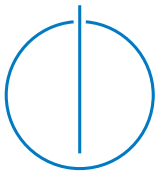
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Report

Black Box Testing Report

Alexis Engelke, Johannes Fischer, Ralph Schaumann,
Saurabh Nawalgaria



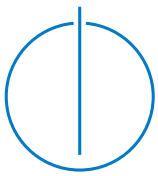
DEPARTMENT OF INFORMATICS

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

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	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

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.

3.2 SQLmap

Using SQLmap, we found the SQL injection vulnerability in the login form, which we found using ZAP earlier. SQLmap did not reveal further SQL injection possibilities.

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

TODO: Forced browsing results.

SecureBank

Observation	We found some HTML snippets, which do not contain any sensitive information, and the compiled executable of the transaction file parser.
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

TODO: Forced browsing results.

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

TODO!

Online Banking

Observation	
Discovery	
Impact	
Likelihood	n/a
CVSS	n/a

SecureBank

Observation	
Discovery	
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

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

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

TODO!

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)
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)
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

Both applications allowd 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 "$< scriptsrc = //bit.ly/1MvAdrm >$". 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 priviledges 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 priviledges
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

TODO

Observation	When trying all HTTP methods, CONNECT returns "Bad Request", while OPTIONS returns "OK". Otherwise, all methods except GET 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.
Likelihood	High
Access Vector	HTTP request for CONNECT or OPTIONS
Access Complexity	
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality	
Intigrity	
Availability	

SecureBank

Observation	When trying all HTTP methods, CONNECT returns "Bad Request", while OPTIONS returns "OK". Otherwise, all methods except GET 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.
Likelihood	
Access Vector	
Access Complexity	
Privileges Required	
User Interaction	
Scope	Unchanged
Confidentiality	
Intigrity	
Availability	

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

Access Vector	Network
Access Complexity	Low
Privileges Required	None
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	ZAP, Burp, w3af
Impact	n/a
Likelihood	n/a
CVSS	n/a

Comparison

While our SecureBank is protected against SQL injections, InternetBanking can be attacked easily and in many spots.

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 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).
TODO!

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

TODO!

4.9 Business Logic Testing

TODO!

4.10 Client Side Testing

TODO!