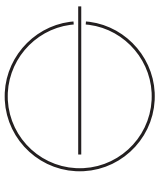
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Secure Coding Phase 2

Team 3: Patrick Sattler, Aurel Roci, Stefan Kohler



Contents

1 Time Tracking Table

2 Vulnerabilities Overview

Based on our testing, we identified the following vulnerabilities for the Secode21 Bank and the OnlineBanking Bank:

2.1 Secode21

2.1.1 Static Session ID

- Likelihood: *high*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-SESS-003 (see section **??**)

The session id is saved in form of the (static) user id in a cookie. This cookie can be used on any machine to take over the account of a user. The lifetime of this cookie is only limited by the cookie lifetime field.

2.1.2 Stored XSS in Registration

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-002 (see section **??**)

Using stored cross-site-scripting attacks, one can inject JavaScript code, that is run, when the Administrator/Employee logs in. Arbitrary code can be loaded from a third party page.

2.1.3 Missing Lock Out Mechanism

- Likelihood: *high*
- Implication: *medium*
- Risk: *medium*
- Reference: OWASP OTG-AUTHN-003 (see section **??**)

The application has no lock out mechanism, which allows brute force attacks on known usernames and testing for a valid password

2.2 Team3 Online Banking

2.3 Vulnerability Overview

3 Detailed Report

The following pages describe for each test how both applications Secode21 and Online Banking Bank performed. The test is divided in different sections following the OWASP Testing Guide v4.

3.1 Configuration and Deploy Management Testing

3.1.1 Test File Extensions Handling for Sensitive Information (OTG-CONFIG-003)

Secode21

Likelihood: 8
Impact: 5
Risk: 5

Secode21	
Observation	File extensions are handled correctly but while testing we found a folder called SQL with sql files and pdf files describing the database structure and the sql commands used by the web application.
Discovery	Thanks to the tool *dotdotpwn*, that tries automatically different URLs, we found the SQL folder. We passed the following parameters:
Likelihood	The likelihood is quite high that someone tries a tool to find these kind of vulnerabilities. There is no need for special knowledge because the tools work quite automatically without much configuration.
Implication	These vulnerabilities could help attackers to perform sql injection attacks because you know the database structure and the sql commands used in the implementation of the web application.
Recommendations	Block the access to sql files and to those folders that describe the web applications architecture.
Comparison	Our web application handles file extensions correctly, but it is possible to access the compiled c program that handles the batch files. This is a problem because you can reverse engineer the code and use the vulnerabilities found. This scenario is possible but is very complex.

Metric	Value
Access Vector	N
Attack Complexity	L
Privileges Required	N
User Interaction	N
Scope	U
Confidentiality Impact	L
Integrity Impact	N
Availability Impact	N

3.1.2 Test HTTP Methods (OTG-CONFIG-006)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	We did not observe any notable behavior.
Discovery	We used the Zed Attack Proxy (ZAP) to change the HTTP requests method to the ones listed below. The requests that were allowed responded with the index page or an empty body. The rejected requests responded with an error message in the body. Methods that were allowed • HEAD • OPTIONS • GET • POST • PUT Methods that were rejected • TRACE • CONNECT
Likelihood	N/A
Implication	N/A
Recommendations	N/A
Comparison	Both webapps have the same allowed methods.

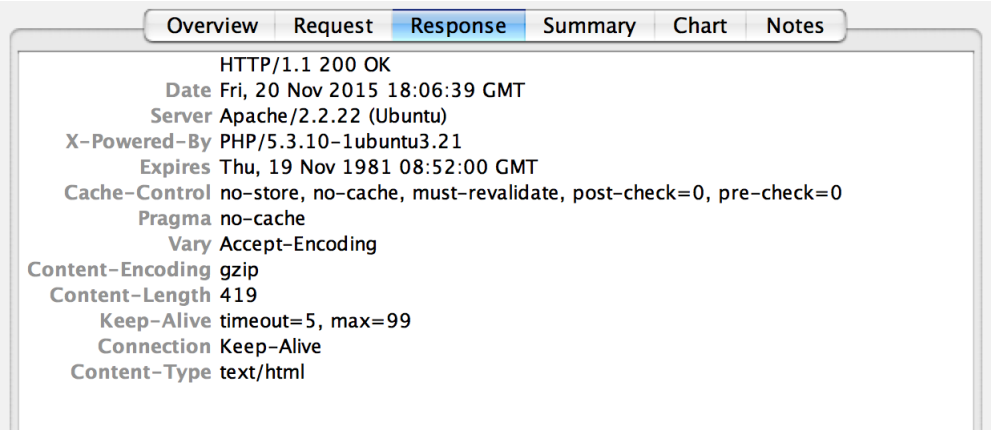
3.1.3 Test HTTP Strict Transport Security (OTG-CONFIG-007)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	The *HTTP Strict Transport Security* protocol is never used.
Discovery	We used *Charles Web Proxy* to check the HTTP response headers and the *Strict-Transport-Security* header was not found.
Likelihood	N/A
Implication	N/A
Recommendations	It would be better so transport some data via https and use the HSTS protocol.
Comparison	The same results apply for our web application.

