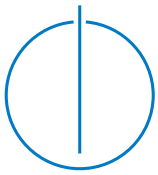
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Report

Black Box Testing Report

Alexis Engelke, Johannes Fischer



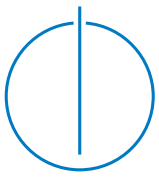
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Report

Black Box Testing Report

Author:	Alexis Engelke, Johannes Fischer
Team:	9
Lecture:	Secure Coding, Phase 2
Submission Date:	November 24, 2015



Executive Summary

1 Online Banking

We found several vulnerabilities, which allow an attacker to cause high damage to the bank behind the Online Banking web application. The system, as is, should not be used in production!

It is possible to login as any user without knowing the valid password. The attacker can either brute-force the password, or get into the account using more advanced techniques (SQL injection). This allows an attacker to take over an employee of the bank, get a list of all registered users and approve newly registered users as well as transaction above 10,000 units of currency. Furthermore, it is then possible to change the account balances at will. It is also possible to perform transactions without valid transaction codes using transaction batch files. Beside this, it is also possible to insert scripts into the frontend of an employee using the username field or the transaction description. This allows an attacker to take over the session of an employee.

If an attacker performs a man-in-the-middle attack, he will be able to intercept the whole traffic, as the server only allows communication via unencrypted channels. This allows an attacker to take over the session of a user. However, the attacker has to be more advanced to perform such an attack.

Finally, there is also an issue in the business logics of the application: it is possible to transfer money from other accounts to the own account by putting a negative amount into a transaction in a batch file, and therefore “steal" money from other customers.

2 SecureBank

We found some issues in the SecureBank’s online banking application, which allow an attacker to cause damage to the bank. The system, as is, should not be used in production.

For an attacker, it is possible to brute-force the password of a user, as there is no lock-out mechanism. It is furthermore possible to insert scripts into the frontend of an employee using the transaction description field. While this requires that the attacker is a client of the bank, it allows the attacker to take over the session of an employee.

If an attacker performs a man-in-the-middle attack, he will be able to intercept the whole traffic, as the server only allows communication via unencrypted channels. This allows an attacker to take over the session of a user. However, the attacker has to be more advanced to perform such an attack.

3 Comparison

In summary, we found that the web application of the SecureBank has less and also less critical vulnerabilities than the Online Banking web application.

Contents

Executive Summary **ii**
1 Online Banking ii
2 SecureBank ii
3 Comparison iii

1 Time Tracking **1**

2 Vulnerabiliteis Overview **3**
2.1 Online Banking 3
2.1.1 Stored XSS in Registration and Transaction Description 3
2.1.2 Missing check for amount in transactions from batch file 3
2.1.3 SQL injection in login form 3
2.1.4 SQL injection in transaction batch file 4
2.1.5 Missing lock out mechanism 4
2.1.6 Missing transport layer security 4
2.2 SecureBank 5
2.2.1 Stored XSS in transaction description 5
2.2.2 Missing lock out mechanism 5
2.2.3 Missing transport layer security 5

3 Tools **6**
3.1 Zed Attack Proxy (ZAP) 6
3.2 SQLmap 6
3.3 W3AF 6
3.4 SQL Inject Me 7
3.5 Burp Suite 7
3.6 Firebug / Chromium Developer Tools 7

4 Detailed Report **8**
4.1 Configuration and Deploy Management Testing 9
4.1.1 Test File Extensions Handling for Sensitive Information 9
4.1.2 Test HTTP Methods 15

4.1.3 Test HTTP Strict Transport Security . . . 16
4.1.4 Test RIA cross domain policy . . . 17
4.2 Identity Management Testing . . . 18
4.2.1 Test Role Definitions . . . 18
4.2.2 Test User Registration Process . . . 20
4.2.3 Test Account Provisioning Process . . . 22
4.2.4 Testing for Account Enumeration and Guessable User Account . . . 23
4.2.5 Testing for Weak or unenforced username policy . . . 24
4.3 Authentcation Testing . . . 25
4.3.1 Testing for Credentials Transported over Encrypted Channel . . . 25
4.3.2 Testing for default credentials . . . 25
4.3.3 Testing for Weak lock out mechanism . . . 26
4.3.4 Testing for bypassing authentication schema . . . 28
4.3.5 Testing for Vulnerable Remember Password . . . 30
4.3.6 Testing for Browser Cache Weakness . . . 31
4.3.7 Testing for Weak password policy . . . 32
4.3.8 Testing for Weak security question/answer . . . 33
4.3.9 Testing for Weak password change or reset functionalities . . . 33
4.3.10 Testing for Weaker authentication in alternative channel . . . 33
4.4 Authorization Testing . . . 34
4.4.1 Testing Directory traversal/file include . . . 34
4.4.2 Testing for bypassing authorization schema . . . 35
4.4.3 Testing for Privilege Escalation . . . 36
4.4.4 Testing for Insecure Direct Object References . . . 37
4.5 Session Management Testing . . . 38
4.5.1 Testing for Bypassing Session Management Schema . . . 38
4.5.2 Testing for Cookies attributes . . . 40
4.5.3 Testing for Session Fixation . . . 41
4.5.4 Testing for Exposed Session Variables . . . 43
4.5.5 Testing for Cross Site Request Forgery . . . 45
4.5.6 Testing for logout functionality . . . 47
4.5.7 Test Session Timeout . . . 48
4.5.8 Testing for Session puzzling . . . 49
4.6 Data Validation Testing . . . 50
4.6.1 Testing for Reflected Cross Site Scripting . . . 50
4.6.2 Testing for Stored Cross Site Scripting . . . 52
4.6.3 Testing for HTTP Verb Tampering . . . 54
4.6.4 Testing for HTTP Parameter pollution . . . 55
4.6.5 Testing for SQL injection . . . 56

4.6.6 Testing for LDAP Injection . . . 59
4.6.7 Testing for ORM Injection . . . 59
4.6.8 Testing for XML Injection . . . 59
4.6.9 Testing for SSI Injection . . . 59
4.6.10 Testing for XPath Injection . . . 59
4.6.11 IMAP/SMTP Injection . . . 59
4.6.12 Testing for Code Injection . . . 60
4.6.13 Testing for Local/Remote File Inclusion . . . 61
4.6.14 Testing for Command Injection . . . 62
4.6.15 Testing for Buffer overflow . . . 63
4.6.16 Testing for incubated vulnerabilities . . . 64
4.6.17 Testing for HTTP Splitting/Smuggling . . . 65
4.7 Error Handling . . . 66
4.7.1 Analysis of Error Codes . . . 66
4.7.2 Testing for Stack Traces . . . 68
4.8 Testing for weak Cryptography . . . 69
4.8.1 Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection . . . 69
4.8.2 Testing for Padding Oracle . . . 71
4.8.3 Testing for Sensitive information sent via unencrypted channels 71
4.9 Business Logic Testing . . . 72
4.9.1 Test Business Logic Data Validation . . . 72
4.9.2 Test Ability to Forge Requests . . . 74
4.9.3 Test Integrity Checks . . . 75
4.9.4 Test for Process Timing . . . 76
4.9.5 Test Number of Times a Function Can be Used Limits . . . 77
4.9.6 Testing for the Circumvention of Work Flows . . . 78
4.9.7 Test Defenses Against Application Mis-use . . . 79
4.9.8 Test Upload of Unexpected File Types . . . 80
4.9.9 Test Upload of Malicious Files . . . 81
4.10 Client Side Testing . . . 82

Acronyms **83**

1 Time Tracking

If a task is prefixed with (o), it refers to the Online Banking web application, if a task is prefixed with (s), the task refers only to the SecureBank web application.

