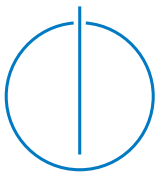
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Secure Coding - Phase 4

Whitebox Testing Report

Team 12
Alexander Lill
Lorenzo Donini
Florian Mauracher
Mahmoud Naser



Executive Summary

1 Bank System- Team 7

During our intensive testing of the Bank System application we found a fair amount of vulnerabilities, hence it is not advisable to deploy this application on a production server yet.
For starters, the server configuration is incomplete and allows to access sensitive information on the server: the private key used by the server for initiating SSL connections can be downloaded directly by anybody, hence breaking completely the assumption of encrypted connections. Also, adminer and some directories can be accessed directly via a browser.
Stack traces can be intercepted, providing knowledge about the application and what technologies are involved. Furthermore, command injection is possible when performing multiple transactions, allowing an attacker to execute arbitrary code. More specifically, when exploiting command injection, it becomes perfectly feasible to view all the source code, as well as other sensitive information like the database access credentials. Gaining access to the database gives full control over the bank data.
Moreover, we found that the logout functionality of the application is insecure, since the session data on server side is not invalidated after a user logs out. This vulnerability, combined with the fact that the cookies stored on client side are not secure, can lead to to successful session hijacking attacks.
Besides these business logic flaws, the application is also vulnerable to the renowned Heartbleed and Poodle vulnerabilities.

2 Goliath National Bank- Team 12

We found a few minor vulnerabilities that were not decisive for the overall behaviour of the application, given the high complexity required for such attacks and given the fact that they do not allow attackers to gain full access over the application (e.g. weak user registration process and application mis-use). The only vulnerability which would compromise the connection between a user and the application is given by the Poodle vulnerability.

Contents

1 Timetracking

Alexander Lill

Task	Time
Role definitions - Testing & Report	1h
User Registration Process - Testing & Report	1h
Account Provisioning Process - Testing & Report	1h
Account Enumeration and Guessable User Account - Testing & Report	1h
Weak or unenforced username policy - Testing & Report	1h
Weak security question/answer - Testing & Report	0,5h
Bypassing authorization schema - Testing & Report	2h
Cookies attributes - Testing & Report	2h
Logout functionality - Testing & Report	2h
Reflected Cross Site Scripting - Testing & Report	1h
Stored Cross Site Scripting - Testing & Report	2h
Local File Inclusion - Testing & Report	1h
Remote File Inclusion - Testing & Report	1h
Analysis of Error Codes - Testing & Report	1h
Integrity Checks - Testing & Report	0,5h
Ability to Forge Requests - Testing & Report	1h
Cross Site Request Forgery - Testing & Report	1h
Defenses Against Application Mis-use - Testing & Report	0,5h
Process Timing - Testing & Report	1h
Setting up further tools (phpcallgraph, phpmetrics, CodeClimate)	2h
Using further tools (phpcallgraph, phpmetrics, CodeClimate)	1h
Documenting and comparing tools	1h
Preparing the presentation	2h
Sum	27,5 h

Lorenzo Donini

Task	Time
Setting up report & presentation	1 h
SCS static code analysis - testing	2 h
SCS static code analysis - report	2 h
C parser - variables mapping	2 h
C parser - reverse engineering	2 h
Configuration and deploy management - testing & report	2 h
Default credentials - testing & report	1 h
Browser cache weakness - testing & report	2 h
Insecure direct object references - testing & report	1 h
Session management - testing & report	1 h
Data validation - testing & report	1 h
Command Injection - testing & report	2 h
Stack traces - testing & report	0.5 h
Business logic data validation - testing & report	2 h
Client side - testing & report	2 h
Group meeting	2 h
CVSS Scores	1 h
Executive summary & vulnerabilities overview - report	2 h
Presentation & videos	2 h
Sum	30.5 h

Florian Mauracher

Task	Time
Reverse Engineering basics	1 h
Familiarizing with Radare2	2 h
parser - Determining variable mapping	1 h
parser - Reverse engineering - Init and file parsing	2 h
parser - Reverse engineering - Transaction processing	2 h
parser - Testing the equivalent binary	1 h
Test basic functionality of Bank System	2 h
Test HTTP Methods - Testing & Report	1 h
Testing for Credentials Transported over an Encrypted Channel - Testing & Report	1 h
Testing for bypassing authentication schema - Testing & Report	1 h
Test remember password functionality - Testing & Report	1 h
Testing for weak password change or reset functionalities - Testing & Report	2 h
Testing for Privilege Escalation - Testing & Report	1 h
Testing for Session Fixation - Testing & Report	2 h
Test Session Timeout - Testing & Report	1 h
Testing for HTTP Verb Tampering - Testing & Report	0.5 h
Testing for Heap overflow - Testing & Report	1 h
Testing for Stack overflow - Testing & Report	1 h
Testing for Format string - Testing & Report	1 h
Testing for Sensitive information sent via unencrypted channels - Testing & Report	0.5 h
Test Upload of Unexpected File Types - Testing & Report	1 h
Test Cross Origin Resource Sharing - Testing & Report	0.5 h
Writing C part of reverse engineering section	1 h
Documenting and comparing tools	1 h
Sum	28.5 h

Mahmoud Naser

Task	Time
PHP Code review using RATS	1 h
PHP Code review using RIPS	1 h
Testing for Weak lock out mechanism	1 h
Testing for Weak password policy	1 h
Testing Directory traversal/file include	1 h
Testing for Bypassing Session Management Schema	1 h
Testing for SQL Injection	1 h
Testing for XML Injection	0.5 h
Testing for SSI Injection	1 h
Testing for Code Injection	1 h
Testing for incubated vulnerabilities	1 h
Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	2 h
Researching SSL Vulnerabilities	2 h
Using Tools to test SSL vulnerability	1 h
Adding SSL details in the report	1.5 h
Testing for the Circumvention of Work Flows	2 h
Testing for DOM based Cross Site Scripting	0.5 h
Testing for CSS Injection	1 h
Testing for Clickjacking	1 h
Adding Tools in Report	2 h
Sum	23.5 h

2 Vulnerabilities Overview

This section will discuss the main vulnerabilities found in the Bank Systemapplication by Team 12, it will categorize, describe and score each vulnerability according to CVSS 3.0.

2.1 Bank System

2.1.1 Directory listing and file extensions handling

- CVSS Score: *7.5*
- Likelihood: *low*
- Impact: *high*
- Risk: *high*
- Reference: OWASP OTG-CONFIG-003

Some directories can be accessed by the browser directly, as well as some sensitive files, like the database dump file and the SSL private key. This is due to the loose Apache configuration policies.

