code for sessions from auth.php

```
session_start();
$currentCookieParams = session_get_cookie_params();
session_set_cookie_params(
    $currentCookieParams["lifetime"],
    $currentCookieParams["path"],
    $currentCookieParams["domain"],
    true,
    true
);
```

Listing 5.5: PHP code for auth-check in clientController.php

```
if (!defined('MyConst')){
    die('not␣permitted');
}
if ($sessionisemployee){
    die('Not␣client.');
}
```

5.3.5 Test remember password functionality - OTG-AUTHN-005

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

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

	Online Banking
Observation	No settings related to caching were found in the server configuration. However, the HTTP responses contained the header `Cache-Control: must-revalidate, pre-check=0, post-check=0, no-store, no-cache`, which relates to the Apache module `mod_expires`. It is also found that sensitive data is not saved anywhere throughout the application.
Discovery	Manual inspection of `.htaccess` file and the Apache configuration was done.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	No settings related to caching were found in the server configuration. However, the HTTP responses contained the header `Cache-Control: must-revalidate, pre-check=0, post-check=0, no-store, no-cache`, which relates to the Apache module `mod_expires`. It is also found that sensitive data is not saved anywhere throughout the application.
Discovery	Manual inspection of `.htaccess` file and the Apache configuration was done.
Likelihood	Same as described for Online Banking.
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.

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

	Online Banking CVSS Score: 8.3
Observation	It has been observed that there is restriction on the choice of passwords during registration. User has to enter minimum 6 characters which should contain atleast one uppercase character, one lowercase character, one number and one non-alphanumeric character. This makes it difficult to crack. But the reset password option does not have such restriction and user might set a simple password which can be easily cracked through different tools.
Discovery	• Manual inspection of code was done to check for restrictions on password. Refer 5.6 for the related code excerpts. • Further inspection of code reveals that the password has been encrypted using `md5` before storing in the database. Refer 5.7 for the related code excerpt. So for users having the same password, the entries in the database will be the same. Anyone having access to the database can find this easily. • MD5 is a hash function and two different strings can absolutely generate colliding MD5 codes. In particular, MD5 codes have a fixed length; so the possible number of MD5 codes is limited. The number of strings (of any length), however, is definitely unlimited so it logically follows that there must be collisions. Though algorithms such as MD5 are designed to minimize the probability of a Hash collision, it is still possible. However the probability of two randomly chosen strings having the same MD5 hash is very low. So it becomes difficult to guess the string from the md5 hash. But using sites that have huge repositories of strings and their corresponding md5 hashes, it is possible to come up with a valid string. One such site is `http://www.md5online.org/` using which we were able to retrieve passwords from the md5 values.

Likelihood	Likelihood is high. The attacker can use Brute Force to crack the passwords as there is no lockout mechanism. Moreover, since there is no restriction enforced on passwords(during password reset), 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	Locking out of account after certain unsuccessful tries should be done. In this way brute force attack gets complicated. Also, restrictions on passwords should be applicable both at the time of registration and during password change.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	Low
	Availability Impact	High

	SecureBank
Observation	It has been observed that there is restriction on the choice of passwords during registration. User needs to enter minimum 6 characters which should contain atleast one uppercase character, one lowercase and one number. This reveals that passwords of users cannot be cracked easily and this vulnerability has been reduced in the Login page. There is also a lockout mechanism to restrict Brute-force attacks.

Discovery	• Manual inspection of code was done to check for restrictions on password. • Before saving the password in the database, it has been encrypted using the `crypt` function based on a random `salt`. So even for users having the same password, the database entries will be different. Refer 5.8 for the related code excerpt.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

On comparing, SecureBank is better than Online Banking as it has a strong password policy, lockout mechanism and restrictions on password even during password change.

Listing 5.6: PHP code for checking password strength from DataAccess.php

```
function checkPassword($pwd) {
    $errors = "";
    $flag = 0;

    if( strlen($pwd) < 6 ) {
        $errors .= "Password␣too␣short,␣length␣should␣be␣>=6!";
        $flag = 1;
    }

    if( strlen($pwd) > 15 ) {
        $flag = 1;
        $errors .= "Password␣too␣long,␣length␣should␣be␣<=20!";
    }
    if( (!preg_match("#[0−9]+#", $pwd)) || (!preg_match("#[a−z]+#", $pwd))
    ||(!preg_match("#[A−Z]+#", $pwd))||(!preg_match("#\W+#", $pwd))) {
        if($flag==1){
            $errors .= "␣";
            $flag = 2;
        }
        $errors .= "Password␣must␣include␣at␣least␣one␣number,
```

```
␣␣␣␣␣␣␣␣lowercase,␣uppercase␣and␣symbol!<br>";
    }

    if($flag ==1)
        $errors .= "<br>";
    return $errors;
}
```

Listing 5.7: PHP code for encryption of password from DataAccess.php

```
$query = "insert␣into␣userrequest(username,␣password,␣email,
␣␣␣␣isemployee,␣createdate)␣"
    ."values("
    .""".$username."',␣"
    ."md5('".$password."'),␣"
    .""".$email."',␣"
    .($usertype == 'employee' ? 1 : 0 )
    .",␣now()␣);";
```

Listing 5.8: PHP code for encryption of password from RegistrationController.php

```
$salt = $this−>get("random")−>getString(16);
$model−>setSalt($salt);
$model−>setPassword(crypt($model−>getPasswordPlain(), $salt));
```

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

	Online Banking
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
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 Online Banking.
CVSS	N/A

Comparison

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

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

	OnlineBanking CVSS Score: 6.7
Observation	Password change functionality has been provided to the user though which the user receives a secret key which can then be used to change the password. The process is weak since any user can invoke such a process if they know any registered email id.
Discovery	Manual inspection of code revealed the following flaws in implementation. • For password change, the system doesn't check whether it is a registered email or not, and straightaway shoots an email to the specified email; thus allowing to change the password even if the email is not registered. • Rules applicable on password during Registration are not checked at the time of Password change; thus leading to weak passwords. Password field can even be left blank, which is extremely vulnerable. • There is no timeout mechanism implemented after reset password functionality was invoked. Hence there can be misuse of the same by an attacker if he gets access of the email id at any point in time. • If password reset was invoked more than once and while resetting the password, an old secret key was used; the process is completed successfully and password changes with outdated secret key. • If an attacker gets the access of the victim's email id , he/she can reset the password. If the username of the victim is known, then impersonation as user is also possible. • After resetting of password, brute force attack can be performed on password as password policy was not enforced during password reset (in case of registered users). Refer 5.9 for the code excerpt from `passwordController.php`.

Likelihood	Likelihood is high, since anybody having the knowledge of victim's email id can invoke the process. There is no security check before invoking the reset password process.	
Impact	Impact is high, since this vulnerability causes Denial of Service attack for the victim and there can be a total compromise of the application to the attacker.	
Recommendations	Password reset functionality should be implemented along with a security measure such as a security question. It can be used to avoid random invocation of the password change functionality. Along with this a timeout mechanism should also be implemented during which the password can be reset, thus reducing the time period of the attacker's exploitation.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	High
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	High
	Integrity Impact	Low
	Availability Impact	High

	SecureBank
Observation	Password change functionality has been provided to the user though which user receives a link to reset password through email, that is valid for 30 minutes. This can be an issue if the registered email id is known to the attacker.
Discovery	Manual inspection of code was done.
Likelihood	N/A
Impact	N/A

Recommendations	It would be better if security question is also implemented during password change, thus raising the difficulty even if the e-mail is compromised.
CVSS	N/A

Comparison

SecureBank is clearly more secure in this regard, compared to Online Banking that has multiple flaws exposed.