Table 1.1: Time Tracking Table

Name	Task	Time
Alexis Engelke	Setting up LaTeX template	1
Alexis Engelke	(o) Analyzing XSS vulnerabilities using ZAP	2
Alexis Engelke	(o) Analyzing SQL injection vulnerabilities in the web interface using SQLmap	1.5
Alexis Engelke	(o) Analyzing SQL injection vulnerabilities in the file upload	2
Alexis Engelke	(o) Exploiting the TAN verification in the file upload	2
Alexis Engelke	(o) Documenting SQL injection	1
Alexis Engelke	(s) Searching for XSS vulnerabilities	1
Alexis Engelke	(s) Testing and Documenting SQL injection	2
Alexis Engelke	Testing and Documenting Configuration and Deploy Management Testing	2
Alexis Engelke	Testing and Documenting Identity Management Testing	1
Alexis Engelke	Testing and Documenting Authentication Testing	2
Alexis Engelke	Testing and Documenting Authorization Testing	2
Alexis Engelke	Presentation and Screencasts	2
Foo	Fixing all issues	10

2 Vulnerabiliteis Overview

Through our testing, we identified the following vulnerabilities as the most critical for the Online Banking application and the SecureBank:

2.1 Online Banking

2.1.1 Stored XSS in Registration and Transaction Description

- Likelihood: high
- Implication: high
- Risk: high

With stored cross site scripting attacks it is possible to inject JavaScript code, which is run whenever an employee logs in and opens the list of unapproved accounts or transactions. It is also possible to inject script from other sites.

2.1.2 Missing check for amount in transactions from batch file

- Likelihood: medium
- Implication: high
- Risk: high

It is possible to get money from another client of the bank by filling in a negative number in the amount field of a transaction batch file. Therefore, one client can generate an infinite amount of money, while reducing the amount of money of other clients.

2.1.3 SQL injection in login form

- Likelihood: high
- Implication: high

- Risk: high

The application is vulnerable to SQL injections in the login form. It is possible to login as a user while only knowing its name.

2.1.4 SQL injection in transaction batch file

- Likelihood: medium
- Implication: high
- Risk: high

The application is vulnerable to SQL injections in the transaction batch files. Therefore, it is possible to perform transactions while using any unused TAN in the system, which is not known to the attacker and might come from another client.

2.1.5 Missing lock out mechanism

- Likelihood: high
- Implication: medium
- Risk: medium

The application does not provide a lock out mechanism, allowing an attacker to brute-force for the password of a known username.

2.1.6 Missing transport layer security

- Likelihood: high
- Implication: high
- Risk: high

The application does not communicate over a TLS/SSL encrypted channel. This allows Man-in-the-Middle attacks, allowing to take over the session of a user.

2.2 SecureBank

2.2.1 Stored XSS in transaction description

- Likelihood: high
- Implication: high
- Risk: high

With stored cross site scripting attacks it is possible to inject JavaScript code, which is run whenever an employee logs in and opens the list of unapproved accounts or transactions. It is also possible to inject script from other sites.

2.2.2 Missing lock out mechanism

- Likelihood: high
- Implication: medium
- Risk: medium

The application does not provide a lock out mechanism, allowing an attacker to brute-force for the password of a known username.

2.2.3 Missing transport layer security

- Likelihood: high
- Implication: high
- Risk: high

The application does not communicate over a TLS/SSL encrypted channel. This allows Man-in-the-Middle attacks, allowing to take over the session of a user.

3 Tools

3.1 Zed Attack Proxy (ZAP)

Using the Zed Attack Proxy (ZAP), we were able to reveal significant parts of the directory structure in both web applications. In the *Online Banking* web application, we found a stored XSS vulnerability in the registration and the transaction description as well as a SQL injection vulnerability in the login form using the fuzzer. We were also be able to find a buffer overflow vulnerability for the transaction description in the transaction batch files. In the *SecureBank* web application, we were unable to find further SQL injection or XSS vulnerabilities.

3.2 SQLmap

Using SQLmap, we found the SQL injection vulnerability in the login form of the *Online Banking* application, which we found using ZAP earlier. SQLmap did not reveal further SQL injection possibilities in any of the two applications.

3.3 W3AF

With W3AF, we made a full audit, including file_upload(looks for uploadable files), eval(insecure eval() usage), un_ssl(secure content via http), os_commanding, lfi(local file inclusion), rfi, sqli(injection), preg_replace(insecure preg_replace() in PHP), mx_injection, generic, format_string, websocket_hijacking, shell_shock, ldapi, buffer_overflow, redos(DOS using slow regex), global_redirect(any redirecting scripts), xpath, cors_origin(consistency of HTTP origin header and sender), htaccess_methods, dav(WebDAV module configuration), ssi(server side inclusion), csrf, xss, ssl_certificate, xst(cross site tracing), blind_sqli, phishing_vector, response_splitting, rfd(reflected file download), frontpage(tries uploading files using frontpage extensions).

W3AF was able to quite easily find XSS, CSRF and SQL injection points. It also informed about unhandled errors and possible click-jacking on both sites.

However, it also showed a few false positives (showing sql injections as eval vulnerabilities, path disclosure) and missed a few vulnerabilities that our other tools found,

like buffer overflows.

Overall, I'd say W3AF is a Jack of all Trades. If the functionality is not enough, however, it's possible to write custom plugins.

3.4 SQL Inject Me

Already installed in Firefox on the Samurai machines, this tool has one goal and is pretty straightforward to use. It attacks the chosen form fields with brute-force SQL Injection attacks, hoping to get some intel on possible attack vectors. Fields that you don't want to attack can be supplied with a fixed value. Attacking the username in the login form of Online Banking passes 15479 of 15480 tests, but the input name `' OR username IS NOT NULL OR username = '` returns a 302 error code. This shows us that the username field might be vulnerable.

However, seeing that only one test shows some kind of reaction in a place where we afterwards confirmed the existence of a rather simple vulnerability (`admin'#`) makes "SQL Inject Me" seem like a rather weak tool.

Something else to behold is that when brute-forcing any kind of form where data is created (like registration), there will be more than 15.000 similar actions performed within a few seconds. The tool doesn't do anything to hide its intentions.

3.5 Burp Suite

We also analyzed the applications regarding SQL injections using Burp. However, we did not find more vulnerabilities. The sequencer is useful for confirming high enough entropy for session IDs.

3.6 Firebug / Chromium Developer Tools

We used the developer tools to inspect the HTML structure of the site. With these tools, we found some HTML5 input patterns, which gave us hints which parameters to tweak using an interception proxy. It's also possible to resend requests with manipulated parameters. And if Javascript was heavily used or XSS would be blocked in some half vulnerable way, it can debug the Javascript code step-by-step.