2.1.2 Guessable user account

- CVSS Score: *5.3*
- Likelihood: *low*
- Impact: *low*
- Risk: *low*
- Reference: OWASP OTG-IDENT-004

The server returns different error codes after a login attempt, depending on whether the inserted username or the password were incorrect. This makes it easier to guess if a user account exists on the database or not.

2.1.3 Default admin credentials

- CVSS Score: *5.4*
- Likelihood: *high*
- Impact: *medium*
- Risk: *medium*
- Reference: OWASP OTG-AUTHN-002

The default admin credentials can be easily brute-forced by a malicious attacker, granting them employee rights inside the Bank System application.

2.1.4 Insecure cookies

- CVSS Score: *5.4*
- Likelihood: *medium*
- Impact: *medium*
- Risk: *high*
- Reference: OWASP OTG-SESS-002

The application uses insecure cookies, which can be accessed by a malicious attacker. Since the application mainly relies on the session ID cookie for every operation, it is possible to hijack an existing session and access the client/employees accounts.

2.1.5 Cross Site Request Forgery

- CVSS Score: *4.2*
- Likelihood: *low*
- Impact: *low*
- Risk: *low*
- Reference: OWASP OTG-SESS-005

By setting a custom value inside the XSRF cookie and the value inside the GET request, it is possible to perform XSRF attacks. This is due to an incorrect comparison between the cookie and the GET parameter passed to the server.

2.1.6 Logout functionality

- CVSS Score: *4.2*
- Likelihood: *low*
- Impact: *medium*
- Risk: *medium*
- Reference: OWASP OTG-SESS-006

The adopted logout mechanism invalidates the cookie on the client side, but does not correctly invalidate the session on server side. If an attacker gained access to a session cookie before this was deleted, this could lead to a session hijacking scenario.

2.1.7 Command injection

- CVSS Score: *9.6*
- Likelihood: *high*
- Impact: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-013

It is possible to exploit the batch transaction functionality by injecting arbitrary bash commands directly into the description field, which will then be executed by the application. This also allows to view all the source code of the application, including some sensitive informations (e.g. db password).

2.1.8 Heap overflow

- CVSS Score: *3.3*
- Likelihood: *low*
- Impact: *low*
- Risk: *low*
- Reference: OWASP OTG-INPVAL-014-3

This vulnerability affects the C parser, although it cannot be directly exploited via the web interface. Due to incorrect `strncpy` input, it is possible to produce heap overflows with the correct parameters, overriding thus some memory areas.

2.1.9 Insufficient transport layer protection

- CVSS Score: *6.8*
- Likelihood: *high*
- Impact: *high*
- Risk: *high*
- Reference: OWASP OTG-CRYPST-001

Due to a missing openssl update, the application is subject to the Heartbleed vulnerability. Also, because SSLv3 is not disabled, the application is subject to the Poodle bug.

2.2 Goliath National Bank

2.2.1 Insufficient transport layer protection

- CVSS Score: *3.1*
- Likelihood: *high*
- Impact: *high*
- Risk: *high*
- Reference: OWASP OTG-CRYPST-001

Because SSLv3 is not disabled, the application is subject to the Poodle bug.

2.2.2 Defenses Against Application Mis-use

- CVSS Score: *7.1*
- Likelihood: *high*
- Impact: *low*
- Risk: *low*
- Reference: OWASP OTG-BUSLOGIC-007

No mechanisms to prevent against application mis-use are in place except the lockout-functionality (see OTG-AUTHN-003). No critical functionalities are disabled and no logs are kept.

2.2.3 File Extensions Handling for Sensitive Information

- CVSS Score: *3.1*
- Likelihood: *low*
- Impact: *low*
- Risk: *low*
- Reference: OWASP OTG-CONFIG-003

The Apache server is configured globally for the whole application, disabling directory listing but allowing direct access to all files inside the web folder. This folder mainly contains PHP, Javascript, HTML, CSS and media files (images). Among these, no sensitive information can be leaked. We found, however, that the upload folder contents are potentially accessible. So, if an attacker could brute-force the name of an uploaded file (which is entirely random) before it gets deleted by the server, he could read the contents.

2.3 Comparison

The major threats on the Bank System application come from its vulnerability to Command Injection and Heartbleed both of which have a high CVSS v3 score.

While the Goliath National Bank applications major vulnerability was its susceptibility to the SSL Poodle attack.

While both application are not production ready due to the vulnerabilities available, the Goliath National Bank application has shown more promise and is far more resilient to attacks than the Bank System application.

3 Detailed Report

3.1 Tool description

Web Testing Tools

SSL Testing Shell script

- **Used by**
  Mahmoud Naser
- **Used for**
  This tool was very useful in identifying SSL vulnerabilities and ciphers, it was a ready-to-use tool listed OWASP website, sample screen shots of the output are provided in figures Figure 3.4, Figure 3.5, Figure 3.6 and Figure 3.5
- **Useful in**
  OTG-CRYPST-001 . It detected that both applications were vulnerable to POODLE and Bank System was also vulnerable to Heartbleed.

devbug.co.uk

- **Used by**
  Alexander Lill
- **Used for**
  DevBug is a basic PHP Static Code Analysis (SCA) tool written mostly in JavaScript. The idea behind DevBug is to make basic PHP Static Code Analysis accessible online, to raise security awareness and to integrate SCA into the development process. DevBug can be used to quickly test a page of PHP that you think may have some potential vulnerabilities, to run across a piece of code you have found on Google that you are unsure of or to directly write your own code in.

  This tool was used as an additional resource for Static Code Analysis reports.

  The website shows e.g. Cross-Site-Scripting and Command Injection vulnerabilities.

- **Useful in**
  OTG-INPVAL-001 and OTG-INPVAL-002

Kiuwan

- **Used by**
  Alexander Lill
- **Used for**
  Kiuwan (see `https://www.kiuwan.com`) is a software as a service (SaaS) static program analysis multi-technology software for software analytics, quality and security measurement and management.

  This tool was used as a main source for possible vulnerabilities using Static Code Analysis.