Listing 5.9: PHP code for change password functionality from passwordController.php

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {

    include_once("DataAccess.php");
    $db = new DataAccess();
    $display = "setpaswd";

    if (isset($_POST['key']) && isset($_POST['paswd'])){
        $error = "";

        $paswd = $_POST['paswd'];
        $conf = $_POST['conf'];
        $key = $_POST['key'];

        if ($paswd != $conf){
            $error = "password␣doesn't␣match␣confirmation";
        }
        if ($error == ""){
            try{
                $db->ChangePaswd($key, $paswd);
                $display = "completed";
            } catch(Exception $ex){
                $error = $ex->getMessage();
            }
        }
    } else if (isset($_POST['email'])){
            $newkey = uniqid("", true).uniqid("", true);
            $newkey = str_replace(".", "", $newkey);

            $db->SavePasswordRecoveryKey($_POST['email'], $newkey);
            sendMail($_POST['email'], "Online␣Banking␣Password␣Recovery",
            "Your␣secret␣key␣is:␣". $newkey);
    }
}
```

5.3.10 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.

5.4 Authorization Testing

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

	Online Banking	CVSS Score: 7.5
Observation	Generally speaking, files can be downloaded from and directory listing is enable for all folders with three exceptions. Accessing files in the folders `controller`, `model` and `view` shows a blank page. For example you can download the main c-file for the parser via the link `http://IP_ADDRESS/Online Banking/parser/main.c`. Although the page shows "not permitted", the browser lets the user save the file.	
Discovery	In each of the folders `controller`, `model` and `view` there is a `index.php` file which simply outputs a blank page. For the other folders you could access the path for the files and download them.	
Likelihood	If a hacker knows the folder structure of the application, he/she could easily download all the source code files and look for vulnerabilities.	
Impact	Depending on the content of the files a hacker could use that information for attacking the application.	
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
Observation	Directory listing is disabled.
Discovery	In the `src/.htaccess` file there is the line with `Options -Indexes`, which disables directory listing.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking has directory listing for some folders enabled, whereas SecureBank does not. Therefore it is possible to download some source code from Online Banking by directly accessing the files/folders.

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

	Online Banking
Observation	It has been noted that it is not possible to access administrative data though the attacker is logged in as a user with ordinary privileges.
Discovery	A clear distinction of privileges among the users has been defined due to which a client cannot access employees pages and employee cannot see client specific pages. It has been implemented in `clientController.php` and `employeeController.php` based on session values. Refer 5.26 for code related to client check and 5.27 for code related to employee check based on session value.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	It has been noted that it is not possible to access administrative data though the attacker is logged in as a user with ordinary privileges.
Discovery	A clear distinction of privileges among the users has been defined in the code. It has been implemented in every function which is being called on an operation based on session values. Refer 5.28 for code related to customer check and 5.29 for code related to employee check based on session value. Similar checks are implemented for all client, employee and administrator operations.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications are secure in this aspect, since functionality is divided based on groups of users and privileges are assigned based on type of user.

Listing 5.10: PHP code for client check from clientController.php

```
if ($sessionisemployee){
    die('Not client.');
}
```

Listing 5.11: PHP code for employee check from employeeController.php

```
if ($sessionisemployee != 1){
    die('Not employee.');
}
```

Listing 5.12: PHP code for client check in loadOverview function from CustomerController.php

```
$customer = $this->get("auth")->check(_GROUP_USER);
```

Listing 5.13: PHP code for client check in loadOverview function from EmployeeController.php

```
$employee = $this->get("auth")->check(_GROUP_EMPLOYEE);
```

5.4.3 Testing for Privilege Escalation - OTG-AUTHZ-003

	Online Banking
Observation	We could not detect any possibilities of privilege escalation.
Discovery	Controller functions are properly prepended by access right checks. Where the Users Id is used in code it is taken from the session and not from GET or POST parameters. There is no functionality accessible for the Client that allowes write access to the Clients data. It has been implemented in `clientController.php` and `employeeController.php` based on session values. Refer 5.26 for code related to client check and 5.27 for code related to employee check based on session value.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	We could not detect any possibilities of privilege escalation.
Discovery	Each controller function is prepended by an access right check, that checks for the current users static and contextual privileges. Where the Users Id is used in code it is taken from the session and not from GET or POST parameters. Refer 5.28 for code related to customer check and 5.29 for code related to employee check based on session value. Similar checks are implemented for all client, employee and administrator operations.
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

Comparison

Neither application is vulnerable in this regard.

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

	Online Banking
Observation	It was observed that Smart Card Simulator can be downloaded directly through the URL `http://<IP-address>Online Banking/SCS/SCS.jar` without being logged in to the application. Though nothing can be done without the pin and user's credentials, the simulator can be used as a test object to observe the TANs generated and create future attacks.
Discovery	On inspecting the code, it was found that no authentication was applied on the download link. • **Scenario 1 -** In case of client, all other actions point to `client.php` which in-turn refers to `clientController.php` where there is a check for client. Refer 5.14 and 5.15 for the related code excerpts. • **Scenario 2 -** In case of employee, All other actions point to `employee.php` which in-turn refers to `employeeController.php` where there is a check for employee. Refer 5.16 and 5.17 for the related code excerpts.
Likelihood	N/A
Impact	N/A
Recommendations	The simulator should not be allowed to be downloaded without authentication.
CVSS	N/A

	SecureBank
Observation	It was observed that Smart Card Simulator cannot be downloaded directly through the URL.
Discovery	Code inspection revealed that only clients who have chosen SCS as preferred method for TAN generation, during registration get the link to download the SCS. Refer 5.18 for related code.

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

Comparison

SecureBank is more secure than Online Banking since the Smart Card Simulator cannot be downloaded without authentication.

Listing 5.14: HTML code for download of SCS from client.php

```
<li <?php if ($_GET['action'] == 'account') { echo 'class="active"'; } ?>>
    <a href="client.php?action=account">My account</a>
</li>
<li <?php if ($_GET['action'] == 'history') { echo 'class="active"'; } ?>>
    <a href="client.php?action=history">History</a>
</li>
<li <?php if ($_GET['action'] == 'online') { echo 'class="active"'; } ?>>
    <a href="client.php?action=online">New online transfer</a>
</li>
<li <?php if ($_GET['action'] == 'file') { echo 'class="active"'; } ?>>
    <a href="client.php?action=file">Load file</a>
</li>
<li>
    <a href="SCS/SCS.jar">SCS</a>
</li>
```

Listing 5.15: PHP code to check for client from clientController.php

```
if ($sessionisemployee){
  die('Not client.');
}
```

Listing 5.16: HTML code for download of SCS from employee.php

```
<li <?php if ($_GET['action'] == 'accounts') { echo 'class="active"'; } ?>>
    <a href="employee.php?action=accounts">Accounts</a>
</li>
<li <?php if ($_GET['action'] == 'approvepayments') { echo 'class="active"'; } ?>>
    <a href="employee.php?action=approvepayments">Approve transfers</a>
</li>
<li <?php if ($_GET['action'] == 'approveregistrations') { echo 'class="active"'; }
```

```
?>>
    <a href="employee.php?action=approveregistrations">
    Approve registrations</a>
</li>
<li>
    <a href="SCS/SCS.jar">SCS</a>
</li>
```

Listing 5.17: PHP code to check for employee from employeeController.php

```
if ($sessionisemployee != 1){
        die('Not employee.');
}
```

Listing 5.18: HTML code for download of SCS from client_base_html.php

```
<?php if ((int)$currentUser->getTanMethod() === _TAN_METHOD_SCS) { ?>
    <li class="treeview">
        <a href="/downloadscs">
            <i class="fa fa-download"></i>
            <span>Download SCS</span>
        </a>
    </li>
<?php }?>
```

5.5 Session Management Testing

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

	Online Banking 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. Also, transmission is over an unecrypted HTTP channel.
Discovery	We used EditThisCookie extension of Chrome to look into the cookies present in the application. Steps are as follows: • Go to the login page of the application. Check the cookies with the extension, which shows that there is no cookie. • Login with valid credentials. Upon checking the cookies now, a cookie PHPSESSID can be seen with some value. • The cookie remains persistent throughout the application. If the application is not idle the cookie remains set, otherwise the user is logged out using cookie "expires" attribute. No other cookie is generated throughout the application. The cookie is set to HostOnly and Session; HttpOnly and Secure are 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. Also as the cookie is not set to Secure, data transmission is not encrypted and hence Man-In-The-Middle attack can be performed. Upon inspecting the code, it was found that though `session_set_cookie_params` is called with the correct flags i.e., true for `HttpOnly` and `Secure`, it does not work as the session name is not set using `session_name`.
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
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. The cookie attribute is also set to `Secure` as we are using HTTPS and hence data transmitted is encrypted.
Discovery	This was found by manually inspecting the code. Refer 5.19 for the related code.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is better than Online Banking, as it disallows session hijacking through the HttpOnly and Secure attributes set for the session cookie and also uses HTTPS for secure communication.

Listing 5.19: PHP code for setting cookie attribues SessionService.php