4 Detailed Report

4.1 Configuration and Deploy Management Testing

4.1.1 Test File Extensions Handling for Sensitive Information

Online Banking

Observation	We found various files which are served as plain text but are PHP source files. One of these files contains the credentials of the mail server. We were also able to download the compiled executable as well as the source code of the batch file parser.
Discovery	Using the OWASP ZAP tool, we used the forced browse functionality on `/InternetBanking/`. We received a list of files which were found using this tool, see below.
Likelihood	This can be tested by anyone who enters specific strings into the address bar of a browser. However, the likelihood of this vulnerability is much higher if the attacker uses specific tools which test specific paths systematically.
Impact	The attacker can get sensitive information, e.g. credentials to the mail server or the database. He can analyze the source of the parser and find vulnerabilies there.
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	High
Intigrity	No Impact
Availability	No Impact

Forced Browsing Results:

http://vm/InternetBanking/
http://vm/InternetBanking/DataAccess/
http://vm/InternetBanking/auth/
http://vm/InternetBanking/client/
http://vm/InternetBanking/controller/
http://vm/InternetBanking/controller/clientController.php
http://vm/InternetBanking/controller/clientFunctions.inc
http://vm/InternetBanking/controller/employeeController.php
http://vm/InternetBanking/controller/employeeFunctions.inc
http://vm/InternetBanking/controller/loginController.php
http://vm/InternetBanking/controller/logoutController.php
http://vm/InternetBanking/controller/registrationController.php
http://vm/InternetBanking/css/
http://vm/InternetBanking/css/bootstrap-theme.css
http://vm/InternetBanking/css/bootstrap-theme.css.map
http://vm/InternetBanking/css/bootstrap-theme.min.css
http://vm/InternetBanking/css/bootstrap.css
http://vm/InternetBanking/css/bootstrap.css.map
http://vm/InternetBanking/css/bootstrap.min.css
http://vm/InternetBanking/employee/
http://vm/InternetBanking/fonts/
http://vm/InternetBanking/fonts/glyphicons-halflings-regular.eot
http://vm/InternetBanking/fonts/glyphicons-halflings-regular.svg
http://vm/InternetBanking/fonts/glyphicons-halflings-regular.ttf
http://vm/InternetBanking/fonts/glyphicons-halflings-regular.woff
http://vm/InternetBanking/fonts/glyphicons-halflings-regular.woff2
http://vm/InternetBanking/index/
http://vm/InternetBanking/js/
http://vm/InternetBanking/js/bootstrap.js
http://vm/InternetBanking/js/bootstrap.min.js
http://vm/InternetBanking/js/npm.js
http://vm/InternetBanking/login/
http://vm/InternetBanking/logout/
http://vm/InternetBanking/model/
http://vm/InternetBanking/model/Payment.class
http://vm/InternetBanking/model/PaymentRequest.class
http://vm/InternetBanking/model/User.class

http://vm/InternetBanking/model/UserRequest.class
http://vm/InternetBanking/parser/
http://vm/InternetBanking/parser/Makefile
http://vm/InternetBanking/parser/exec
http://vm/InternetBanking/parser/main.c
http://vm/InternetBanking/parser/mysql_query_function.c
http://vm/InternetBanking/parser/mysql_query_function.h
http://vm/InternetBanking/registration/
http://vm/InternetBanking/view/
http://vm/InternetBanking/view/account.inc
http://vm/InternetBanking/view/accounts.inc
http://vm/InternetBanking/view/approvepayments.inc
http://vm/InternetBanking/view/approveregistrations.inc
http://vm/InternetBanking/view/client.inc
http://vm/InternetBanking/view/employee.inc
http://vm/InternetBanking/view/file.inc
http://vm/InternetBanking/view/history.inc
http://vm/InternetBanking/view/historypdf.inc
http://vm/InternetBanking/view/login.inc
http://vm/InternetBanking/view/online.inc
http://vm/InternetBanking/view/registration.inc

SecureBank

Observation	We found some HTML snippets, which do not contain any sensitive information, and the compiled executable of the transaction file parser. We also found that the library TCPDF appears to be used.
Discovery	Using the OWASP ZAP tool, we used the forced browse functionality on `/seccoding-2015/`. We received a list of files which were found using this tool, see below.
Likelihood	This can be tested by anyone who enters specific strings into the address bar of a browser. However, the likelihood of this vulnerability is much higher if the attacker uses specific tools which test specific paths systematically.
Impact	The attacker only has access to the parser executable, which might contain information about the database connection. He can analyze the parser and find vulnerabilies there.
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	Low
Intigrity	No Impact
Availability	No Impact

Forced Browsing Results:

```
http://vm/seccoding-2015/
http://vm/seccoding-2015/database/
http://vm/seccoding-2015/html/
http://vm/seccoding-2015/html/auth.html
http://vm/seccoding-2015/html/dberror.html
http://vm/seccoding-2015/html/default.html
http://vm/seccoding-2015/html/display_userstate.html
```

http://vm/seccoding-2015/html/display_userstate_table.html
http://vm/seccoding-2015/html/dologin_fail.html
http://vm/seccoding-2015/html/dologin_success.html
http://vm/seccoding-2015/html/doregister_fail.html
http://vm/seccoding-2015/html/doregister_success.html
http://vm/seccoding-2015/html/edoverify_fail.html
http://vm/seccoding-2015/html/ehome.html
http://vm/seccoding-2015/html/ehome_transaction.html
http://vm/seccoding-2015/html/ehome_user.html
http://vm/seccoding-2015/html/entry_page.html
http://vm/seccoding-2015/html/etakeover.html
http://vm/seccoding-2015/html/etransactionpdf_fail.html
http://vm/seccoding-2015/html/home.html
http://vm/seccoding-2015/html/login.html
http://vm/seccoding-2015/html/register.html
http://vm/seccoding-2015/html/root.html
http://vm/seccoding-2015/html/simpleRoot.html
http://vm/seccoding-2015/html/transaction.html
http://vm/seccoding-2015/html/udotransactionupload_fail.html
http://vm/seccoding-2015/html/utransaction.html
http://vm/seccoding-2015/html/utransactionupload.html
http://vm/seccoding-2015/index/
http://vm/seccoding-2015/login/
http://vm/seccoding-2015/parser/
http://vm/seccoding-2015/parser/parser
http://vm/seccoding-2015/tcpdf/
http://vm/seccoding-2015/tcpdf/CHANGELOG.TXT
http://vm/seccoding-2015/tcpdf/LICENSE.TXT
http://vm/seccoding-2015/tcpdf/README.TXT
http://vm/seccoding-2015/tcpdf/composer.json
http://vm/seccoding-2015/tcpdf/config/
http://vm/seccoding-2015/tcpdf/config/tcpdf_config.php
http://vm/seccoding-2015/tcpdf/fonts/
http://vm/seccoding-2015/tcpdf/include/
http://vm/seccoding-2015/tcpdf/include/barcodes/
http://vm/seccoding-2015/tcpdf/include/barcodes/datamatrix.php
http://vm/seccoding-2015/tcpdf/include/barcodes/pdf417.php
http://vm/seccoding-2015/tcpdf/include/barcodes/qrcode.php
http://vm/seccoding-2015/tcpdf/include/sRGB.icc