  The tool shows defects in the categories "Maintainability", "Security", "Efficiency","Portability" and "Reliability" (see Figure 3.1) as well as estimates for the effort to fix those issues. It can be connected with e.g. Github and automatically analyzes the code after every commit and shows metrics.
- **Useful in**
  OTG-INPVAL-001 and OTG-INPVAL-002

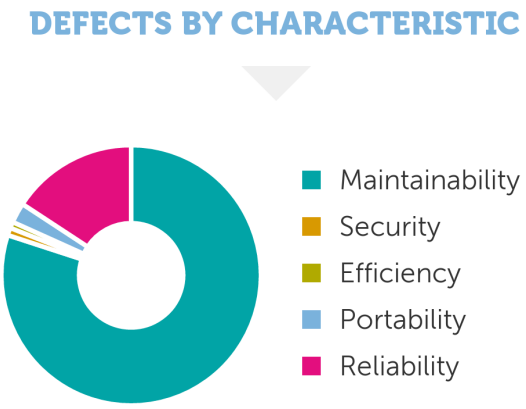


Figure 3.1: Example graph from Kiuwan

Advanced REST Client

- **Used by**
  Alexander Lill, Lorenzo Donini
- **Used for**
  This tool provides a very useful interface to send a variety of HTTP requests and shows all the attributes of the responses and sent requests. It can be used to create specific requests including customized parameters as well as checking for resulting error codes / error messages and how long it took the web server to process the request.
- **Useful in**
  OTG-ERR-002 , OTG-ERR-001 , OTG-SESS-005 and OTG-BUSLOGIC-004

Cookie Inspector

- **Used by**
  Alexander Lill
- **Used for**
  This tool provides the possibility to read, write and modify cookies. It also enables to examine the cookies which shows all their attributes. It was used to examine the cookies attributes and create customized cookies for testing and exploitation.
- **Useful in**
  OTG-SESS-002 , OTG-SESS-005 and OTG-SESS-006

RIPS

- **Used by**
  Mahmoud Naser
- **Used for**
  This tool was used to test PHP code on both Bank System and Goliath National Bank
- **Useful in**
  While most errors found using this tool were false positives, it did point out a few variables that were not immediately sanitized in the code in index.php for Bank System but after further inspection variables were sanitized and checked at a later stage.

RATS

- **Used by**
  Mahmoud Naser, Florian Mauracher
- **Used for**
  This tool was used to test PHP code on both Bank System and Goliath National Bank. It was also used to test the reverse engineered parser of Bank System and the parser source of Goliath National Bank.
- **Useful in**
  This tool while useful in detecting SQL injection, unsanitized inputs etc, it did not provide much insight due to the quality of the used code. For the c code it provided hints to problematic function and variable usages, although these were later manually identified as false positives.

Comparison

Both RIPS, RATS, devbug.co.uk and Kiuwan were used for grading the quality of the PHP code for both applications Bank System and Goliath National Bank, and while RIPS and RATS were slightly better in providing more substantial insights, both were significantly high on false positives compared to Kiuwan and devbug.co.uk, which makes using them slightly impractical for enterprise applications with huge code bases. Devbug.co.uk and Kiuwan provided very little false positives, and especially Kiuwan provided a lot of useful information about coding guidelines, maintainability issues and best practices.

C and Java Testing Tools

Java Decompiler

- **Used by**
  Lorenzo Donini
- **Used for**
  The Java decompiler standalone application was used to obtain the Java source code of the SCS application developed by Bank System, given the compiled `.jar` file. Once decompiled, all the classes were exported and analyzed with separate tools. The Java decompiler was obtained from `http://jd.benow.ca/`.

- **Useful in**
  No major vulnerabilities were found using this tool, but for a more in-depth description of how this tool was used, refer to chapter 4.

FindBugs Intellij IDEA Plugin

- **Used by**
  Lorenzo Donini

- **Used for**
  This plugin for the Intellij IDEA Java IDE (software by JetBrains) was used to statically analyze the already decompiled Java source code of the whole SCS application developed by Bank System. The tool returns a list of explicit vulnerabilities, bugs, dodgy code and more, with details about each entry and suggestions on how to fix them.

- **Useful in**
  No major vulnerabilities were found using this tool, but for a more in-depth description of how this tool was used, refer to chapter 4.

IDA Pro - Free

- **Used by**
  Lorenzo Donini, Florian Mauracher

- **Used for**
  IDA Pro was used in order to reverse engineer the C parser binaries of the Bank System application. The tool allowed us to interactively disassemble the compiled code and reverse engineer the whole parser starting from the assembly x86 instructions.
  The main functionalities of IDA Pro that we used were the program-flow graph view, the strings view and the renaming of variables and locations with humand-readable logical names.

- **Useful in**
  This tool didn't provide any direct insight on potential vulnerabilities, but allowed to retrieve the source code from the C parser binaries, needed for a further analysis. For a more in-depth description of the reverse engineering process, refer to chapter 4.

variable_mapper.py

- **Used by**
  Lorenzo Donini

- **Used for**
  We developed this script ourselves to aid us in the analysis of the assembly instructions of the C source code. This tool works in conjunction with the IDA Pro software, allowing us to automatically generate a table of variable mappings, given the assembly instructions from IDA Pro in which memory locations on the stack are declared. These locations are usually mapped to specific variables in the source code, and finding out the size of each variable/pointer on the stack can provide useful information for the whole reverse engineering process.
  This custom python script parses a txt file containing all declared stack variables and arguments, analyzes the size of each one by checking the memory location of consecutive variables and finally generates an Excel sheet in which these raw variables are mapped.
  The source txt file looks like this:
  ```
  .text:08048AD1 var_2E4 = dword ptr -2E4h
  .text:08048AD1 var_2E0 = dword ptr -2E0h
  .text:08048AD1 var_2D9 = byte ptr -2D9h
  .text:08048AD1 var_2D8 = dword ptr -2D8h
  ```
  The script is located in our project repository under `/phase4/Team7/tools/variable_mapper.py`, together with a README file and an Excel file `variables.xlsx` containing the generated variable mapping.

- **Useful in**
  This tool was useful for speeding up the reverse engineering process of the C source code, since it allowed us to have a lookup table for all variables; it also helped us to figure out the types of each variable.

Radare2

- **Used by**
  Florian Mauracher

- **Used for**
  Radare2 was used as the primary tool for the reverse engineering of the parser binary of the Bank System application.

- **Useful in**
  Like IDA Pro, this tool didn't provide any direct insight on potential vulnerabilities, but allowed the analysis of the binary, by providing a disassembly functionality with additional functionality to analyze the program flow and the used data.

Comparison

Due to the nature of Java Bytecode it is possible for the Java Decompiler to provide a decompilation of the jar file that is identical to the original program code. In comparison the two tools used to decompile the parser binary ("Ida Pro - Free" and "Radare2") could only assist in the manual decompilation process by providing a disassembly view and some further assistance to understand the supplied binary.
Ida Pro is a commercial product and probably the de facto standard for reverse engineering, unfortunately it's publicly available free version only offers a reduced feature size as it's several versions behind the current and is only available for Windows. While searching for possible alternatives to IDA we came across the Radare2.
Radare 2 is an open source reverse engineering framework available for a multitude of different operating systems including Windows, Linux and OS X. It's initial usability seems to be worse than IDA, as it only offers a command line interface and all commands consist of stringed together single letters. (e.g. `pdf @ main` stands for "print disassembly function" at address of main). But as all functions are documented by just appending a ? we decided to give it a try. After getting used to the command structure it could be effectively used to create the decompiled source code for the parser binary as described in subsection 4.2.1.