```
session_name($name . '_session');
session_set_cookie_params($limit, $path, $domain, $https, true);
session_start();
```

5.5.2 Testing for Cookies attributes - OTG-SESS-002

	Online Banking CVSS Score: 9.8	
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	Setting of cookie attributes is explained in section 5.5.1	
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. Secure attribute of the cookie should also be set to avoid easy 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
Observation	It is found that the cookies that are set upon user login; have the HttpOnly and Secureflag set.
Discovery	Setting of cookie attributes is explained in section 5.5.1
Likelihood	N/A

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

Comparison

Unline Online Banking, chances of stealing the cookie are limited in SecureBank, since the session cookie is set to HttpOnly and Secure, thus eliminating the possibility of client-side manipulation and reducing data compromise.

5.5.3 Testing for Session Fixation - OTG-SESS-003

	Online Banking 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	We used the Cookie extensions in Firefox & Chrome and EditThis-Cookie in Chrome for executing this attack. Steps are as follows. • Login with Administrator credentials into to the application on Chrome. • Now click on the Cookie extension of Chrome and observe that the PHPSESSID cookie is set to some value. Copy this value for future use. • Note that the HostOnly and Session checkboxes are enabled while Secure and HttpOnly are not. This tells us the session can be hijacked by client-side manipulation of the cookie. • Now we open the Login page in Firefox. Open the Cookie extension through Tool tab in Firefox and add the PHPSESSID cookie manually and set it to the previously copied value. • Open the link `http://<IP-address>/OnlineBanking/employee.php?action=accounts`. 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	Session cookie should be set to HttpOnly and Secure.	
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
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	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

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

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

	Online Banking	CVSS Score: 6.5
Observation	The session cookie does not have the `httponly` flag set, therefore it is possible to set/edit the cookie via JavaScript or other methods.	
Discovery	In `controller/loginController.php` a PHP session is started, if the user is logged in successfully. The function for setting the cookie parameters is commented.	
Likelihood	To read the cookies a hacker only needs to read out the HTTP headers. Executing the JavaScript code `document.cookie = "PHPSESSID =SESSION_COOKIE_VALUE"` in the browser while on the website sets the cookie.	
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
Observation	The session cookie has the `secure` and `httponly` flag.
Discovery	The file `Service/SessionService.php` uses the function `session_set_cookie_params` for setting parameters for the session cookie.
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 Online Banking since copying the session cookie gives a hacker access to the account, if the user is logged in. With SecureBank this is not possible.

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

	Online Banking	CVSS Score: 5.3
Observation	No CSRF tokens were used for HTML forms.	
Discovery	The files for rendering the HTML forms do not show any indication for CSRF tokens. For example checking the transfer for CSRF tokens, the files `view/online.php` and `controller/clientFunctions.php` were checked for any CSRF tokens, but none were found.	
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
Observation	For all HTML forms there is a hidden input field for a CSRF token. Furthermore, the CSRF token is saved in the PHP session.
Discovery	The files `Service/CSRFService.php` and `Helper/TemplatingFormExtension.php` generate CSRF tokens and include it in every HTML form. The CSRF token is unique for each PHP session.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank uses CSRF tokens, whereas Online Banking does not.

5.5.6 Testing for logout functionality - OTG-SESS-006

	Online Banking CVSS Score: 4.8
Observation	Logout functionality requirements: • **Testing for log out user interface:** A logout button is clearly visible at the top right corner of the app • **Testing for server-side session termination:** The Session is terminated on server side after Logout. The Sessionid remains unchanged. Refer to: 5.20. • **Testing for session timeout:** The Session is terminated by PHP after the standart value of 1440s inactivity.
Discovery	Test of the logout functionality requirements: • **Testing for server-side session termination:** After Logout the PHPSESSID cookie remains unchanged in the Chrome resource inspector. • **Testing for session timeout:** See OTG-SESS-007
Likelihood	The fact that the session is not cleared properly makes the application more vulnerable to session hijacking atacks.
Impact	The fact, that the Session is not terminated properly makes yields the possibility that session variables that have not been cleared properly can be read by the next person who logs in.
Recommendations	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
Observation	Logout functionality requirements: • **Testing for log out user interface:** The logout button can be reached after using the context menu in the top right corner. • **Testing for server-side session termination:** The Session is not terminated on server and client side but continued. It seems only the variable that stores the user is reseted. Refer to: 5.21 • **Testing for session timeout:** The Session is automatically terminated by PHP after the standart value of 1440s inactivity.
Discovery	Test of the logout functionality requirements: • **Testing for server-side session termination:** The Session is terminated on server side after Logout. The Sessionid changes. • **Testing for session timeout:** See OTG-SESS-007
Likelihood	N/A
Impact	N/A

Recommendations	N/A
CVSS	N/A

Comparison

Only Online Banking is vulnerable here.

Listing 5.20: PHP code for logout from logoutController.php

```
session_start();
session_destroy();
```

Listing 5.21: PHP code for logout from SessionService.php

```
/**
* Resets the Session completely
*/
ublic function reset() {
    $_SESSION = array();
    session_destroy();
    session_start();
    session_regenerate_id();
    $this->init();
```

5.5.7 Test Session Timeout - OTG-SESS-007

	Online Banking
Observation	The Session is terminated by PHP after the standard value of 1440s inactivity.
Discovery	In the file `auth.php` no explicit Session timeout is specified. `php.ini` specifies no timeout different from the standart.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	The Session is terminated by PHP after the standard value of 1440s inactivity.
Discovery	In the file `Service/SessionService.php` no explicit Session timeout is specified. `php.ini` specifies no timeout different from the standart.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications exhibit similar behaviour.

5.5.8 Testing for Session puzzling - OTG-SESS-008

	Online Banking
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. Refer section 5.5.3 for details about session hijacking.
Discovery	Upon looking up the code, it is seen the session variables are set in the file `loginController.php` which is invoked once during login.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	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. Refer section 5.5.3 for details about session hijacking.
Discovery	Upon looking up the code, it is seen the session variables are set in the file `SessionService.php` which is invoked once during login.
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.

5.6 Data Validation Testing

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

	Online Banking	CVSS Score: 0.0
Observation	It has been found that Reflected Cross Site Scripting is possible in the application in the `Make Payment` interface.	
Discovery	Manual inspection of code revealed usage of functions such as `htmlspecialchars`, `filter_var` and `preg_match` for input sanitization; leading to HTML tags being rendered as plain text. However, while rendering output, absence of sanitization leads to this issue. Usage of `echo $_POST['receipt'];` was found at multiple locations in code. Refer to 5.22 for the related code. This was confirmed by a black box test. See 5.2 for an example of this vulnerability.	
Likelihood	Likelihood is low as exploitation of this vulnerability requires some technical skills. Moreover, an attacker can only perform this attack if he/she has access to user credentials.	
Impact	Impact is none as exploitation of this vulnerability does not affect other users.	
Recommendations	It is recommended to properly sanitize output before generating web content for the user.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	None
	Availability Impact	None

	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	Manual inspection of code revealed usage of functions such as `htmlspecialchars`, `filter_var` and `preg_match` for input sanitization; leading to HTML tags being rendered as plain text.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is better than Online Banking as it is completely safe from XSS attacks.

Listing 5.22: PHP code for rendering content in Make Payment page from online.php

```
<div class="form-group">
    <input type="text" name="receipt" value="<?php
    echo $_POST['receipt']; ?>"
     class="form-control" id="receipt" placeholder="Receipt"
     required>
</div>
<div class="form-group">
    <input type="text" name="amount" value="<?php
    echo $_POST['amount']; ?>"
     class="form-control" id="amount" placeholder="Amount in euro"
     required aria-required="true" pattern="[0-9]+"
     title="Please, input the field with digits">
</div>
<div class="form-group">
    <textarea class="form-control black-field" name="purpose"
     class="form-control" id="purpose" placeholder="Purpose"
     rows="2" required aria-required="true"><?php
     echo $_POST['purpose']; ?></textarea>
</div>
<div class="form-group">
    <input type="text" name="trancode" value="<?php
```