Metric	Value
Access Vector	N
Attack Complexity	L
Privileges Required	N
User Interaction	N
Scope	U
Confidentiality Impact	L
Integrity Impact	N
Availability Impact	N


Overview
Request
Response
Summary
Chart
Notes
HTTP/1.1 200 OK
Date Fri, 20 Nov 2015 18:06:39 GMT
Server Apache/2.2.22 (Ubuntu)
X-Powered-By PHP/5.3.10-1ubuntu3.21
Expires Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma no-cache
Vary Accept-Encoding
Content-Encoding gzip
Content-Length 419
Keep-Alive timeout=5, max=99
Connection Keep-Alive
Content-Type text/html

3.1.4 Test RIA cross domain policy (OTG-CONFIG-008)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	There are no RIA applications on the system and therefore is no crossdomain.xml file provided.
Discovery	Using *wget* we tried to find a *crossdomain.xml* or *clientaccesspolicy.xml* file and couldn't find it.
Likelihood	N/A
Implication	N/A
Recommendations	N/A
Comparison	The same results applies for our web application.

Metric	Value
Access Vector	N/A
Attack Complexity	N/A
Privileges Required	N/A
User Interaction	N/A
Scope	N/A
Confidentiality Impact	N/A
Integrity Impact	N/A
Availability Impact	N/A

3.2 Identity Management Testing

3.2.1 Test Role Definitions (OTG-IDENT-001)

Secode21

Likelihood: 10
Impact: 4
Risk: 4

Secode21	
Observation	We found out that there exist two different roles in the system. There is the role of a normal customer and the role of an employee. Employees have the additional functionality to view account and transaction details of all the customers. Transactions over 10000 euro and new user registrations can be accepted by the employee.
Discovery	No special tools except a browser were needed because all the roles and their available functions are described.
Likelihood	It is very likely that people find this information.
Implication	There is no direct implication, but knowing the roles and their functionality helps with other attacks.
Recommendations	Don't describe the roles on the web page.
Comparison	Our web application provides the same roles, but the roles are not described on the web page.

Metric	Value
Access Vector	N
Attack Complexity	L
Privileges Required	L
User Interaction	N
Scope	U
Confidentiality Impact	L
Integrity Impact	L
Availability Impact	N

3.2.2 Test User Registration Process (OTG-IDENT-002)

Secode21

Likelihood: 5
Impact: 5
Risk: 5

Secode21	
Observation	Any person can register themself as an user and this registration than gets validated by an employee. One person can register multiple times and with different roles. There is no proof of the identity of a user possible. The identification requirements include the name, surname, phone number, email address and username, but only two of these can be verified.
Discovery	No special tools are needed to get this information. A browser and multiple registration tests provided the necessary results.
Likelihood	It is quite likely that this information can be retrieved by any user with minimal experience.
Implication	User could try to register multiple times and with wrong information to get access to user accounts with more permissions or to create multiple bank accounts.
Recommendations	The information passed in the registration form should be validated, especially the email address and phone number can be verified very easily. The name can be validated by hand if a customer would go to the bank and the employee would than accept his registration.
Comparison	Our web application doesn't require a phone number for the registration an the role of the user can be selected in the registration form. It doesn't make our application less secure, because the registration has still to be accepted by an employee.

Metric	Value
Access Vector	N
Attack Complexity	L
Privileges Required	N
User Interaction	N
Scope	U
Confidentiality Impact	N
Integrity Impact	N
Availability Impact	N

3.2.3 Test Account Provisioning Process (OTG-IDENT-003)

Secode21

Likelihood: N/A
Impact: N/A
Risk: N/A

Secode21	
Observation	Our observation showed us that employees can accept customer registrations and can make customer accounts to employee accounts.
Discovery	All the observations were made with the *Chrome* web browser.
Implication	If an employee account gets hacked you can make even other accounts to employees and accept new registrations.
Recommendations	N/A
Comparison	In our web application the employee doesn't make customer accounts to employee accounts but rather accepts special employee registrations. It makes no difference in the security.