http://vm/seccoding-2015/tcpdf/include/tcpdf_colors.php
http://vm/seccoding-2015/tcpdf/include/tcpdf_filters.php
http://vm/seccoding-2015/tcpdf/include/tcpdf_font_data.php
http://vm/seccoding-2015/tcpdf/include/tcpdf_fonts.php
http://vm/seccoding-2015/tcpdf/include/tcpdf_images.php
http://vm/seccoding-2015/tcpdf/include/tcpdf_static.php
http://vm/seccoding-2015/tcpdf/tcpdf.php
http://vm/seccoding-2015/tcpdf/tcpdf_autoconfig.php
http://vm/seccoding-2015/tcpdf/tcpdf_barcodes_1d.php
http://vm/seccoding-2015/tcpdf/tcpdf_barcodes_2d.php
http://vm/seccoding-2015/tcpdf/tcpdf_import.php
http://vm/seccoding-2015/tcpdf/tcpdf_parser.php
http://vm/seccoding-2015/tcpdf/tools/
http://vm/seccoding-2015/tcpdf/tools/convert_fonts_examples.txt
http://vm/seccoding-2015/tcpdf/tools/tcpdf_addfont.php

Comparison

The web application of the SecureBank discloses less sensitive information. However, both applications disclose information which should not be available to unauthorized persons.

4.1.2 Test HTTP Methods

Online Banking

Observation	The server responded that the method `POST`, `GET`, `OPTIONS` and `HEAD` are supported.
Discovery	We submitted the request `OPTIONS / HTTP/1.1` to the server via NetCat on port 80.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The server responded that the method `POST`, `GET`, `OPTIONS` and `HEAD` are supported.
Discovery	We submitted the request `OPTIONS / HTTP/1.1` to the server via NetCat on port 80.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

There are no significant differences between both applications.

4.1.3 Test HTTP Strict Transport Security

Online Banking

Observation	The server did not send any `Strict-Transport-Security` header.
Discovery	Executing the command `curl -s -D- http://vm/InternetBanking/ | grep Strict` resulted in no results.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The server did not send any `Strict-Transport-Security` header.
Discovery	Executing the command `curl -s -D- http://vm/InternetBanking/ | grep Strict` resulted in no results.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

There are no significant differences between both applications.

4.1.4 Test RIA cross domain policy

Online Banking

Observation	No cross domain policy files were found.
Discovery	We scanned the traffic using ZAP.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	No cross domain policy files were found.
Discovery	We scanned the traffic using ZAP.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

There are no significant differences between both applications.

4.2 Identity Management Testing

4.2.1 Test Role Definitions

Online Banking

Observation

We found the following functionality for the different roles:

	Client	Employee
View own account	×	×
View own transaction history	×	–
Create new transactions	×	–
View account and transaction history of clients and employees	–	×
Change account details and balance of clients and employees	–	×
Approve transactions	–	×
Approve registrations of clients and employees	–	×

We noticed that there are links to view the transaction history and change the account balance of employees, too.

Discovery

We gathered the information by exploring the web application interface manually.

Impact

n/a

Likelihood

n/a

CVSS

n/a

SecureBank

Observation We found the following functionality for the different roles:

	Client	Employee
View own account	×	–
View own transaction history	×	–
Create new transactions	×	–
View account and transaction history of clients	–	×
Approve transactions	–	×
Approve registrations of clients and employees	–	×

Discovery We gathered the information by exploring the web application interface manually.

Impact n/a

Likelihood n/a

CVSS n/a

Comparison

The SecureBank web application does not offer a possibility for an employee to change the account balance of a client. However, the Online Banking application allows to view the transaction history and change the account balance also for employees, which have no account. This behaviour might be confusing.

4.2.2 Test User Registration Process

Online Banking

Observation	For registration, a username, an e-mail address, a password and whether the registrant is a client or an employee are needed. Anyone can register for access. The registration has to be approved by an employee before the registrant can use the account. A person can register only one time with the same e-mail address. However, a person can register many times with the same username. (The activation of such an account fails with a database error.) We could not find out, whether the registrants are verified personally before the approval.
Discovery	We tried to register several accounts with the same e-mail address and/or username using the web application.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	For registration, the full name, an e-mail address, a password and whether the registrant is a client or an employee are needed. Anyone can register for access. The registration has to be approved by an employee before the registrant can use the account. A person can register only one time with the same e-mail address. We could not find out, whether the registrants are verified personally before the approval.
Discovery	We tried to register several accounts with the same e-mail address and/or names using the web application.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The Online Banking web application allows the double-registration of the same username at first, it only fails at the activation. This behaviour is confusing. Also, the application should ask for the full name be able to verify the name. Otherwise, there are no significant differences between both applications.

4.2.3 Test Account Provisioning Process

Online Banking

Observation	There is no way to change the role of a user. Account requests (both, client and employee) must be approved by an employee.
Discovery	We followed the links in the user interface and tried to login as a non-verified user.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	There is no way to change the role of a user. Account requests (both, client and employee) must be approved by an employee.
Discovery	We followed the links in the user interface and tried to login as a non-verified user.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

There are no significant differences between both applications.

4.2.4 Testing for Account Enumeration and Guessable User Account

Online Banking

Observation	There are no differences in the servers response for not activated accounts, valid usernames and invalid usernames.
Discovery	We tested the login for activated and non-activated accounts, existing and not-existing usernames and valid or invalid passwords.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	There are no differences in the servers response for not activated accounts, valid usernames and invalid usernames.
Discovery	We tested the login for activated and non-activated accounts, existing and not-existing usernames and valid or invalid passwords.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

There are no significant differences between both applications.

4.2.5 Testing for Weak or unenforced username policy

Online Banking

Observation	We were not able to find a username policy.
Discovery	We tested various usernames.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The username has to be a valid e-mail address of the client/employee. There is no policy regarding the e-mail address.
Discovery	We tested valid and invalid e-mail addresses.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The only difference between the applications is that the Online Banking application uses usernames, which might have less correlation to the user than the e-mail address.

4.3 Authentcation Testing

4.3.1 Testing for Credentials Transported over Encrypted Channel

See Section 4.8.1.

4.3.2 Testing for default credentials

We decided to not test for default credentials, because we are working with custom software and therefore assume that all users and administrators choose secure passwords.

4.3.3 Testing for Weak lock out mechanism

Online Banking

Observation	We were not able to find any lock out mechanism. Therefore, brute force attacks on passwords are possible.
Discovery	We entered a valid username and incorrect passwords 10 times, and always got the error message about an incorrect password. Afterwards, we were able to log in with a correct password.
Impact	An attacker can brute-force the password of any user and therefore take the user over.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	Low
Intigrity	Low
Availability	No Impact

SecureBank

Observation	We were not able to find any lock out mechanism. Therefore, brute force attacks on passwords are possible.
Discovery	We entered a valid username and incorrect passwords 10 times, and always got the error message about the failed login. Afterwards, we were able to log in with a correct password.
Impact	An attacker can brute-force the password of any user and therefore take the user over.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	Low
Intigrity	Low
Availability	No Impact

Comparison

Both applications do not provide any lock out mechanism.