```
␣␣␣␣␣echo␣$_POST['trancode'];␣?>"
        class="form-control" placeholder="TAN" id="TAN" required>
</div>
```

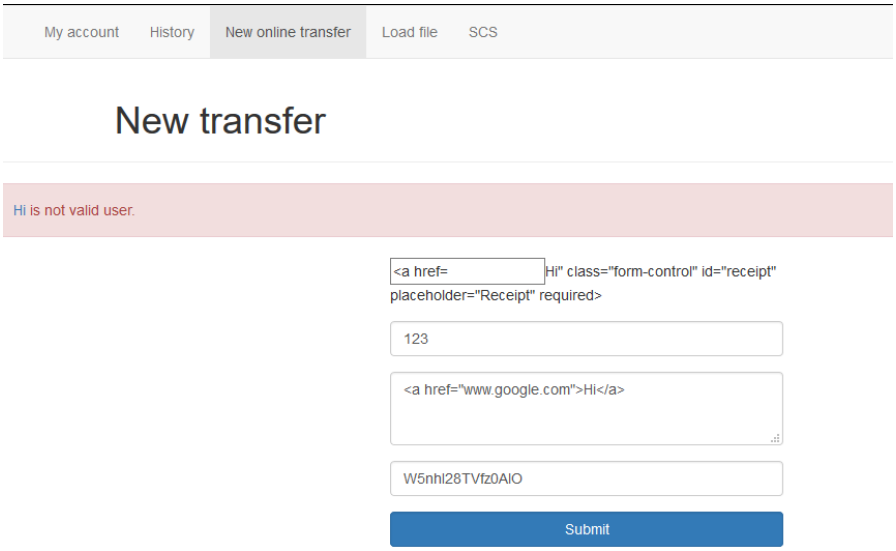


Figure 5.2: Reflected XSS in Make Payment page

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

	Online Banking
Observation	It was found that it is not possible perform stored XSS in the application. Simple HTML and Script tags were tried and but the attacks were unsuccessful.
Discovery	Manual inspection of code revealed usage of functions such as `htmlspecialchars`, `filter_var` and `preg_match` for input sanitization; leading to HTML tags being rendered as plain text.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	It was found that it is not possible to perform stored XSS in the application. Simple HTML and Script tags were tried and the attacks were unsuccessful.
Discovery	Manual inspection of code revealed usage of functions such as `htmlspecialchars`, `filter_var` and `preg_match` for input sanitization; leading to HTML tags being rendered as plain text.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications are secure with respect to Stored XSS attacks.

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

	Online Banking
Observation	It was observed that Verb Tampering could be done with HTTP requests but no critical vulnerability was exposed with it. Methods that were allowed : • GET • POST • HEAD • OPTIONS Methods that were rejected: • TRACE • CONNECT With HEAD requests, there were no response data shown. In case of `TRACE` and `CONNECT`, the requests were rejected because of Same Origin Security restriction.
Discovery	It was found that there is no reference to methods other than GET and POST, in the source code by scanning the code using `grep`.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	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 Online Banking.

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.

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

	Online Banking
Observation	In POST and GET requests only the last occurrence of a parameter is parsed. Viewing account related information does not depend on POST or GET requests.
Discovery	The application uses PHP/Apache and therefore only the last occurrence of a parameter is parsed.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	In POST and GET requests only the last occurrence of a parameter is parsed. Viewing account related information does not depend on POST or GET requests.
Discovery	The application uses PHP/Apache and therefore only the last occurrence of a parameter is parsed.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both applications are secure against HTTP parameter pollution.

5.6.5 Testing for SQL Injection - OTG-INPVAL-005

	Online Banking	CVSS Score: 7.5
Observation	Online Banking only uses prepared statements for some queries and does not validate/sanitize user input before using the values in the queries.	
Discovery	The file `DataAccess.php` contains functions for SQL queries. Not every function uses prepared statements. Example: The function `getAccountDetails` takes a user ID as parameter and the value is directly included into the query without being escaped. For example it is used in `controller/employeeController.php` in line 34. The parameter is the account number, which they directly retrieve from the GET parameter *account*.	
Likelihood	SQL injection is one of the most common vulnerabilities and therefore a hacker will try to exploit that.	
Impact	With SQL injection a hacker could gain sensitive information, drop the database or change values.	
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
Observation	Prepared statements were used for SQL queries. All user input is sanitized.
Discovery	The file `Model/Repository.php` is a base class for all SQL queries. Each of the queries use the `PDO::prepare` function. Since all repositories inherit the functions from the base repository and extra database functions (e.g. in `Model/TransactionRepository.php`) also use the `PDO::prepare` function, all queries on the database is escaped. Furthermore, `Helper/ValidationHelper.php` and `Helper/SanitizationHelper.php` implement functions for sanitizing and validation user input. Those functions are used when a form is rendered.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking is vulnerable against SQL injection since they do not use prepared statements in most cases. SecureBank does use prepared statements for all queries and sanitizes/validates all user input.

5.6.6 Testing for LDAP Injection - OTG-INPVAL-006

Both applications do not use LDAP

5.6.7 Testing for ORM Injection - OTG-INPVAL-007

Refer to OTG-INPVAL-005.

5.6.8 Testing for XML Injection - OTG-INPVAL-008

	Online Banking
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	Code was scanned using `grep` and no occurences of `.xml` were found
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	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	Code was scanned using `grep` and no occurences of `.xml` were found
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.

5.6.9 Testing for SSI Injection - OTG-INPVAL-009

	Online Banking
Observation	It has been observed that there are no `.shtml` files used in the application. But since it cannot be concluded that the server does not support SSI, the code `<pre><!-#echo var=''DATE_LOCAL'' -> </pre>` was inserted in the Registration form and registration was performed successfully. However, upon logging in as 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	Code was scanned using `grep` and no occurences of `.shtml` were found.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

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

5.6.10 Testing for XPath Injection - OTG-INPVAL-010

	Online Banking
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	Code was scanned using `grep` to match `xml` and none were found.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	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	Code was scanned using `grep` to match `xml` and none were found.
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.

5.6.11 IMAP/SMTP Injection - OTG-INPVAL-011

	Online Banking
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
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.

5.6.12 Testing for Code Injection - OTG-INPVAL-012

	Online Banking	CVSS Score: 6.5
Observation	The client and the employee site include a PHP file depending on the GET parameter *action*.	
Discovery	Searching for the keywords `include` and `include_once` in the PHP source code yielded that in `view/client.php` and `view/employee.php` a file is included via `include $action.'.php';`. Tracing the variable back to its initialization showed that the variable is set in `controller/clientController.php` and `controller/employeeController.php` respectively. It is set to the GET parameter *action*.	
Likelihood	A hacker could change the GET parameter for *action* and another file is included. Changing a GET parameter is not difficult.	
Impact	If the hacker is able to upload a PHP file to the `view` and has more information about the application, he could view that PHP file with the GET parameter and depending on the content of the PHP file, he/she could gain sensitive information.	
Recommendations	Do not load the pages depending on the GET parameter or check the GET parameter for validity.	
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
Observation	SecureBank does not use dynamic file inclusion via GET or POST requests.
Discovery	Searching for the keywords `include` and `include_once` in the PHP source code yielded that SecureBank does not include files dynamically depending on GET or POST requests.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking depends on the GET parameter *action* to display the respective page, therefore code injection is possible. SecureBank does not load files dynamically depending on GET or POST requests.

5.6.13 Testing for Command Injection - OTG-INPVAL-013

	Online Banking
Observation	We could not detect a Command Injection vulnerability.
Discovery	The filename of the uploaded file is renamed so there is no command injection possible. See 5.23.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	We could not detect a Command Injection vulnerability.
Discovery	The filename of the uploaded file is renamed so there is no command injection possible. See 5.24.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application is vulnerable to command injection.

Listing 5.23: PHP code for client check from clientController.php

```
$target_dir = "/home/samurai/Documents/parse/";
 ...
$target_file = $target_dir . "form.txt";
 ...
if (move_uploaded_file($_FILES["tranfile"]["tmp_name"], $target_file)) {
    $out = shell_exec("./parser/exec␣" . $sessionuserid . "␣" . $target_file);
    ...
}
```

Listing 5.24: PHP code for client check from clientController.php