Metric	Value
Access Vector	N
Attack Complexity	N/A
Privileges Required	N/A
User Interaction	N/A
Scope	N/A
Confidentiality Impact	N/A
Integrity Impact	N/A
Availability Impact	N/A

3.2.4 Testing for Account Enumeration and Guessable User Account (OTG-IDENT-004)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	We found out that the web application makes no difference between existing usernames and non existing usernames when trying to login with wrong credentials. The same html response and the same response headers are provided by the system.
Discovery	We used the *Charles Web Proxy* to analyze the web application responses.
Implication	N/A
Recommendations	N/A
Comparison	Our web application makes no difference between login tries with existing usernames and non existing ones. Both web applications aren't vulnerable here.

Metric	Value
Access Vector	N/A
Attack Complexity	N/A
Privileges Required	N/A
User Interaction	N/A
Scope	N/A
Confidentiality Impact	N/A
Integrity Impact	N/A
Availability Impact	N/A

3.2.5 Testing for Weak or unenforced username policy (OTG-IDENT-005)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	The usernames are not auto-generated and therefore there is no special structure in the usernames.
Discovery	No tool is used here. The username field in the registration form gives us all the information we need.
Implication	N/A
Recommendations	N/A
Comparison	The same applies for our web application.

Metric	Value
Access Vector	N/A
Attack Complexity	N/A
Privileges Required	N/A
User Interaction	N/A
Scope	N/A
Confidentiality Impact	N/A
Integrity Impact	N/A
Availability Impact	N/A

3.3 Authentication Testing

3.3.1 Testing for Credentials Transported over an Encrypted Channel(OTG-AUTHN-001)

3.3.2 Testing for default credentials(OTG-AUTHN-002)

Secode21

Likelihood: 10
Impact: 4
Risk: 6

Secode21	
Observation	We found out that there exists the default credentials *admin:admin*
Discovery	We were already given these credentials.
Likelihood	It is very likely that people find this information.
Implication	The attacker gain administrator access in the web application.
Recommendations	Use other credentials for testing, or delete the default ones after you launch the application.
Comparison	Our web application has a different combination of *user:password*.

Metric	Value
Access Vector	N
Attack Complexity	L
Privileges Required	N
User Interaction	N
Scope	U
Confidentiality Impact	M
Integrity Impact	M
Availability Impact	N

3.3.3 Testing for Weak lock out mechanism (OTG-AUTHN-003)

3.3.4 Testing for bypassing authentication schema (OTG-AUTHN-004)

3.3.5 Testing for Browser cache weakness (OTG-AUTHN-006)

3.3.6 Testing for Weak password policy (OTG-AUTHN-007)

3.3.7 Testing for Weaker authentication in alternative channel (OTG-AUTHN-010)

3.4 Authorization Testing

3.4.1 Testing Directory traversal/file include (OTG-AUTHZ-001)

3.4.2 Testing for bypassing authorization schema (OTG-AUTHZ-002)

3.4.3 Testing for Privilege Escalation (OTG-AUTHZ-003)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	It is not possible to escalate privileges of the user.
Discovery	We tried to change the user privilege by changing the user id after we saw that they are generated by incrementing from the first user ID.
Likelihood	N/A
Implication	N/A
Recommendations	N/A
Comparison	The same results apply for our web application.

3.4.4 Testing for Insecure Direct Object References (OTG-AUTHZ-004)

3.5 Session Management Testing

3.5.1 Testing for Bypassing Session Management Schema(OTG-SESS-001)

Secode21

Likelihood: 0
Impact: 0
Risk: 0

Secode21	
Observation	PHP session ids are used and such session ids normally can't be bypassed that means calculated easily
Discovery	We used the Chrome extension "Advanced Rest Client" to analyze the Request and the Cookies
Likelihood	NA
Implication	NA
Recommendations	NA
Comparison	Our web application also uses PHP session ids

Metric	Value
Access Vector	NA
Attack Complexity	NA
Privileges Required	NA
User Interaction	NA
Scope	NA
Confidentiality Impact	NA
Integrity Impact	NA
Availability Impact	NA

3.5.2 Testing for Cookies attributes(OTG-SESS-002)

3.5.3 Testing for Session Fixation(OTG-SESS-003)

Secode21

Likelihood: 8
Impact: 5
Risk: 3

Secode21	
Observation	The session id is not invalidated and therefore does not change after the user is authenticated. This means an attacker can force a known session id on a user. Once the user is authenticated the attacker can access also as authenticated user
Discovery	We used the Chrome extension "Advanced Rest Client" to analyze the Request and the Cookies
Likelihood	This attack is pretty easy and can also be performed by low skilled people
Implication	The attacker can do everything the user can
Recommendations	Change the session id after logging in
Comparison	Our web application has exact the same vulnerability

Metric	Value
Access Vector	N
Attack Complexity	L
Privileges Required	N
User Interaction	R
Scope	U
Confidentiality Impact	H
Integrity Impact	H
Availability Impact	N