4.3.4 Testing for bypassing authentication schema

Online Banking

Observation	We were able to bypass the authentication schema via a SQL injection. This gave us the ability to login as any user without knowing the password.
Discovery	Using the fuzzer `jbrofuzz / SQL Injection` of ZAP on the username field of the login page, we were able to login as `admin` or another user without knowing the password. We had no success with direct page requests, modifiying the session ID and parameter modification.
Impact	An attacker can take over a user without knowing the valid access credentials.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	Low
Intigrity	Low
Availability	No Impact

SecureBank

Observation	We were not able to bypass the authentication schema.
Discovery	Using the fuzzer `jbrofuzz / SQL Injection` of ZAP and SQLmap on the username field of the login page, we were not able to find SQL injection vulnerabilities to bypass the authentication schema. We also had no success with direct page requests, modifiying the session ID and parameter modification.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The Online Banking web application provides a way to bypass the authentication schema via SQL injection. The SecureBank application does not offer such vulnerabilities.

4.3.5 Testing for Vulnerable Remember Password

We did not found a remember password functionality, so we decided to not further test on this.

4.3.6 Testing for Browser Cache Weakness

Online Banking

Observation	Clicking the back button in the browser does not cause a re-login. All sites have the header `Cache-Control: no-store, no-cache, must-revaildate, post-check=0, pre-check=0` and the `Pragma: no-cache` as well as an `Expires: <date in the past>` header set.
Discovery	Using ZAP, we analyzed the response header for different pages which are only available when a user is logged in.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	Clicking the back button in the browser does not cause a re-login. All sites have the header `Cache-Control: no-store, no-cache, must-revaildate, post-check=0, pre-check=0` and the `Pragma: no-cache` as well as an `Expires: <date in the past>` header set.
Discovery	Using ZAP, we analyzed the response header for different pages which are only available when a user is logged in.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

There is no significant difference between both applications.

4.3.7 Testing for Weak password policy

Online Banking

Observation	There is a password policy enforced, which states that a password has have a length ≥ 6 and has to include at least one number, one lowercase character, one uppercase character and one symbol. There is no way to change the password. The password does not expire.
Discovery	We tested various passwords, like `123456`. If a password does not match the policy, an error message is shown which informs the user about the policy.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The password has to have a length ≥ 8. There is no way to change the password. The password does not expire.
Discovery	We tested various passwords, like `123456`. If a password does not match the policy, an error message is shown which informs the user about the policy.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The Online Banking application enforces a more strict password policy than the SecureBank application. This reduces the risk of brute force attacks.

4.3.8 Testing for Weak security question/answer

We could not find such functionality in both application. Therefore, we decidede to not proceed testing on this.

4.3.9 Testing for Weak password change or reset functionalities

We could not find such functionality in both application. Therefore, we decidede to not proceed testing on this.

4.3.10 Testing for Weaker authentication in alternative channel

We could not find an alternative channel for authentication. Therefore, we decidede to not proceed testing on this.

4.4 Authorization Testing

4.4.1 Testing Directory traversal/file include

Online Banking

Observation	We could not get access to sensitive files like `/etc/passwd`.
Discovery	We manually tried to access the client main site with different values for the parameter `action`, like `action=../../../etc/passwd`.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We could not get access to sensitive files like `/etc/passwd`.
Discovery	We manually tried to access the site with different values for the parameter `page`, like `page=../../../etc/passwd`.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications do not provide file include vulnerabilities.

4.4.2 Testing for bypassing authorization schema

Online Banking

Observation	We were unable to bypass the authorization schema
Discovery	We manually tried to access the employee page while being logged in as a client and got the error message `Not employee`.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We were unable to bypass the authorization schema
Discovery	We manually tried to access the employee pages like `edoverify` while being logged in as a client and got the error message `You are not authenticated to do this`.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications do not provide vulnerabilities that allow to bypass the authorization schema.

4.4.3 Testing for Privilege Escalation

Online Banking

Observation	We were unable to get the privileges of another user.
Discovery	We manually tried to access the employee page while being logged in as a client and got the error message `Not employee`.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We were unable to get the privileges of another user.
Discovery	We manually tried to access the employee pages like `edoverify` while being logged in as a client and got the error message `You are not authenticated to do this`. We also tried to change the account number of the `userid` GET parameter when downloading a PDF, without success.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications do not provide vulnerabilities that allow a privilege escalation.

4.4.4 Testing for Insecure Direct Object References

Online Banking

Observation	We were unable to find insecure direct object references.
Discovery	We manually changed the GET parameter `action` using ZAP. However, we either got an empty page or the requested page.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We were unable to find insecure direct object references.
Discovery	We manually changed the GET parameter `page`. However, we either got an error message or the requested page.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications do not provide vulnerabilities that allow insecure direct object references.

4.5 Session Management Testing

4.5.1 Testing for Bypassing Session Management Schema

Online Banking

Observation	Only cookies for the PHP session ID were used. This hints at the cookies being rather secure. Using BurpSuite's sequencer we confirm that the entropy is excellent. However, the cookies are sent over an unencrypted connection, which can be exploited with a MitM attack.
Discovery	BurpSuite
Impact	Medium/High, depending on the victim's privileges. By stealing the session from a user (admin or not), we would be able to impersonate them and do actions in their name.
Likelihood	High
Access Vector	Adjacent Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes, the user has to do any action for which the cookie will be transmitted.
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

SecureBank

Observation	Only cookies for the PHP session ID were used. This hints at the cookies being rather secure. Using BurpSuite's sequencer we confirm that the entropy is excellent. However, the cookies are sent over an unencrypted connection, which can be exploited with a MitM attack.
Discovery	BurpSuite
Impact	Medium/High, depending on the victim's privileges. By stealing the session from a user (admin or not), we would be able to impersonate them and do actions in their name.
Likelihood	High
Access Vector	Adjacent Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes, the user has to do any action for which the cookie will be transmitted.
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

Comparison

Both applications are equally secure/insecure regarding this.

4.5.2 Testing for Cookies attributes

Online Banking

Observation	Since only the PHP session ID is saved as cookie, no attributes are saved in any cookies. Obviously there's no attack to execute
Discovery	BurpSuite
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	Only cookies for the PHP session ID were used. This hints at the cookies being rather secure. Using BurpSuite's sequencer we confirm that the entropy is excellent. Not attacks are possible at this point.
Discovery	BurpSuite
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications have no attributes other than the Session ID and are thus secure under this aspect.

4.5.3 Testing for Session Fixation

Online Banking

Observation	Cookies aren't renewed when logging in.
Discovery	Firefox Developer tools
Impact	Medium/High, depending on the victim's privileges. An attacker might create a session and make the user authenticate with it, thus giving the attacker access to the victim's session.
Likelihood	Medium
Access Vector	Adjacent Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes, the user has to do any action for which the cookie will be transmitted.
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

SecureBank

Observation	Cookies aren't renewed when logging in.
Discovery	Firefox Developer tools
Impact	Medium/High, depending on the victim's privileges. An attacker might create a session and make the user authenticate with it, thus giving the attacker access to the victim's session.
Likelihood	Medium
Access Vector	Adjacent Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes, the user has to do any action for which the cookie will be transmitted.
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

Comparison

Both applications have the session fixation vulnerability.