```
$random_file_name = str_replace("/", "", $this->get("random")->getString(10));
$uploaded_file_name = $upload_dir.$random_file_name.".txt";
$shell_command = $_SERVER['DOCUMENT_ROOT'] .
    "/../textparser/textparser␣" .
    escapeshellarg($uploaded_file_name) . "␣" .
    escapeshellarg($customer_id) . "␣" .
    escapeshellarg($customer_name) . "␣" .
    escapeshellarg($customer_account_id) . "␣" .
    escapeshellarg($transaction_code) . "␣" .
    escapeshellarg(_MYSQL_HOST) . "␣" .
    escapeshellarg(_MYSQL_USER) . "␣" .
    escapeshellarg(_MYSQL_PASSWORD) . "␣" .
    escapeshellarg(_MYSQL_DATABASE);
exec($shell_command, $output, $return_var);
```

5.6.14 Testing for Buffer overflow - OTG-INPVAL-014

	Online Banking CVSS Score: 7.5
Observation	Integer overflow was found in the account initialization screen, due to no restriction on number of digits that can be entered. Moreover, in the C part buffer overflow exists.
Discovery	• From the Account details page, an employee can set the balance for a customer. There is no check on the upper limit of balance which can be set on both client and server side. • On entering `111111111` as the amount to be initialized, the balance was set to `99999999.99`. This confirms Integer overflow. • On entering `-200`, the balance was set to `0.00`. • On entering a string `ddd`, the result was a database error thrown to the user. • Upon inspecting the code, and found that there were no checks on the balance to be set and hence it accepted any value (string, negative values, large number etc.). On further observing the database, it was found that the `balance` column in the `user` table was of type `DECIMAL(10,2) UNSIGNED`. Hence Integer overflow takes place when there are more than 8 digits before the decimal. Also, negative value was set to zero due to the `unsigned` data type. In the C part it was found with reverse engineering that functions like `sprintf`, `strcpy` and `strcat` were used without checking for the lengths. Furthermore, using Valgrind it was revealed that only 53 of 98 allocated heaps were freed.
Likelihood	Likelihood is high, since anyone can easily enter large numbers without having any knowledge of the code and databases.
Impact	The impact is low since it does not crash the functionality or alter the flow of application. On the other hand, the impact for the C part is very high.

Recommendations	There should be appropriate validations on the balance to only accept positive numeric values within a limit. Check for the lengths in C before using the mentioned functions.	
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 CVSS Score: 7.5
Observation	There are checks for numeric positive values less than `50000` and hence there is no possibility of integer overflow. Similar checks are enforced on other relevant inputs as well. Moreover, in the C part buffer overflow exists.
Discovery	The code was manually inspected and a check was enforced using the constant `_MAX_ALLOWED_BALANCE_INITIALIZATION` that was set to `50000` in the file `parameters.php`. In the C part it was found with reverse engineering that the function `strcpy` was used without checking for the lengths. Furthermore, using Valgrind it was revealed that only 98 of 119 allocated heaps were freed.
Likelihood	Likelihood is middle because it is complex to reach a buffer overflow without knowing the exact code.
Impact	Impact is high because a hacker could gain control over the system.
Recommendations	Check for the lengths in C before using `strcpy`.

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

Comparison

SecureBank is more secure than Online Banking since there is no possibility of integer overflow, but both applications have the risk of buffer overflow in the C part.

5.6.15 Testing for incubated vulnerabilities - OTG-INPVAL-015

This has already been covered by section 5.6.2 and section 5.6.5.

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

	Online Banking
Observation	The application does not use the `Location` header with GET parameters.
Discovery	Searching for the keyword `location` in the PHP source code we filtered the use of the PHP function `header` for redirects. 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
Observation	The application does not use the `Location` header with GET parameters.
Discovery	Searching for the keyword `location` in the PHP source code we filtered the use of the PHP function `header` for redirects. 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.

5.7 Error Handling

5.7.1 Analysis of Error Codes - OTG-ERR-001

	Online Banking CVSS Score: 4.3
Observation	Error messages: • **Application Errors:** – The Application uses Custom Error messages for most of its error feedback – MySQL Errors are tried to be presented as `mysql_errno()`. Unfortunately this is a invalid syntax. – The Application presents direct shell output to the user. – There are many cases where Exeptions are not catched correctly
Discovery	On most errors the Application responds with `die(message);`. In most cases in `DataAccess.php` the Syntax used (`die(mysql_errno())`) is wrong and causes a php syntax error. Other errors are thrown as Exception with custom message. Many of these Exceptions are not caught later. For Example the Exception Thrown in `DataAccess.php` line 536 is never caught. Due to the current configuration of the Webserver these Exceptions do not end up on the users screen but are only posted to the log file. But a Webserver that is not configured to suppress these Exceptions and Error output would allow to expose a great ammount of Stack traces and Exception codes.
Likelihood	An Atacker can use the informations presented by the error messages to validate further atacks. This results in a higher likelihood for other atacks.
Impact	The Mysql error codes can be used by an atacker to directly verify the success of his actions and help him to correct errors.

Recommendations	Catch Exceptions and hide the direct errors and transalte them to more general custom error messages. Try to avoid incorrect php syntax in code parts that handle errors.	
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
Observation	No direct PHP Error messages are passed on to the user. See 5.25
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking has some severe flaws in Syntax and Programm Logic.

Listing 5.25: PHP code for error handling from RoutingService.php

```
/**
 * Runs the callback for the given request
 */
public function dispatch()
{
    try {

            ...