3.2 Configuration and Deploy Management Testing

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

	Bank System
Observation	We discovered that it is possible to access some files and directories which should not be accessible to the user/attacker. Specifically, we were able to get a hold of the private key used by the server for SSL encryption, the MySQL dump file and other data that should not be accessible from the network and, even less, to attackers.
Discovery	We managed to directly download several files containing sensitive information directly, as well as access some unprotected content: • `http://[HOST]/app/HTTPS/hostonly.key` • `http://[HOST]/adminer` • `http://[HOST]/dump.sql` • `http://[HOST]/Smart-Card-Simulator/src/scs/Main.java`, as well as all other Java sources. Also many other README and temporary files are directly accessible.
Likelihood	Since directory listing is disabled inside the /api and /app subfolders of the application, most of the mentioned files can be accessed only via brute-force attacks, and even then it would prove very difficult to guess the name of the directory and filename correctly. During white-box testing, this was not an issue, but in reality, finding out this weakness would require much time (low likelihood). Other names of files, however, contained in the root folder (directory listing here is not disabled), like dump.sql can be easily guessed.
Impact	Although an attacker cannot directly compromise the integrity and availability of the server, it is possible to access some very important content of the application. In particular, getting a hold of `hostonly.key` allows to start MITM attacks on any victim, since the file contains the private key of the server used for SSL encryption.
Recommendation	Relocate the .htaccess files (or the root of the web application) and place more strict rules for file/directory access, since these are too loose.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** N **A:** N **Score:** 7.5

	Goliath National Bank
Observation	The Apache server is configured globally for the whole application, disabling directory listing but allowing direct access to all files inside the web folder. This folder mainly contains PHP, Javascript, HTML, CSS and media files (images). Among these, no sensitive information can be leaked. We found, however, that the upload folder contents are potentially accessible. So, if an attacker could brute-force the name of an uploaded file (which is entirely random) before it gets deleted by the server, he could read the contents.
Discovery	We accurately analyzed the Apache configuration on the Goliath National Bank system and then tested different cases manually.
Likelihood	For an attacker to be successful, he would not only have to guess the name of a file correctly, but also request it in the time window in which the file gets parsed by the C parser, before it is deleted. This time window is very slim, making the likelihood of this attack very low.
Impact	Assuming such an attack is successful, an attacker could read the contents of a batch transaction file uploaded by a user. The confidentiality implications are not very high, considering the file doesn't contain any sensitive information such as passwords; however, since the Goliath National Bank application renames the files with the SessionID of the user who uploaded the file, having guessed the name of a file gives information about the SessionID of the user, allowing session hijacking. Given that only client sessions could be hijacked this way, the implications affect only confidentiality, and not integrity or availability, since no sensitive information is leaked whatsoever.
Recommendation	Disable some file extensions for the web application in the Apache configuration, such as txt.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** N **A:** N **Score:** 3.1

3.2.2 Test HTTP Methods (OTG-CONFIG-006)

	Bank System
Observation	We observed that the GET, POST, HEAD and OPTIONS methods are allowed over HTTPS. The methods PUT, DELETE and TRACE are not allowed, while other methods like COPY and MOVE are not implemented at all.
Discovery	This discovery was made by testing the available methods with the HTTPS client integrated in openssl. (eg. `openssl s_client -connect HOST:443`)
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	We observed that the GET, POST, HEAD and OPTIONS methods are allowed over HTTPS. The methods PUT, DELETE and TRACE are not allowed, while other methods like COPY and MOVE are not implemented at all.
Discovery	This discovery was made by testing the available methods with the HTTPS client integrated in openssl. (eg. `openssl s_client -connect HOST:443`)
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	Although HTTPS is enabled on port 443, access to the page without using HTTPS is still possible, by simply accessing the server on port 80. However, since this was left on purpose by the developers for testing purposes, we decided not to treat this as a vulnerability (no CVSS score is given).
Discovery	This information was given to us directly by the Team that developed the Bank System application. We also proved this by simply trying to access the application on port 80: this way we had normal access to the application, although without any transport security.
Likelihood	Since we are not considering this as a vulnerability, we won't analyze it in depth.
Impact	The implications of lack of an encrypted connection were already analysed in phase 2. Since we are not considering this as a vulnerability anyway, we won't analyze its implications.
Recommendation	Redirect any requests from port 80 to port 443. This is highly recommended before the application goes live.
CVSS	Secure

	Goliath National Bank
Observation	This vulnerability was fixed in phase 3 and the application is only allowing communication between the client and the server via HTTPS.
Discovery	We discovered this by simply testing to access both ports 80 and 443 on the webserver, which only allowed us to connect to the latter. A connection on port 80 was refused by the server. We also enforced this theory by checking it with nmap, getting the same result.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	Access to crossdomain.xml and clientaccesspolicy.xml was tested.
Discovery	Access to crossdomain.xml and clientaccesspolicy.xml files were tested on all available directories as well as root directory and could not gain access to any of the following directories: • . • app • font • pdfs • src • src/Api • src/Api/Model • tests • tests/Test • tests/Test/Functional • tests/Test/Unit • config • parser
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	Access to crossdomain.xml and clientaccesspolicy.xml files were tested on all available directories as well as root directory and could not gain access to any of the following directories: • . • ./project • ./project/holder • ./project/media • ./project/lib • ./project/templates • ./project/js • ./project/models • ./project/accounts • ./project/pwreset • ./project/uploads • ./project/client • ./project/style • ./project/registration • ./project/employee
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.3 Identity Management Testing

3.3.1 Test Role Definitions (OTG-IDENT-001)

	Bank System
Observation	The role definitions are implemented as specified in their report for phase 2 and are secure.
Discovery	Manual testing and checking the contraints in the source code, e.g. `api/index.php`
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	Manual testing and checking the contraints in the source code, e.g. `db.php`
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The registration process is set up for anyone to register, the process then awaits human interaction for the approval stage, this will serve an extra step of verification. Mail addresses have to be unique and can not be used by multiple users. Identities are not verified nor checked at this stage due to application limitations. No CAPTCHA or similar tests available to test if users are robots or human.
Discovery	Manual testing and checking the source code if duplicate mail addresses are possible. Here the database prevents the entry of a duplicate mail address, while the front end shows "Registration successful. Waiting for approval. You will be notified to email." even if the mail address was already in the dabase and the new account not successfully created.
Likelihood	High. It is easy to create a lot of new users as long as a syntactically valid mail address is provided.
Impact	Medium. A lot of new users could lead to performance or availability problems in the database. These registered users can not be used on the website as long as they are not approved, though.
Recommendation	The introduction of CAPTCHAs, any other check for human users or some invitation mechanism might solve this issue.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** N **A:** L **Score:** 5.3