4.5.4 Testing for Exposed Session Variables

Online Banking

Observation	The cookies are sent over an unencrypted connection. Cookies also aren't renewed when logging in.
Discovery	BurpSuite
Impact	Medium/High, depending on the victim's privileges. By using MitM to steal the session from a user (admin or not), we would be able to impersonate them and do actions in their name. This compromises the integrity of the application.
Likelihood	High
Access Vector	Adjacent Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes, the user has to do any action for which the cookie will be transmitted.
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

SecureBank

Observation	The cookies are sent over an unencrypted connection. Cookies also aren't renewed when logging in.
Discovery	BurpSuite
Impact	Medium/High, depending on the victim's privileges. By using MitM to steal the session from a user (admin or not), we would be able to impersonate them and do actions in their name. This compromises the integrity of the application.
Likelihood	High
Access Vector	Adjacent Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes, the user has to do any action for which the cookie will be transmitted.
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

Comparison

Both applications are equally insecure regarding this.

4.5.5 Testing for Cross Site Request Forgery

Online Banking

Observation	There is no mechanism to prohibit CSRF. GET requests aren't enough, though, since all of the important data is only transmitted via POST requests.
Discovery	Manual test
Impact	Medium/High, depending on the victim's privileges. By getting our victim to open a malicious web page, that creates a POST request with JS, we would be able to perform actions in their name.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

SecureBank

Observation	There is no mechanism to prohibit CSRF. GET requests aren't enough, though, since all of the important data is only transmitted via POST requests.
Discovery	Manual test
Impact	Medium/High, depending on the victim's privileges. By getting our victim to open a malicious web page, that creates a POST request with JS, we would be able to perform actions in their name.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	Yes
Scope	Unchanged
Confidentiality	Low
Intigrity	High
Availability	None

Comparison

Both applications are equally insecure regarding this.

4.5.6 Testing for logout functionality

Online Banking

Observation	The logout button is easily visible and works as it should: It invalidates the session.
Discovery	Manual test
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The logout button is easily visible and works as it should: It invalidates the session.
Discovery	Manual test
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications have a correctly implemented logout function.

4.5.7 Test Session Timeout

Online Banking

Observation	The user isn't logged out automatically.
Discovery	Manual test
Impact	Medium/High, depending on the victim's privileges. All people using the same (maybe public) machine afterwards have access to the victim's account.
Likelihood	High
Access Vector	Physical access to the machine that the victim used
Access Complexity	Low
Privileges Required	None
User Interaction	Yes
Scope	Unchanged
Confidentiality	High
Intigrity	High
Availability	None

SecureBank

Observation	The user is automatically logged out after 10 minutes of inactivity.
Discovery	Manual test
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

SecureBank is a lot more secure in this regard.

4.5.8 Testing for Session puzzling

Online Banking

Observation	We found no way to perform Session Puzzling. The same cookie is used for all interactions.
Discovery	Zap, w3af
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We found no way to perform Session Puzzling. The same cookie is used for all interactions.
Discovery	Zap, w3af
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications aren't vulnerable to Session puzzling.

4.6 Data Validation Testing

4.6.1 Testing for Reflected Cross Site Scripting

Online Banking

Observation	The only GET parameters are "action" and if logged in as employee, "account". When entering an action name that doesn't exist, the fake name is not displayed in any kind of error message. The same happens if the input account doesn't exist. If the account exists and the username is equal to some XSS attack, that should be considered a stored XSS attack instead. Thus, there's no attack vector for Reflected XSS.
Discovery	w3af (XSS plugin) and manually
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The only GET parameter is "page". When entering a page name that doesn't exist, the fake name is not displayed in any kind of error message. Thus, there's no attack vector for Reflected XSS.
Discovery	w3af (XSS plugin) and manually
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications allowed no reflected XSS.

4.6.2 Testing for Stored Cross Site Scripting

Online Banking

Observation	Stored XSS can be injected via the username when registering and into the purpose field when making transactions. The email used for registration, however, is checked for validity. User names can't be longer than 32 signs. This is just enough to inject something like `<script src=//bit.ly/1MvAdrm>`. Users who register after this exploit has been used won't be shown, since the end tag is missing. For the transaction's description there is more space, but an account is needed.
Discovery	TODO
Impact	Using this it's possible to e.g. automatically accept payments of over 10000 as soon as an admin opens the "Approve Transfers" page. Given a user's TAN, it's also possible to transfer their money into one's own account by sending them a manipulated transaction. The manipulated transaction just has to include code that creates a transaction in the victim's name. Furthermore, it's possible to steal sessions or, by injecting fake login forms even credentials of admins. By injecting into the username, the attacker can also auto-accept themselves as real user as soon as an admin opens the "Approve Registrations" tab.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	N/L: None for the registration XSS, Low user privileges for XSS in transactions
User Interaction	Yes
Scope	Unchanged
Confidentiality	High
Intigrity	High
Availability	Low

SecureBank

Observation	User names are escaped, so XSS is impossible here. Transaction descriptions, however, still allow XSS. It isn't triggered on the page where transactions will be verified by employees, but by opening the recipient's (or the attacker's) transaction history. This way it's just slightly harder to get the admin's credentials. Getting to the recipient's account is as easy as in Online Banking's app.
Discovery	TODO
Impact	Using this it's possible to e.g. automatically accept payments of over 10000 as soon as an admin opens the "Approve Transfers" page. Given a user's TAN, it's also possible to transfer their money into one's own account by sending them a manipulated transaction. The manipulated transaction just has to include code that creates a transaction in the victim's name. Furthermore, it's possible to steal sessions or, by injecting fake login forms even credentials of admins.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	normal user privileges
User Interaction	Yes
Scope	Unchanged
Confidentiality	High
Intigrity	High
Availability	None

Comparison

Both applications allowed for stored XSS, but Online Banking was easier to compromise (no account needed). Our application is the better one in this regard.

4.6.3 Testing for HTTP Verb Tampering

Online Banking

Observation	When trying all HTTP methods, CONNECT returns "Bad Request", while OPTIONS returns "OK". Otherwise, all methods except GET, HEAD and POST are not allowed
Discovery	script http_tampering.sh in phase2/scripts/ (from OWASP website)
Impact	The web server might respond in unexpected ways to these methods and might execute scripts without privilege control when using HEAD instead of GET.
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	When trying all HTTP methods, CONNECT returns "Bad Request", while OPTIONS returns "OK". Otherwise, all methods except GET, HEAD and POST are not allowed
Discovery	script http_tampering.sh in phase2/scripts/ (from OWASP website)
Impact	The web server might respond in unexpected ways to these methods and might execute scripts without privilege control when using HEAD instead of GET.
Likelihood	n/a
CVSS	n/a

4.6.4 Testing for HTTP Parameter pollution

Online Banking

Observation	The application behaves as it should, ignoring all but the last parameter with the same name. Trying to e.g. pollute the email during registration with a value that has the wrong format shows "Email is invalid".
Discovery	Manipulated parameters using BurpSuite manually, since HPP apparently can't really be tested using tools (too many false positives)
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	The application behaves as it should, ignoring all but the last parameter with the same name. Trying to e.g. pollute the email during registration registers the user with the last email parameter, if the email hasn't been taken.
Discovery	Manipulated parameters using BurpSuite manually, since HPP apparently can't really be tested using tools (too many false positives)
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both apps are protected against HTTP parameter pollution.