    } catch (\Exception $e) {
        if(_DEBUG === true) throw $e;
        if (!array_key_exists("503", $this->error_callbacks)) {
            $this->error_callbacks["503"] = function() {
                header($_SERVER['SERVER_PROTOCOL']." 503 Fail...");
                echo '503';
            };
        }
        call_user_func($this->error_callbacks["503"]);
        return;
    }
}
```

5.7.2 Analysis of Stack Traces - OTG-ERR-002

	Online Banking	CVSS Score: 0
Observation	The Application outputs a php exception with stack trace to the log (to the screen on insecure Webservers) in many error cases. Example is a wrong user input in the recover password functionality.	
Discovery	See OTG-ERR-001.	
Likelihood	An Atacker can use the informations presented by the error messages to validate further atacks. This results in a higher likelihood for other atacks.	
Impact	An atacker can use the application errors to estimate application internas like file and include structures.	
Recommendations	Hide the direct errors and translate 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

	SecureBank
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

Comparison

Both Apps seem to be clean on the first sight. But Online Banking has invisible deficits.

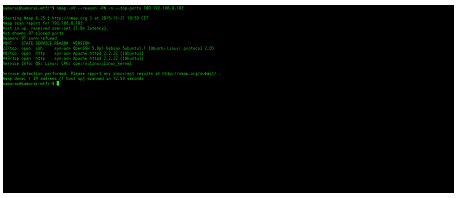
5.8 Cryptography

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

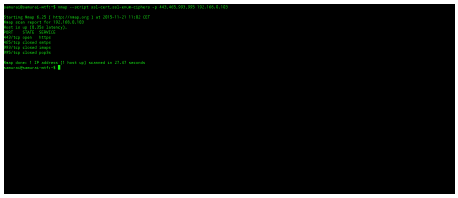
	Online Banking
Observation	It has been found that application works only on HTTP and does not support transmission over HTTPS. Neither does the application encrypt data used in requests. It is also observed that there are no ports having SSL services and hence no further testing could be done.
Discovery	Tests to determine transmission over HTTP/HTTPS have been described in section 5.3.1. We also performed tests to check for Basic Authentication over HTTP and SSL configuration in the ports. Following are the details. • **Test for HTTP Basic Authentication -** – Open the Login page in the browser. Also open Firebug in Firefox or Developer Tools in Chrome and navigate to the Network tab. – Enter credentials in the login form and click on "Submit". – Observe the request captured in the Network tab. The response does not contain the “WWW-Authenticate” header indicating that the server does not use Basic Authentication. • **Test for SSL services -** – Open the terminal and type `nmap -sV -reason -PN -n -top-ports 100 <IP-address>`. – To also check typical ports with SSL support, type `nmap -script ssl-cert,ssl-enum-ciphers -p 443,465,993,995 <IP-address>`. See Figure 5.3. 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
Observation	It has been found that the application works on HTTPS. It is also observed that there are ports having SSL services and HSTS has also been set.
Discovery	The SSL certificate can be found in `/etc/ssl/certs/server.crt`. All other details can be found in the configuration file **secure-bank.conf**.
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 5.3: Testing for ports with SSL configuration

Comparison

SecureBank is more secure than Online Banking since it works on HTTPS and SSL certificates are used.

5.8.2 Testing for Padding Oracle - OTG-CRYPST-002

	Online Banking
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 in the PDF through Email or in the SCS. 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
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 in the PDF through Email or in the SCS. 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

Comparison

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

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

Refer section 5.3.1

5.9 Business Logic Testing

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

	Online Banking CVSS Score: 8.6
Observation	It has been found that it is possible to enter valid data and cause the application to behave differently due to a deviation in the business logic. Two such vulnerabilities have been found and they are as follows. • In the Transaction page, it is possible to perform a transfer to own account. This transfer also reflects in the Transaction History but does not affect the Account Balance in any way. • In the Account Details page, employee can reset the balance of any customer to 0 repeatedly and thus prevent the customer from performing any transactions at all.
Discovery	By checking the code, there was no code related to checking transfer to self.
Likelihood	Likelihood is high. The attacker need not have any technical knowledge to perform this action.
Impact	It is possible to deny performing transfers by setting the balance to 0 infinite number of times.
Recommendations	• Transfer to self should be restricted. • Balance initialization should be restricted to a specific number of attempts.

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

	SecureBank
Observation	It has been found that there are appropriate checks to validate all data and hence no vulneranbility has been found in this regard.
Discovery	By manually inspecting the code, validations during transfer, balance initialization. Balance initialization is also a one-time process. After the emloyee has set the balance of an employee once, it cannot be done again.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

SecureBank is better than Online Banking as it does not allow circumvention of workflow in the application.

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

	Online Banking	CVSS Score: 6.5
Observation	Section 5.5.4 and section 5.5.5 have already shown that copying the session cookie from Online Banking 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 5.5.4 and section 5.5.5 describe the discovery of the vulnerability.	
Likelihood	Refer to sections 5.5.4 and 5.5.5.	
Impact	Refer to sections 5.5.4 and 5.5.5.	
Recommendations	Refer to sections 5.5.4 and 5.5.5.	
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
Observation	Concluding from section 5.5.4 and section 5.5.5 forging requests is not possible.
Discovery	Section 5.5.4 and section 5.5.5 describe the discovery of the results.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Using Online Banking forging requests is possible, whereas SecureBank is safe against request forgery.

5.9.3 Test Integrity Checks - OTG-BUSLOGIC-003

	Online Banking
Observation	There are no hidden input fields, which may depend on the current user role.
Discovery	Searching for the keyword `hidden` in the PHP source code we filtered out the 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
Observation	There are no hidden input fields, which may depend on the current user role.
Discovery	Searching for the keyword `hidden` in the PHP source code we filtered out the 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.

5.9.4 Test for Process Timing - OTG-BUSLOGIC-004

	Online Banking
Observation	The only siginificant timing abnormaly we could discover was on the customer approval functionality. We could not identify this as a thread.
Discovery	We checked page load times unsing the Google Chrome Developer Tools to determine time abnormalities. Most page load times where between 10ms and 40ms.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

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

Comparison

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

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

	Online Banking
Observation	The Transaction functionality could only be used 100 times. After that all TANs are used and no new tans are supplied.
Discovery	There is no functionality in the code that resends tans after all of them have been used.
Likelihood	The applications core functionality is no longer available after 100 transactions.
Impact	The applications core functionality is no longer available after 100 transactions.
Recommendations	Resend Tancodes after all of them have been used.
CVSS	N/A

	SecureBank
Observation	The Transaction functionality could only be used 100 times. After that all TANs are used and no new tans are supplied.
Discovery	There is no functionality in the code that resends tans after all of them have been used.
Likelihood	The applications core functionality is no longer available after 100 transactions.
Impact	The applications core functionality is no longer available after 100 transactions.
Recommendations	Resend Tancodes after all of them have been used.

CVSS	N/A

Comparison

Both apps do not take care about re-sending transaction codes.

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

	Online Banking
Observation	It is not possible to alter the work-flow of the application due to appropriate checks throughout the application. User privileges have been segregated according to the roles.
Discovery	By manually inspecting the code, it has been found that user type has been checked throughout the application before serving any page. Refer 5.26 for code related to client check and 5.27 for code related to employee check based on session value.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	It is not possible to alter the work-flow of the application due to appropriate checks throughout the application. User privileges has been segregated according to the roles.
Discovery	By manually inspecting the code, it has been found that user type has been checked throughout the application before serving any pages or performing any actions. Refer 5.28 for code related to customer check and 5.29 for code related to employee check based on session value. Similar checks are implemented for all client, employee and administrator operations.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both of the applications are secure since there is restriction on the altering the normal work-flow of the application.

Listing 5.26: PHP code for client check from clientController.php

```
if ($sessionisemployee){
    die('Not␣client.');
}
```

Listing 5.27: PHP code for employee check from employeeController.php

```
if ($sessionisemployee != 1){
    die('Not␣employee.');
}
```

Listing 5.28: PHP code for client check in loadOverview function from CustomerController.php

```
$customer = $this->get("auth")->check(_GROUP_USER);
```

Listing 5.29: PHP code for client check in loadOverview function from EmployeeController.php

```
$employee = $this->get("auth")->check(_GROUP_EMPLOYEE);
```

5.9.7 Test Defenses Against Application Mis-use - OTG-BUSLOGIC-007

	Online Banking CVSS Score: 6.5
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.
Discovery	• **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 5.3.7. • **Mis-use in performing Transactions -** – Login as a Customer and click on New online transfer at the top. – Fill the form with all the details and click on the Submit button OR use the Load File feature to perform a transaction. In both cases, the action can be replicated multiple times even with incorrect details. The Firefox extension FormFuzzer, Fuzz feature of ZAProxy or a similar tool can be used for filling the forms. • These attacks are not monitored which was observed while inspecting the code and finding no mechanism implemented for storing error logs. There is also no lockout mechanism implemented to prevent the user from further attacking the system.

Likelihood	This vulnerability does not require any technical skills. Logging into the web application through Brute-force methods is not easy since there is a policy on strong passwords but without a lockout mechanism in-place, it can also be exploited. 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. Thus the impact of such vulnerability is high.	
Recommendations	• The application should restrict or lock out the user after he exceeds a certain number of the failed attempts while performing any operation. • Logs of suspected actions should be maintained in database/file so as to monitor attempts for attacks.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality Impact	Low
	Integrity Impact	Low
	Availability Impact	None

	SecureBank
Observation	It has been found that there is no error log maintained for unsuccessful operations. In the absence of such monitoring, the application can be attacked by the user without being noticed. However, there is a lockout mechanism on failed login.
Discovery	By manually inspecting the code, the lockout mechanism was found in `DBAuthProvider.php` that locks the account for 60 minutes.
Likelihood	N/A
Impact	N/A
Recommendations	Logs of suspected actions should be maintained in database/file so as to monitor attempts for attacks.
CVSS	N/A

Comparison

SecureBank is more secure than Online Banking since there is lockout mechanism implemented the entry point of the application which restricts further attacks.

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

Refer Section 5.1.1

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

	Online Banking	CVSS Score: 6.1
Observation	Only text files (upto 500 bytes in size) can be uploaded through `Load Online` page.Even if files of other types are somehow uploaded, they cannot be used to exploit the application.	
Discovery	No tools were used to discover this vulnerability and manual testing and code review were done. We tried to upload different types of files but only text files were accepted. The uploaded file is moved to a text file `form.txt` and then read for parsing. Therefore command injection through file name of uploaded file is not possible. Since the contents of the file are moved to the same file each time, an attacker having valid login credentials can keep uploading a file and thus causing Denial of service to every other user of the application. Refer 5.30 .	
Likelihood	Likelihood is high since there is no need of technical knowledge for such attacks as the attacker just needs to upload a text file continuously.	
Impact	Impact is high since it can cause Denial of Service to other users of the application as content of the uploaded file is changed continuously by the attacker.	
Recommendations	The uploaded files should be maintained with unique names to avoid conflicts during concurrent events.	
CVSS	Attack Vector	Network
	Attack Complexity	Low
	Privileges Required	High
	User Interaction	Required
	Scope	Unchanged
	Confidentiality Impact	None
	Integrity Impact	High
	Availability Impact	High

	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(upto 1MB in size). Refer 5.31.
Discovery	The uploaded files are moved to text files having unique random name and hence do not cause issues even in case of concurrent events.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Both the applications are secure since they restrict the uploading to text files only and having measures against command injection as well. However, Online Banking has a serious vulnerability prone to DOS attacks, thus making it vulnerable.

Listing 5.30: PHP code for upload file from clientFunctions.php (Online Banking)