3.5.4 Testing for Exposed Session Variables(OTG-SESS-004)

3.5.5 Testing for Cross Site Request Forgery(OTG-SESS-005)

3.5.6 Testing for logout functionality(OTG-SESS-006)

3.5.7 Test Session Timeout(OTG-SESS-007)

3.5.8 Testing for Session puzzling(OTG-SESS-008)

3.6 Data Validation Testing

3.6.1 Testing for Reflected Cross Site Scripting(OTG-INPVAL-001)

Secode21	Likelihood: 8 Impact: 5 Risk:5

Secode21	
Observation	We observed no reflected cross site scripting vulnerability.
Discovery	It seems that all parameters are stored in the database before inserting the values in the HTML.
Likelihood	N/A
Implication	N/A
Recommendations	N/A
Comparison	The same results apply for our web application.

3.6.2 Testing for Stored Cross Site Scripting(OTG-INPVAL-002)

Secode21	Likelihood: 8 Impact: 5 Risk:5

Secode21	
Observation	We observed several possibilities to execute a stored XSS attack. But not all of them could be exploited as the length of the corresponding database fields was often very restricted. We manually tried to inject JavaScript code in every input field. Therefore we used the following code, which just alerts a message.
Discovery	We inserted Javascript code in the name field on the register page. When we logged in as an employee the script was executed. There were cases when the script caused for new registered users after the script was entered to not appear.
Likelihood	This vulnerability can be easily detected, but require some JavaScript knowledge to exploit it. Therefore we estimated the likelihood to be medium.
Implication	The implications are severe as we proofed that it is possible to steal the session. As we injected the code on the admin landing-page, which implies that we were able to act as an admin and register an abitrary account.
Recommendations	Implement a input sanitation on all input fields on the backend side! Try to use whitelisting for the different datatypes and do not rely on the frontend input validation.
Comparison	

Metric	Value
Access Vector	N
Attack Complexity	M
Privileges Required	N
User Interaction	Y
Scope	U
Confidentiality Impact	M
Integrity Impact	M
Availability Impact	L

3.6.3 Testing for HTTP Verb Tampering(OTG-INPVAL-003)

3.6.4 Testing for HTTP Parameter pollution(OTG-INPVAL-004)

3.6.5 Testing for SQL Injection (OTG-INPVAL-005)

Secode21	Likelihood: 8 Impact: 5 Risk:5

Secode21	
Observation	We observed that no SQL Injection was possible.
Discovery	We tried inserting various SQL statements in the fields of using *SQL Inject Me* tool and failed.
Likelihood	N/A
Implication	N/A
Recommendations	N/A
Comparison	Our web application is also immune to SQL Injections

3.6.6 IMAP/SMTP Injection(OTG-INPVAL-011)

3.6.7 Testing for Code Injection, Testing for Local File Inclusion, Testing for Remote File Inclusion(OTG-INPVAL-012)

3.6.8 Testing for Command Injection(OTG-INPVAL-013)

3.6.9 Testing for Buffer overflow, Testing for Heap overflow, Testing for Stack overflow, Testing for Format string (OTG-INPVAL-014)

3.6.10 Testing for incubated vulnerabilities(OTG-INPVAL-015)

3.6.11 Testing for HTTP Splitting/Smuggling(OTG-INPVAL-016)

3.7 Error Handling

Team21

Team21 does not provide a lot of error messages for incorrect inputs (e.g. incorrect TAN length, wrong TAN, TAN used).
Based on the client side input validation, there are also no messages for manipulated input via proxy or by removing the validation patterns, which can lead to problems. Examples would be a malformated email which results in a not working account or a longer input then expected, which cuts off the end of the input. There are some cases when the page returns the path of the file where the error occurred.

Team3

3.8 Cryptography

3.9 Business Logic Testing

3.9.1 Test Business Logic Data Validation(OTG-BUSLOGIC-001)

3.9.2 Test Ability to Forge Requests(OTG-BUSLOGIC-002)

3.9.3 Test Integrity Checks(OTG-BUSLOGIC-003)

3.9.4 Test for Process Timing(OTG-BUSLOGIC-004)

3.9.5 Test Number of Times a Function Can be Used Limits(OTG-BUSLOGIC-005)

3.9.6 Testing for the Circumvention of Work Flows(OTG-BUSLOGIC-006)

3.9.7 Test Defenses Against Application Mis-use(OTG-BUSLOGIC-007)

3.9.8 Test Upload of Unexpected File Types(OTG-BUSLOGIC-008)

3.9.9 Test Upload of Malicious Files(OTG-BUSLOGIC-009)

3.10 Client Side Testing