4.6.5 Testing for SQL injection

Online Banking

Observation	We were able to log into the application as any user without knowing the passwordand to perform transactions with any unused TAN of any user when uploading a transaction batch file.
Discovery	We used the fuzzer of ZAP on the username field of the login form, with the query parameters `username=admin&password=123`, where the fuzzing point was at the end of the username. We compared the answers of the server and found that in one case we got redirected to the employee page. We analyzed the login page also using SQLmap, which delivered a similar result. Additionally, we ran SQLmap on the fields of the perform transaction form, it returned that none of the four inputs where exploitable for SQL injection. We tested the fields of the transaction batch form manually by trying some standard SQL injection parts (e.g. `';-` and `";-`). We noted that the application gives no usual feedback when uploading a transaction file where the username or the comment contains strings that apparantly results in a syntactically erroneous SQL query. We continued to determine the table structure using a brute force script which tests for some expectable table and column names, the results can be found below. Finally, we were able to exploit the SQL injection vulnerability in the TAN field to use any unused TAN from any user.
Impact	An attacker can log into any account, of which he knows the username, and perform transactions without knowing valid TANs for that account by uploading a transaction batch file. Furthermore, as an attacker can take over an administrator account, if he knows the username, he has access to all accounts and can change the account balances at will. Also, an attacker can analyze the structure of a database.
Likelihood	High

Login vulnerability	**Access Vector**	Network
	Access Complexity	Low
	Privileges Required	None
	User Interaction	None
	Scope	Unchanged
	Confidentiality	High
	Intigrity	High
	Availability	No Impact

TAN vulnerability	**Access Vector**	Network
	Access Complexity	Low
	Privileges Required	Low
	User Interaction	None
	Scope	Unchanged
	Confidentiality	High
	Intigrity	High
	Availability	No Impact

Results of the brute-force script

```
Table payment: id, trancode, payer, receipt, amount, purpose
Table user: id, balance, email, username, password, isemployee
Table userrequest: id, email, username, password, isemployee
Table paymentrequest: id, trancode, payer, receipt, amount, purpose
Table trancode: id, clientid
```

SecureBank

Observation	No SQL injections found
Discovery	We used the fuzzer of ZAP on the username field of the login form, with the query parameters `username=admin@localhost&password=123`, where the fuzzing point was at the end of the username. We also analyzed the login page also using SQLmap. Using both methods, we were not able to log in without knowing the password. Additionally, we ran SQLmap on the fields of the registration form and the perform transaction form, it returned that none of the four inputs where exploitable for SQL injection.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The Online Banking web application is vulnerable to SQL injection at different places, allowing an attacker to take over user accounts and transfer money without knowing any credentials. In the web application of the SecureBank, we were not able to find SQL injection vulnerabilities.

4.6.6 Testing for LDAP Injection

4.6.7 Testing for ORM Injection

4.6.8 Testing for XML Injection

4.6.9 Testing for SSI Injection

4.6.10 Testing for XPath Injection

We did not test any of these injection types as it appears that these techniques are not used in any of the two applications. Therefore, we decided to not further investigate in these types of injection.

4.6.11 IMAP/SMTP Injection

We don't want to attack the used mail servers, since they're probably not part of the programming part. And we don't want to go to prison for trying to hack gmail.

4.6.12 Testing for Code Injection

Online Banking

Observation	The only code injection we found is the SQL injection in the uploaded file, as described in 4.6.5. Apart from that nothing was found.
Discovery	Tested the action parameter in the GET requests. Our tools found no Code Injections either.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	No vulnerability found.
Discovery	Tested the page parameter in the GET requests. Our tools found no Code Injections either.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

If we count the SQL injection in the uploaded file to the point about SQL injection, both applications perform the same.

4.6.13 Testing for Local/Remote File Inclusion

Online Banking

Observation	No paths are supplied through any of the GET or POST requests, so the application is secure.
Discovery	Tested the action parameter in the GET requests. Our tools found no File Inclusion points either.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	No paths are supplied through any of the GET or POST requests, so the application is secure.
Discovery	Tested the action parameter in the GET requests. Our tools found no File Inclusion points either.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications are equally secure regarding Local/Remote File Inclusion.

4.6.14 Testing for Command Injection

Online Banking

Observation	W3af found two places where it suspected possible command injection via eval(). Both of these places turned out to just be SQL injection again, though. Apart from this we found no command injection.
Discovery	W3af and then manual testing.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We found no places where command injection is possible
Discovery	W3AF, ZAP
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications allow no command injections (to the OS).

4.6.15 Testing for Buffer overflow

Online Banking

Observation	We were able to induce a buffer overflow by putting a string of 258 as into the TAN purpose field of the uploaded transaction batch file, although we could not identify whether this a stack or a heap buffer overflow. We were not able to cause a buffer overflow by using formatting characters. The transaction gets executed correctly, except that the purpose field is empty, though.
Discovery	Using the fuzzer of ZAP, we used the `jbrofuzz / Buffer Overflows / Long string's of a's` fuzzing base on the transaction purpose field of the uploaded transaction batch file. For string smaller with a length smaller than 255 bytes, the transaction gets executed, and for very large strings, the application claims that the file is too large. However, in the other cases, the application gives no feedback, as described in Section 4.6.5. We also tried to cause buffer overflows caused by string format function by including strings like `%s` into the username or transaction purpose field.
Impact	As we could not identify whether this is a stack of a heap buffer overflow, we cannot clearly identify the impact. In the worst case, if the buffer overflow happens on the stack, an attacker has access to the database and the file system.
Likelihood	Medium
Access Vector	Network
Access Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Unchanged
Confidentiality	High
Intigrity	High
Availability	High

SecureBank

Observation	We were unable to produce a buffer overflow.
Discovery	We tried to use long strings of `as` on the username and the description field of the uploaded transaction batch file. However, the file was either rejected with the comment that it does not satisfy the specifications or the transaction was executed. We also tried to cause buffer overflows caused by string format function by including strings like `%s` into the username or transaction purpose field.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The SecureBank application is better in this regard, as it does not offer buffer overflow vulnerabilities.

4.6.16 Testing for incubated vulnerabilities

We did not test for incubated vulnerabilities for the reason of complexity.

4.6.17 Testing for HTTP Splitting/Smuggling

Online Banking

Observation	We were unable to find any HTTP splitting vulnerabilities.
Discovery	Using ZAP, we analyzed the response headers and found `Location` headers being sent. However, it did not appear that user inputs are sent as part of the location header.
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	We were unable to find any HTTP splitting vulnerabilities.
Discovery	Using ZAP, we analyzed the response headers and found `Location` headers being sent. However, it did not appear that user inputs are sent as part of the location header.
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications are on the same security level in this regard.