```
$target_dir = "/home/samurai/Documents/parse/";
$target_file = $target_dir . "form.txt";
$uploadOk = 1;
$file_type = pathinfo($_FILES["tranfile"]["name"],PATHINFO_EXTENSION);

if ($_FILES["tranfile"]["size"] > 500){
        $uploadOk = 0;
        throw new Exception("File␣too␣large!");
}

if($file_type != "txt"){
        $uploadOk = 0;
        throw new Exception("Only␣txt␣files␣are␣accepted!");
}


if ($uploadOk != 0){
        if (move_uploaded_file($_FILES["tranfile"]["tmp_name"], $target_file)){
          $out = shell_exec("./parser/exec␣" . $sessionuserid . "␣" . $target_file);
                if($out == "Operation␣succesful"){
```

```
                        return $out;
                }
                else{
                        throw new Exception($out);
                }
        }
        else{
                throw new Exception("Could␣not␣upload␣file");
        }
}
```

Listing 5.31: PHP code for upload file from TransactionController.php (SecureBank)

```
private function processBatchTransfer($request, $helper, $helper2, $customer) {
    $requestVar = $request->getData('make_transfer_via_file_upload');
$transaction_code = $requestVar['transaction_code'];

$upload_dir = $_SERVER['DOCUMENT_ROOT'].'/tmp/';
$file = $request->getFile('make_transfer_via_file_upload', 'file');
if ($file['type'] != "text/plain") {
    $this->get("flash_bag")->add(_OPERATION_FAILURE, "The␣uploaded␣file
␣␣␣␣␣must␣be␣a␣plain␣text␣file", "error");
    $this->get("routing")->redirect("make_transfer_get", array("form" =>
     $helper, "form2" => $helper2));
    return;
} else if ($file['error'] == 2) {
    $this->get("flash_bag")->add(_OPERATION_FAILURE, "The␣uploaded␣file
␣␣␣␣size␣exceeds␣the␣maximum␣of␣1␣MB", "error");
    $this->get("routing")->redirect("make_transfer_get", array("form" =>
    $helper, "form2" => $helper2));
    return;
}

$customer_id = $customer->getId();
$customer_name = $customer->getFirstName()
                    . "␣" . $customer->getLastName();

$random_file_name = str_replace("/", "", $this->get("random")->getString(10));
$uploaded_file_name = $upload_dir.$random_file_name.".txt";

// rename uploaded file name if already exists
$i = 1;
do {
    if ($i == 1) {
        $pos = strrpos($uploaded_file_name, ".txt");
        $uploaded_file_name = substr_replace($uploaded_file_name, "_".$i,
                                                $pos, 0);
    } else {
```

```
            $pos = strrpos($uploaded_file_name, "_".($i−1).".txt");
            $uploaded_file_name = substr_replace($uploaded_file_name, "_".$i, $pos,
             strlen((string)$i)+1);
        }
        $i++;
} while (file_exists($uploaded_file_name));
......
}
```

5.10 Client Side Testing

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

	Online Banking
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	We scanned the code using `grep` for `document.location.href` and `window.location` in Javascript but none were found. We also confirmed by manually appending strings such as `#<script>alert('hi')</script>` to URLs. After refreshing this page with this value, no changes were observed.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	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	We scanned the code using `grep` for `document.location.href` and `window.location` in Javascript but none were found.
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.

5.10.2 Testing for JavaScript Execution - OTG-CLIENT-002

Refer sections 5.6.1 and 5.6.2.

5.10.3 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
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

Both applications exhibit similar behaviour.

5.10.4 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
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

Both applications exhibit similar behaviour.

5.10.5 Testing for CSS Injection - OTG-CLIENT-005

	Online Banking
Observation	It has been observed that CSS injections cannot be performed as user inputs are sanitized. Hence we cannot inject html tags which could be used to execute scripts indirectly.
Discovery	Manual inspection of code revealed usage of functions such as `htmlspecialchars`, `filter_var` and `preg_match` for input sanitization.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
Observation	It has been observed that CSS injections cannot be performed as user inputs are sanitized. Hence we cannot inject html tags which could be used to execute scripts indirectly.
Discovery	Manual inspection of code revealed usage of functions such as `htmlspecialchars`, `filter_var` and `preg_match` for input sanitization.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Neither application is vulnerable to this attack since all inputs are properly sanitized.

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

	Online Banking
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	Upon scanning the code with `grep`, the following injection points were found: `<a>`, `<link>` and `<script>`. However, these tags pointed to static resources and are hence not based on user-input. The URL parameters visible in the Account (`http://<IP-address>/Online Banking/employee.php?action=account&account=xxx`) and History (`http://<IP-address>/Online Banking/client.php ?action=history&account=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
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	Upon scanning the code with `grep`, the following injection points were found: `<a>`, `<link>` and `<script>`. However, these tags pointed to static resources and are hence not based on user-input. The URL parameter visible in the Customer Details (`http://<IP-address>/customer_details/xxx`) page is only being used in queries for retrieval of data from the database and not as targets of any resources. In the file `routes.php`, no other URL was found to contain URL parameters.

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.

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

	Online Banking
Observation	It was found that Cross Origin Resource Sharing is not possible since the `Access-Control-Allow-Origin` header was not set in the requests and hence the application does not support cross origin requests.
Discovery	Though a Javascript code snippet was written to test for CORS support, we were not able to simulate cross site requests directly. Refer 5.32 for the code snippet. 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 5.4. The header should have been set to * or some domain in order to serve cross domain requests.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

	SecureBank
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 Online Banking.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

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

Listing 5.32: Javascript code for testing CORS support

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

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

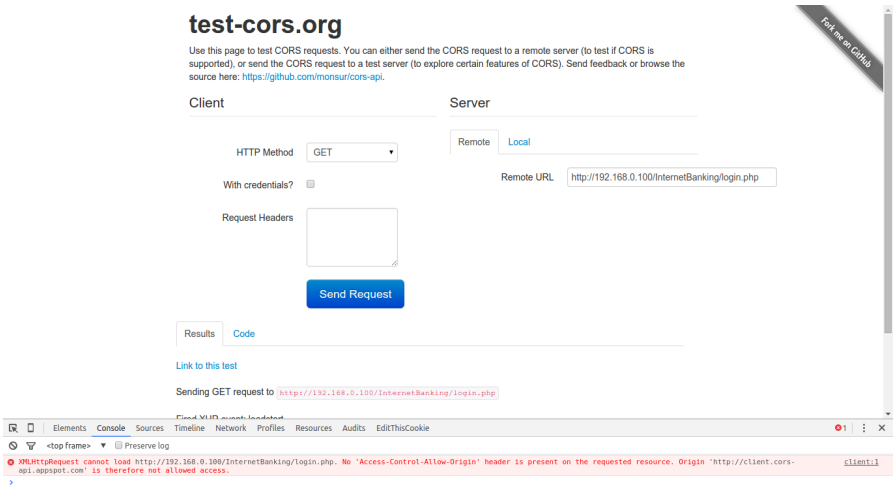


Figure 5.4: Test for Cross Origin Resource Sharing

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

	Online Banking
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
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.

5.10.9 Testing for Clickjacking - OTG-CLIENT-009

	Online Banking	CVSS Score: 5.3
Observation	The HTTP header option `X-Frame-Options` is not set.	
Discovery	No `.htaccess` file was found, which usually is used to set the HTTP header option `X-Frame-Options`.	
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
Observation	SecureBank denies clickjacking.
Discovery	In the `src/.htaccess` file there is the line with `Header set X-Frame-Options DENY`, which denies embedding the site in an iframe.
Likelihood	N/A
Impact	N/A
Recommendations	N/A
CVSS	N/A

Comparison

Online Banking could be loaded into an iframe, whereas SecureBank denies that option.

5.10.10 Testing WebSockets - OTG-CLIENT-010

Both applications do not use any WebSockets and therefore are not vulnerable regarding WebSockets.

5.10.11 Test Web Messaging - OTG-CLIENT-011

None of the apps use Web Messaging functions

5.10.12 Test Local Storage - OTG-CLIENT-012

None of the apps use Local Storage

5.11 Application Testing

Upon extensive testing of the application , numerous flaws and bugs were detected. While some of them do not cause any harm, they still need to be corrected. Few others could have serious effects.

5.11.1 Online Banking

- **Deviations from Application Requirements**
  - There is no concept of Account Number anywhere in the application. All references to accounts are via User-names. Being a Bank application, this is a foremost flaw.
  - There is no provision for the user to choose between PDF and SCS for generation of TANs. As a result, every users is able to use both methods to get the TANs.
  - The password for the encrypted PDF containing TANs is same as the user password for the bank application. If the user credentials are compromised, the user can also open the PDF and perform transactions.
  - The PIN for the SCS is in the PDF file containing TANs. So again, an attacker who has the user's credentials, can utilize the 100 TANs in the PDF and use SCS after that to continue performing transfers.
  - The Batch transfer functionality does not work as expected. Upon upload of any file, the error message `Negative transactions not allowed` is displayed. So it is not at all possible to perform batch transactions.
- **Usability Flaws**
  - In the Account details page, the label for User-name is incorrectly displayed as E-mail. The E-mail is also labelled as E-mail.
  - It is possible to set balance for Employees as well. However, neither do the employees have accounts nor can they perform any transfers and hence this is misleading.
  - When an employee rejects a transfer, the success message is shown as `Payment approved successfully`. This is confusing and requires the employee to cross-check in the transaction history of the customer.
  - The PDF that can be downloaded from the Transaction History page is not well-formed. The TAN numbers overlap and are not clearly readable.

  - In the SCS, TANs are generated even without entering any details like PIN, Amount and Target. Since it is possible to download the SCS from the website, without having an account, this may be used to analyze the algorithm used for TAN generation.
  - In the SCS, once values are entered, there is no way to clear or reset the values. Even after choosing a file, there is no way to clear the file.

- **Functionality Issues**
  - While approving/rejecting registrations, the first registration is always approved or rejected irrespective of which row the action is performed on. Same is the case with approval/rejection of payments. On inspecting code, we found that each row has a form with the same id `f_approve` and upon click of the buttons, the form is fetched using `document.getElementById(f_approve)`. Since Javascript fetches the first element that matches the id, the form in the first row is always returned.
  - After rejecting a registration, it can be approved again. This can either be done via tools such as Advanced Rest Client, ZAP etc. or can also happen when two employees try to perform opposing actions on the same registration. Same is the case with payments.

6 Appendix

6.1 Java Code for the Smart Card Simulator

Listing 6.1: Java code of the Smart Card Simulator