	Goliath National Bank
Observation	See above
Discovery	Manual testing and checking the source code if duplicate mail addresses are possible. Here the database prevents the entry of a duplicate mail address and shows a message upon registration if the mail address was already used.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** N **A:** L **Score:** 5.3

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

	Bank System
Observation	Provisioning clients is an easy process with no effective mechanisms to verify or vet clients besides a manual approval process, provisioning employees is set up in a similar matter. Vulnerabilities with creating the client account have been discussed in the previous section OTG-IDENT-002 , the same applies to creating employee accounts, so a potential DOS attack is possible by creating robot accounts.
Discovery	This was found through following the given process for creating accounts.
Likelihood	High. It is easy to create a lot of new users as long as a syntactically valid mail address is provided.
Impact	Medium. A lot of new users could lead to performance or availability problems in the database. These registered users can not be used on the website as long as they are not approved, though.
Recommendation	The introduction of CAPTCHAs, any other check for human users or some invitation mechanism might solve this issue.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** N **A:** L **Score:** 5.3

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Impact	See above
Recommendation	See above
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** N **A:** L **Score:** 5.3

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

	Bank System
Observation	The user account names are the users mail addresses and therefore not easily guessable. The Bank System is vulnerable for Account Enumeration: Logins with an associated mail address and a wrong password lead to the message "Incorrect data. Wrong user or password. You have X tries more.", while a login with an unassociated mail address and any password leads to the message "Incorrect data. Wrong user or password".
Discovery	This was discovered manually in the source code (file `api/index.php` lines 178-180 and 214-245) and confirmed by manual testing.
Likelihood	The use of this vulnerability is very likely as this is a very easy way to get to know valid user accounts. Also see OTG-BUSLOGIC-004 .
Impact	This vulnerability has a low impact as the only information that can be obtained is the email of a user, which doesn't provide any information about the password or other sensitive information.
Recommendation	We recommend to always show the same error message, even though the user does not see how many tries he still has left.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** N **A:** N **Score:** 5.3

	Goliath National Bank
Observation	The Goliath National Bank application is always showing the same error message and is thus not vulnerable. We do also use the users mail address as user account names which makes them not easily guessable.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	Username has to be in valid email address format.
Discovery	This was found through trial and error and checking the according files (e.g. `index.php`, line 483).
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	This was found through trial and error and checking the according files (e.g. `js/registration.js` line 81, `registration/registration_request.php` line 29).
Likelihood	See above
Impact	See above
Recommendation	See above
CVSS	Secure

3.4 Authentication Testing

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

	Bank System
Observation	All request to the site are done over an encrypted HTTPS connection. No unencrypted messages were observed.
Discovery	After capturing the traffic during login with wireshark, it was verified that all requests made were encrypted.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	All request to the site are done over an encrypted HTTPS connection. No unencrypted messages were observed.
Discovery	After capturing the traffic during login with wireshark, it was verified that all requests made were encrypted.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The developers left a temporary administrator test account with weak credentials. It is likely that this was done specifically for testing purposes, but we decided to treat this as a vulnerability anyway, since this should be avoided when the application goes live. This flaw aside, no other accounts with weak credentials were found; also passwords are not automatically generated for users and the password policy is strict, not allowing weak passwords or empty fields for that matter.
Discovery	The default credentials left for testing purposes were: `user: test4@test.org pass: test` These were given to us directly by the developers. To prove our point, we also searched through the Rocktyou.txt password list (found at `https://wiki.skullsecurity.org/index.php?title=Passwords` and proved that 'test' is a very weak password.
Likelihood	The likelihood of cracking the password is high. Brute-forcing the username may require some more effort, but since the application returns different error codes depending whether the inserted username or password were wrong, an attacker can easily try out any possible username.
Impact	In this particular case the implications are severe, since a successful attack would grant an attacker admin rights, with consequent full access to the application.
Recommendation	It is highly recommended to change the default credentials of the application admin before deploying the application.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** U **C:** L **I:** N **A:** L **Score:** 5.4

	Goliath National Bank
Observation	The same observations made for Bank System apply. Furthermore, no weak default credentials were left at all (not even for testing purposes).
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The lockout mechanism was implemented in phase 3 on all systems as part of the requirements.
Discovery	Multiple incorrect logins were attempted, until the account was locked for 15 minutes after 6 incorrect attempts.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	Multiple incorrect logins were attempted, until the account was locked after 5 incorrect attempts, the account can only be unlocked manually by an employee account.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	No way to bypass the authentication schema was discovered. All internal pages properly redirect to the login page.
Discovery	A controller is responsible for routing the requests, after looking at it's source it could be verified, that no internal pages are accessible without authentication.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No way to bypass the authentication schema was discovered. All internal pages properly redirect to the login page.
Discovery	After inspecting the code, all php sites only meant for an authenticated user appear to check for a valid user session before further processing the request.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.4.5 Test remember password functionality (OTG-AUTHN-005)

	Bank System
Observation	Password is only transmitted during the login phase, and stored in a secure way server side.
Discovery	A strong hash function is used in combination with a salt to store the password in the database: `hash('sha256', $plainpassword.$salt);`
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	
Discovery	Password is only transmitted during the login phase, and stored in a secure way server side.
Likelihood	A strong hash function is used in combination with a salt to store the password in the database: `hash('sha512', $this->MAGIC . $password . $salt);` MAGIC is an additional token hardcoded in the php code and `sha512` offers additional entropy compared to `sha256`, so the Goliath National Bank application is slightly more secure here.
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	We observed that no Browser cache weaknesses could be exploited for this web application. This statement only holds when HTTPS is enforced (see OTG-CONFIG-007). It is important to stress though, that the Bank System application does not explicitly avoid cache weaknesses, as can be seen in any intercepted server response. This is not a problem, however, since the used Framework dynamically loads the sensitive information of the page using asynchronous Ajax requests, which are not cached by the browser.
Discovery	Browser history was tested manually, showing no issues. The browser cache was tested using the CacheViewer2 addon for Firefox: this showed us that server responses are actually cached by the browser, but these are encrypted as long as HTTPS is used. Also, given that Bank System dynamically loads pages, we only managed to restore some pages from the cache.
Likelihood	N/A
Impact	N/A
Recommendation	Although the application didn't show evident browser cache weaknesses, it is still recommended to set some additional cache-control: no-cache, no-store, must-revalidate.
CVSS	Secure