4.7 Error Handling

4.7.1 Analysis of Error Codes

Online Banking

Observation	We were not able to get reveal any information about the database, but we were able to get usual error codes of an apache server. The error pages also revealed information about the version, which is 2.2.22 on an Ubuntu server.
Discovery	By using the fuzzer `jbrofuzz / SQL Injection / MySQL Injection 101` of ZAP on the purpose field of the online transaction form, we got one response which simply stated `Database error`, but gave no further information. Using a browser, we requested the page `/InternetBanking/foobar/`, which returned a 404 error, with further information about the server operating system and the Apache version.
Impact	Using the apache version, an attacker can run known exploits for this specific version.
Likelihood	Medium
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	No Impact
Intigrity	No Impact
Availability	No Impact

SecureBank

Observation	We were not able to get reveal any information about the database, but we were able to get usual error codes of an apache server. The error pages also revealed information about the version, which is 2.2.22 on an Ubuntu server. On the login page, we found an error code if the login was unsuccessful. Apparently, the code is always 1, if the credentials are incorrect.
Discovery	By using the fuzzer `jbrofuzz / SQL Injection / MySQL Injection 101` of ZAP on the purpose field of the online transaction form, we were not able to produce a databse error. Using a browser, we requested the page `/seccoding-2015/foobar/`, which returned a 404 error, with further information about the server operating system and the Apache version.
Impact	Using the apache version, an attacker can run known exploits for this specific version.
Likelihood	Medium
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	No Impact
Intigrity	No Impact
Availability	No Impact

Comparison

Although the Online Banking application produces an error message about a database error, they provide the similar information in error messages.

4.7.2 Testing for Stack Traces

We were not able to produce stack traces in both applications using invalid inputs (e.g. negative numbers), or SQL injection.

4.8 Testing for weak Cryptography

4.8.1 Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection

Online Banking

Observation	We were unable to connect to the application via HTTPS.
Discovery	We tried to access the application via HTTPS on port 443.
Impact	An attacker can retrieve sensitive information by sniffing the network.
Likelihood	High
Access Vector	Adjacent Network
Access Complexity	Medium
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	High
Intigrity	No Impact
Availability	No Impact

SecureBank

Observation	We were unable to connect to the application via HTTPS.
Discovery	We tried to access the application via HTTPS on port 443.
Impact	An attacker can retrieve sensitive information by sniffing the network.
Likelihood	High
Access Vector	Adjacent Network
Access Complexity	Medium
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	High
Intigrity	No Impact
Availability	No Impact

Comparison

Both applications transport sensitive information about non-encrypted channels.

4.8.2 Testing for Padding Oracle

As there is no transport layer encryption in any of the two applications, we decided to not test for a padding oracle.

4.8.3 Testing for Sensitive information sent via unencrypted channels

All information is sent via an unencrypted channel. See Section 4.8.1

4.9 Business Logic Testing

4.9.1 Test Business Logic Data Validation

Online Banking

Observation	When uploading transaction files, it's possible to enter negative amounts, which results in gaining money at the "recipient's" cost.
Discovery	Manual testing
Impact	Attackers can steal money from anybody they want
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	Low
User Interaction	None
Scope	Unchanged
Confidentiality	No Impact
Intigrity	Low
Availability	No Impact

SecureBank

Observation	We found no ways to make unreasonable requests, like negative money transactions.
Discovery	Manual testing
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

The Online Banking fails here, where our application correctly validates business data.

4.9.2 Test Ability to Forge Requests

Online Banking

Observation	None of the forms have hidden input fields that save critical information. Editing a user as admin, e.g. has a hidden user ID field. But since admins are allowed to change any user's values, that's no vulnerability.
Discovery	Firefox dev tools
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	There's a hidden input field in the file upload form to limit the file size. But if we go around this by uploading by forging the request and increasing this value, we're still unable to upload files with a size of more than 3000.
Discovery	Firefox dev tools, manual
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications are securd against forged requests.

4.9.3 Test Integrity Checks

Online Banking

Observation	No vulnerability found.
Discovery	ZAP, W3AF
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	No vulnerability found.
Discovery	ZAP, W3AF
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications have sufficient integrity checks.

4.9.4 Test for Process Timing

Both

Observation	Apart from blind SQL, this is not applicable, since all actions are done in atomic steps, without locking.
Discovery	n/a
Impact	n/a
Likelihood	n/a
CVSS	n/a

4.9.5 Test Number of Times a Function Can be Used Limits

Both

Observation	There's no benefit for the users when using functions more often. Thus this isn't applicable for either of the apps.
Discovery	n/a
Impact	n/a
Likelihood	n/a
CVSS	n/a

4.9.6 Testing for the Circumvention of Work Flows

Both

Observation	All actions in both apps are atomic, so this doesn't apply.
Discovery	Browsing the app.
Impact	n/a
Likelihood	n/a
CVSS	n/a

4.9.7 Test Defenses Against Application Mis-use

Both

Observation	Defenses Against Application Mis-use aren't part of either of the applications. When sending many malicious requests at once, due to testing, neither of the applications tried to stop us. The same holds true for any attacker who wants to check the apps for vulnerabilities.
Discovery	ZAP, W3AF
Impact	This itself is barely a problem, since at best it DDOSes the app. But that would be noticeable then. It makes it easier for an attacker to find other vulnerabilities, though.
Likelihood	High
Access Vector	Network
Access Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	Low
Intigrity	None
Availability	Low

4.9.8 Test Upload of Unexpected File Types

Online Banking

Observation	While it's technically possible to upload files for transactions, these files aren't stored permanently (or at least we couldn't find them), so this vulnerability is not applicable. Online Banking also ensures that only .txt files can be uploaded.
Discovery	w3af, Zap
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	While it's technically possible to upload files for transactions, these files aren't stored permanently (or at least we couldn't find them), so this vulnerability is not applicable.
Discovery	w3af, Zap
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications seem secure, but Online Banking goes one step further and therefore wins in this regard.

4.9.9 Test Upload of Malicious Files

Online Banking

Observation	Only .txt files are allowed. If they then don't conform to the format for uploading transactions, we get the error message "Invalid format". Since transaction files probably also aren't stored permanently, we consider Online Banking to be secure.
Discovery	Uploaded eicar.com.txt (http://www.eicar.org/85-0-Download.html)
Impact	n/a
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	If uploaded files don't conform to the format for uploading transactions, we get an empty page, just like when the file isn't in the right transaction format. This could hint at a vulnerability. However, nothing hints at transaction files being stored anywhere.
Discovery	Uploaded eicar.com (http://www.eicar.org/85-0-Download.html)
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications seem to reject the uploaded file, but Online Banking makes that clearer, due to its error message. Since the uploaded files are just stored until they're parsed, vulnerability is unlikely.

4.10 Client Side Testing

We decided to not perform client-side testing, because we priotized it low. We focused on server-side attacks like XSS and SQL injection.

Acronyms

TUM Technische Universität München.