```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.PrintStream;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class TanGenerator {
	static final JTextField fileField = new JTextField();

	private static int normalize(int paramInt) {
		if (paramInt < 10) {
			return paramInt;
		}
		return paramInt + 8;
	}

	public static String compute(String paramString1, String paramString2,
	String paramString3) throws NoSuchAlgorithmException {
		Random localRandom = new Random();
	MessageDigest localMessageDigest = MessageDigest.getInstance("MD5");
```

```
    byte[] arrayOfByte1 = (paramString1 + paramString2
    + paramString3).getBytes();
    byte[] arrayOfByte2 = new byte[arrayOfByte1.length + 2];

    arrayOfByte2[0] = ((byte) (48 + normalize(localRandom.nextInt(35))));
    arrayOfByte2[1] = ((byte) (48 + normalize(localRandom.nextInt(35))));

    for (int i = 0; i < arrayOfByte1.length; i++) {
            arrayOfByte2[(i + 2)] = arrayOfByte1[i];
    }
    byte[] arrayOfByte3 = localMessageDigest.digest(arrayOfByte2);

    StringBuilder localStringBuilder = new StringBuilder();
    localStringBuilder.append((char) arrayOfByte2[0]);
    localStringBuilder.append((char) arrayOfByte2[1]);

    for (int j = 0; j < 13; j++) {
            int k = arrayOfByte3[j] & 0xFF;
            int m = normalize(k % 35);
            localStringBuilder.append((char) (48 + m));
    }
    return localStringBuilder.toString();
  }

  public static void main(String[] paramArrayOfString)
  throws NoSuchAlgorithmException {
    System.out.println(compute("123456", "admin", "100"));

    final JFrame localJFrame = new JFrame("SCS");
    localJFrame.setLayout(new BorderLayout());

    localJFrame.setDefaultCloseOperation(3);

    final JFileChooser localJFileChooser = new JFileChooser();

    JPanel localJPanel1 = new JPanel(new GridLayout(1, 2));
    localJFrame.add(localJPanel1, "North");

    fileField.setEditable(false);
    fileField.setColumns(16);

    JPanel localJPanel2 = new JPanel(new FlowLayout(0));
    localJPanel2.add(fileField);
    localJPanel1.add(localJPanel2);

    JButton localJButton1 = new JButton("Choose␣file");
    localJButton1.addActionListener(new ActionListener() {
```

```
public void actionPerformed(ActionEvent event) {
  if (localJFileChooser.showOpenDialog(localJFrame) == 0) {
      File localFile = localJFileChooser.getSelectedFile();
      TanGenerator.fileField.setText(localFile.getPath());
  }
}
  });
  localJPanel1.add(localJButton1);

  JPanel localJPanel3 = new JPanel(new GridLayout(5, 1));
  JPanel localJPanel4 = new JPanel(new GridLayout(5, 1));
  localJFrame.add(localJPanel3, "West");
  localJFrame.add(localJPanel4, "Center");

  JTextField localJTextField1 = new JTextField();
  localJTextField1.setColumns(16);
  JLabel localJLabel1 = new JLabel("PIN:", 4);
  localJLabel1.setLabelFor(localJTextField1);

  localJPanel3.add(localJLabel1);
  JPanel localJPanel5 = new JPanel(new FlowLayout(0));

  localJPanel5.add(localJTextField1);
  localJPanel4.add(localJPanel5);

  final JTextField localJTextField2 = new JTextField();
  localJTextField2.setColumns(16);
  JLabel localJLabel2 = new JLabel("Target:", 4);
  localJLabel2.setLabelFor(localJTextField2);

  localJPanel3.add(localJLabel2);

  JPanel localJPanel6 = new JPanel(new FlowLayout(0));
  localJPanel6.add(localJTextField2);
  localJPanel4.add(localJPanel6);

  final JTextField localJTextField3 = new JTextField();
  localJTextField3.setColumns(16);
  JLabel localJLabel3 = new JLabel("Amount:", 4);
  localJLabel3.setLabelFor(localJTextField3);
  localJPanel3.add(localJLabel3);
  JPanel localJPanel7 = new JPanel(new FlowLayout(0));
  localJPanel7.add(localJTextField3);
  localJPanel4.add(localJPanel7);

  final JTextField localJTextField4 = new JTextField();
  localJTextField4.setColumns(16);
  localJTextField4.setEditable(false);
```

```
            JLabel localJLabel4 = new JLabel("Generated␣TAN:", 4);
            localJLabel4.setLabelFor(localJTextField4);
            localJPanel3.add(localJLabel4);
            JPanel localJPanel8 = new JPanel(new FlowLayout(0));
            localJPanel8.add(localJTextField4);
            localJPanel4.add(localJPanel8);

            JButton localJButton2 = new JButton("Generate␣TAN");

            localJButton2.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent event) {
                      try {
                              String str1;
                        if (TanGenerator.fileField.getText().isEmpty()) {
                              str1 = TanGenerator.compute(localJTextField1.getText(),
                              localJTextField2.getText(),
                              localJTextField3.getText());
                      } else {
                        byte[] arrayOfByte = Files
                        .readAllBytes(Paths.get(TanGenerator.fileField.getText(),
                        new String[0]));
                        String str2 = new String(arrayOfByte,
                        StandardCharsets.UTF_8);
                        str1 = TanGenerator.compute(localJTextField1.
                        getText(), str2, "");
                      }
                      localJTextField4.setText(str1);
                      } catch (Exception localException) {
                              localException.printStackTrace();
                      }
                    }
            });

            JPanel localJPanel9 = new JPanel();
            localJPanel9.add(localJButton2);
            localJFrame.add(localJPanel9, "South");

            localJFrame.setLocationRelativeTo(null);
            localJFrame.pack();
            localJFrame.setVisible(true);
          }
}
```