	Goliath National Bank
Observation	The Goliath National Bank application enforces cache invalidation, preventing the browser from storing any sensible information other that stylesheets and javascript files locally. Browser history also couldn't be exploited, since the application delivers the pages over HTTPS.
Discovery	Browser history was tested manually, showing no issues. Browser cache was tested using the CacheViewer2 addon for Firefox. We also analyzed some responses from the server, only to see that the application sets the following flags: `Cache-Control: private, must-revalidate`
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The weak password policy was rectified as a requirement in phase 3 on all systems.
Discovery	Registration attempts were made with weak passwords, all attempts failed.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.4.8 Testing for Weak security question/answer (OTG-AUTHN-008)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.4.9 Testing for weak password change or reset functionalities (OTG-AUTHN-009)

	Bank System
Observation	**Password change:** A logged in user is able to change his password by supplying his current password. This is considered safe. **Password reset:** A password reset can be requested by an unauthenticated user, by supplying the email address of the account on the web page. An email with a password reset link is sent to the specified address, where the password can be reset without further authentication. This enables an attacker with control over the email address to take over the account.
Discovery	The change password functions were tested manually.
Likelihood	Unlikely. Attacker needs control over the email account of the victim.
Impact	High for a single account. The attacker gains full control over the victims account.
Recommendation	Require an additional secret to reset the password and identify the user. This could be a security question or the PIN. (As long as its clearly stated that the user needs to store it safely)
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 5.0

	Goliath National Bank
Observation	**Password change:** This functionality has not been implemented / is not applicable in this application **Password reset:** A password reset can be requested by an unauthenticated user, by supplying the email address of the account on the web page. An email with a password reset link is sent to the specified address, where the password can be reset by specifying the account PIN. This second factor prevents an attacker with control over the email address to take over the account.
Discovery	The password reset function was tested manually.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The application does not provide alternative channels for authentication. The separate Smart Card Simulator Java application does not communicate with the server directly and does not perform any authentication for the user, hence this vulnerability is not applicable.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	The same observation made for the Bank System application applies.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.5 Authorization Testing

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

	Bank System
Observation	This was done through blackbox testing using ZED proxy tool and a variation of the commands bellow : • egrep -r '(include | require | fopen | readfile)'. • egrep -r '$_(POST | GET | FILE)'.
Discovery	No vulnerabilities were found for this application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	This was fixed in phase 3 as part of the requirements.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	No vulnerabilities in this area were found for the Bank System application.
Discovery	This was observed by checking all necessary functions in the file `api/index.php` which does all the checks if a request is valid or not and some manual trial and error testing.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No vulnerabilities in this area were found for the Goliath National Bank application.
Discovery	This was observed by checking all necessary functions in the according files and some manual trial and error testing.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	No vectors for privilege escalation could be identified in the Bank System application.
Discovery	After inspecting the source of the controller responsible for routing the requests, it appears that all pages requiring elevated permissions correctly verify that these elevated permissions are present.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No vectors for privilege escalation could be identified in the Goliath National Bank application.
Discovery	After inspecting the code, all php sites only meant for users with elevated permission appear to check for these permissions before further processing the request.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	All calls which involve retrieving sensitive information from the server are done via REST APIs. We analyzed all callback functions (contained inside the `/api/index.php` file) and tested the APIs manually, providing unexpected parameters. Doing this we did not manage to find any insecure direct object reference, since the server always checks whether the user issuing a request is allowed to perform a certain operation, preventing a malicious attacker to bypass the authorization schema.
Discovery	We used the Advanced REST Client extension for Chrome in order to test the available APIs (e.g. `/api/getHistory` and `/api/userinfodetails`).
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The application performs thorough checks on every page on which application-specific functionalities are called (for example lines 7-22 inside the `/gnb/project/account/download_transactions.php` file). These checks prevent users with to access data they're not allowed to access. We did not manage to find any insecure direct object reference this while analyzing the Goliath National Bank application.
Discovery	We used static PHP code analysis and also checked all server-side functionalities manually.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.6 Session Management Testing

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

	Bank System
Observation	This was done by checking the Session ID, Token Length, Session time-out mechanism and HTTPS
Discovery	No vulnerabilities was found for this application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	This was fixed in phase 3 as part of the requirements.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	We observed that the Bank System application is not setting all possible cookie attributes, see Table 3.1.
Discovery	This was discovered using manual examination of the cookies and additionally by checking the source code for cookie settings.
Likelihood	The likelihood is quite high for the exploitation of the missing "HTTP only" and "secure" attributes. The missing "secure" attribute could be related to OTG-CONFIG-007 .
Impact	This could have a quite big impact as Cookies could be transferred over an insecure connection. Additionally cookies can be read using Javascript e.g. by XSS vulnerabilities.
Recommendation	We recommend to set the missing cookie attributes accordingly.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** N **Score:** 5.4

	Goliath National Bank
Observation	The Goliath National Bank application is setting all necessary and/or useful cookie attributes, see Table 3.1.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

Attribute	Bank System	Goliath National Bank
Session	yes	yes
Host only	yes	yes
HTTP only	no	yes
Secure	no	yes
Expiry date	Not necessary because of "session" attribute	

Table 3.1: Cookies attributes

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

	Bank System
Observation	A new session is generated after the user successfully logs in.
Discovery	This was tested manually and verified by inspecting the code responsible.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	A new session is generated after the user successfully logs in.
Discovery	This was tested manually and verified by inspecting the code responsible.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	We discovered that the only session variable used during the lifetime of the application is the SessionID, which is always exchanged between client and server using a Cookie. Although this cookie is not secure, this information is always transferred over an encrypted channel, not allowing MITM attacks.
Discovery	We tried capturing packets with Wireshark, in order to prove that the session ID is not visible by simply inspecting the encrypted packets.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	Similar to the case of the Bank System application, the only session variable is the SessionID, which is transferred between client and server over an encrypted channel, not allowing MITM attacks.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	We observed a serious vulnerability in the Cross Site Request Forgery mechanism of the Bank System application. The application only compares the client-side XSRF token sent in the cookie with the client-side token sent in the header of the request. That means that this token is useless if the one in the cookie as well as the one in the header are changed simultaneously on the client. Tokens are not stored server-side.
Discovery	We checked the source code for measures against CSRF and found the not properly working mechanism (see `api/asset.php` lines 48-63). We confirmed this vulnerability with manual testing and the tools "Advanced REST Client" and "Cookie inspector".
Likelihood	Medium. It is quite likely that this vulnerability is quite easy to exploit.
Impact	Medium. Any request sent to the Bank System application website can be easily forged via Cross Site Scripting (also notice that the cookies are not secured against JavaScript manipulation, OTG-SESS-002), but neither transactions nor password change requests can be executed as either a TAN or the current password is needed for that.
Recommendation	We recommend to store the token value on the server instead and check the sent token against this one.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** N **Score:** 4.2

	Goliath National Bank
Observation	No Cross Site Request Forgery vulnerability was observed in the Goliath National Bank application.
Discovery	We checked the source code (e.g. see `accounts/new_transaction.php` lines 17-23) for the CSRF algorithm and also observed a client-server session using the Chrome Developer Tools.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	We observed that the logout can be easily circumvented by just re-creating the old cookies with the same values before the logout. That means that we can easily undo the logout by just adding a cookie locally.
Discovery	We discovered this by manually looking through the source code and confirmed this by manual testing. Listing 3.1 shows the bug that makes this exploit possible. The variable `$UserId` in line 15 is not defined in this context. leading to an SQL `UPDATE` statement with the following invalid `WHERE` clause: `where UserId='';`.
Likelihood	It is very likely that this vulnerability will be exploited as cookies can be stolen quite easy (see OTG-SESS-002) and this removes the timing aspect of stealing a given session. Logging out from the users session does not prevent any further attacks for this session.
Impact	The potential impact of this vulnerability is high as this makes stealing a user's session much easier. An attacker only needs to get access to the users PHPSESSID or the not completely secure cookie.
Recommendation	This issue can be easily resolved by using `$user->UserId` instead if the undefined `$UserId`.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** N **Score:** 4.2

Listing 3.1: Logout Bug

```
1  $app->get('/logout', function() use ($app, $db, $queries) {
2      csrf_check();
3      $user = getCurrentUserInfo();
4      session_start();
5      if (isset($_COOKIE['PHPSESSID'])) {
6          setcookie('PHPSESSID', '', time() - 3600, '/');
7      }
8      if (isset($_COOKIE['XSRF-TOKEN'])) {
9          setcookie('XSRF-TOKEN', '', time() - 3600, '/');
10     }
11     session_unset(); // Remove the $_SESSION variable information.
12     session_destroy();// Remove the server-side session information.
13     $query = $queries["updateSessionId"]();
14     $statement = getStatement($query);
15     $statement->bind_param('si', $sessionId, $UserId);
16     $sessionId=NULL;
17     $statement->execute();
18     $statement->store_result();
19 });
```

	Goliath National Bank
Observation	No vulnerabilities in the logout mechanism have been found in the Goliath National Bank application.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.6.7 Test Session Timeout (OTG-SESS-007)

	Bank System
Observation	A session timeout occurs after 20 minutes.
Discovery	The timeout found in the code was verified manually. After the timeout has occured, all furhter requests are denied.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	A session timeout occurs after approximately 24 minutes.
Discovery	The timeout is implemented by configuring the `session.gc_maxlifetime` value. This does not appear to be the most reliable approach according to the documentation, but was confirmed working in practice.
Likelihood	N/A
Impact	N/A
Recommendation	Implement a more reliable way for the session timeout.
CVSS	Secure

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

	Bank System
Observation	While analyzing the source code, we found out that the SessionID is generated entirely randomly, using the PHP APIs (this can be seen inside the login callback function contained in `/api/index.php`). It is also used only for session tracking purposes, therefore no session puzzling vulnerability can occur.
Discovery	We manually analyzed the PHP source code of the target application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The Goliath National Bank application generates the SessionID entirely randomly and uses it for session tracking purposes as well as for replacing the names of the batch files. This, however, isn't a vulnerability, since the transaction batch files cannot be accessed through the network, once uploaded, and they are removed from the system as soon as the transaction has been executed.
Discovery	We manually analyzed the PHP source code of the target application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.7 Data Validation Testing

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

	Bank System
Observation	No Reflected Cross Site Scripting vulnerabilities have been found in the Bank System application. There is only one place where user input is directly echoed onto a website, but this happens in the API (see `api/index.php` line 142) and is thus not visible.
Discovery	This was discovered by checking the results of RIPS (which only showed false positives), the results of `http://devbug.co.uk` and manually testing a few functionalities.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No Reflected Cross Site Scripting vulnerabilities have been found in the Bank System application. The only vulnerable page (`accounts/verify_transaction.php`) uses proper sanitization to prevent Reflected Cross Site Scripting.
Discovery	This was discovered by checking the results of RIPS (which only showed false positives), the results of `http://devbug.co.uk` and manually testing a few functionalities.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	No Stored Cross Site Scripting vulnerabilities have been found in the Bank System application.
Discovery	This was discovered by checking the results of RIPS (which only showed false positives), the results of `http://devbug.co.uk` and manually testing a few functionalities. The Bank System application is using the AngularJS framework (see `https://angularjs.org`) which also includes input sanitization (see `https://docs.angularjs.org/api/ngSanitize/service`).
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No Stored Cross Site Scripting vulnerabilities have been found in the Bank System application.
Discovery	This was discovered by checking the results of RIPS (which only showed false positives), the results of `http://devbug.co.uk` and manually testing a few functionalities. The Goliath National Bank application is not using any frameworks but instead implemented its own input sanitization (see `genericfunctions.php` lines 10-103.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The available HTTP methods were already documented in OTG-CONFIG-006 . As only the default methods are allowed no further testing had to be performed.
Discovery	The testing was performed in OTG-CONFIG-006 .
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The available HTTP methods were already documented in OTG-CONFIG-006 . As only the default methods are allowed no further testing had to be performed.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	Since the web application server backend is PHP/Apache, when using duplicate parameters inside a GET or POST request, the server will always parse only the last occurrence (as explained in the OWASP guide).
Discovery	We tried to duplicate parameters inside multiple requests using the Burp Proxy and discovered that only the last of the duplicate parameters was parsed.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The same observations made for the Bank System application apply.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	This was done by using the python based testing tool located in the Samurai VM.
Discovery	No vulnerabilities was found for this application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	This was fixed in phase 3 as part of the requirements.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.7.6 Testing for LDAP Injection (OTG-INPVAL-006)

	Bank System
Observation	The application doesn't use LDAP, hence no tests were possible for this vulnerability
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	The application doesn't use LDAP, hence no tests were possible for this vulnerability
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.7.7 Testing for ORM Injection (OTG-INPVAL-007)

	Bank System
Observation	The application doesn't use Object Relational Mapping, hence no tests were possible for this vulnerability
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	The application doesn't use Object Relational Mapping, hence no tests were possible for this vulnerability
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.7.8 Testing for XML Injection (OTG-INPVAL-008)

	Bank System
Observation	N/A
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	N/A
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.7.9 Testing for SSI Injection (OTG-INPVAL-009)

	Bank System
Observation	While SSI is enabled, input is sanitized and hence this is not a vulnerability on this system.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.7.10 Testing for XPath Injection (OTG-INPVAL-010)

	Bank System
Observation	The application doesn't use XPath, hence no tests were possible for this vulnerability
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	The application doesn't use XPath, hence no tests were possible for this vulnerability
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

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

	Bank System
Observation	The application uses PHPMailer to send mails through an external server, so tests were not possible for this vulnerability.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	The application uses PHPMailer to send mails through an external server, so tests were not possible for this vulnerability.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.7.12 Testing for Code Injection (OTG-INPVAL-012)

	Bank System
Observation	ASP Code is not used, and the eval() is not used, hence code injection vulnerability is not applicable.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	See above
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.7.13 Testing for Local File Inclusion (OTG-INPVAL-012-1)

	Bank System
Observation	No Local File inclusion vulnerabilities have been found in the Bank System application.
Discovery	This was discovered by checking the results of RIPS (which only showed false positives) and checking a few files manually.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.7.14 Testing for Remote File Inclusion (OTG-INPVAL-012-2)

	Bank System
Observation	No Remote File Inclusion vulnerabilities have been found in the Bank System application.
Discovery	This was discovered by checking the results of RIPS (which only showed false positives) and checking a few files manually.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	We observed that it is possible to perform command injection attacks, by exploiting the batch transaction functionality, as no input validation upon the values inserted by the user in the transaction description field is performed. Regardless of the type of used banking method (either SCS or pre-generated TANs), as long as a user inserts a valid TAN when performing a transaction, it will also be possible to inject commands to the target system through the description field.
Discovery	This was discovered while analyzing the `uploadFile` callback function in the index.php file (lines 95-154). Since the application doesn't sanitize user input (neither on client nor on server side), the `exec` function called from the server to execute the C parser can be exploited to execute arbitrary commands, by simply inserting them into the description field. These commands will simply be executed after the parser. Here is an example: `test"; ls -l; exit 1 #` **Note:** since the output of the `exec` operation is only visible on client side if the return code was different than 0, we just need to force a return code (as shown in the example above).
Likelihood	This kind of attack may require several brute-force attempts and has therefore medium likelihood, but once found out, the vulnerability is easy to exploit.
Impact	The implications of this attack are high, since the attacker is able to execute arbitrary commands on the target system. It is possible to read all the source code of the application, including database credentials. It is important to stress though, that it is not possible to modify any existing files in the target web directory, due to insufficient privileges.
Recommendation	It is highly recommended to sanitize the description value inserted by the user, or to make the user write the descriptions directly inside the batch file.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** C **C:** H **I:** N **A:** H **Score:** 9.6

	Goliath National Bank
Observation	The only point of the application where command injection was possible is the `exec` function called by the server when performing a multiple transaction (see `new_transaction_multiple.php`). After phase 3, no command injection is possible anymore, as the user input is first sanitized then verified.
Discovery	We analyzed the PHP source code to check for possible command injection attacks.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

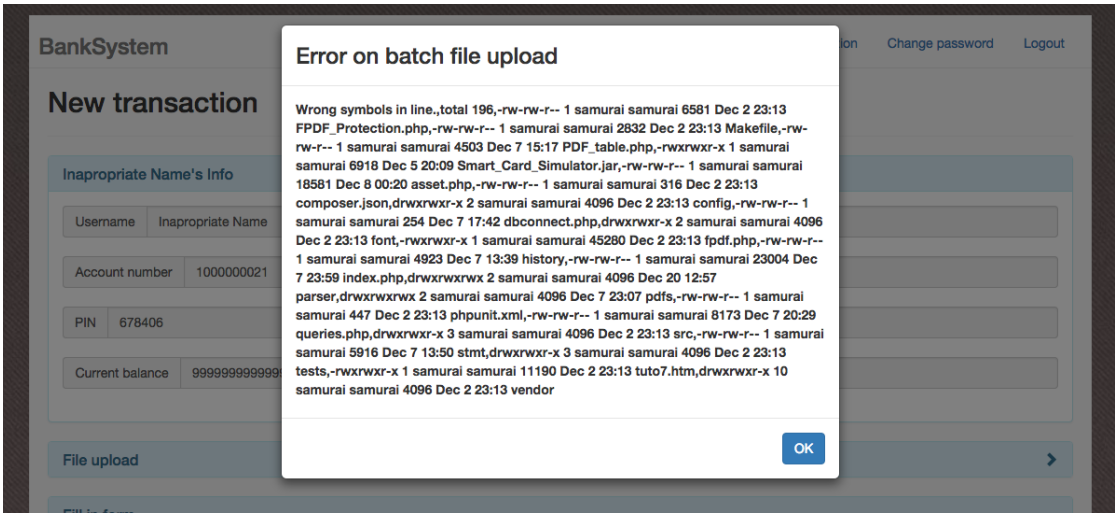


Figure 3.2: Command injection example for the Bank System application

3.7.16 Testing for Buffer overflow (OTG-INPVAL-014)

A Heap overflow was discovered and is described in OTG-INPVAL-014-3 .

3.7.17 Testing for Stack overflow (OTG-INPVAL-014-2)

	Bank System
Observation	No stack overflow was discoverd in the Bank System application.
Discovery	In the process of reverse engineering the binary we were keeping track of all accesses to variables on the stack. Afterwards we inspected the resulting source code for potential stack overflows. This was done manually and with the RATS application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No stack overflow was discoverd in the Goliath National Bank application.
Discovery	We inspected the source code for potential stack overflows. This was done manually and with the RATS application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.7.18 Testing for Heap overflow (OTG-INPVAL-014-3)

	Bank System
Observation	We discoverd a heap overflow in the Bank System application which can be triggered by invoking the application with arguments longer than 200 characters. The details of this vulnerability are described in the reverse engineering chapter 4.2.3.
Discovery	In the process of reverse engineering the binary we were keeping track of all accesses to the memory allocated on the heap. Afterwards we inspected the resulting source code for potential heap overflows. This was done manually and with the RATS application.
Likelihood	Low. The attacker needs an user account on the machine as the relevant arguments for the parser can not be modified from the web interface.
Impact	Low. The attacker already needs the permission to execute the programm with custom arguments so he gains litte additional benefits form exploiting this vulnerability.
Recommendation	Change the strncpy call as described in section 4.2.3
CVSS	**AV:** L **AC:** L **PR:** L **UI:** N **S:** U **C:** N **I:** L **A:** N **Score:** 3.3

	Goliath National Bank
Observation	No stack overflow was discoverd in the Goliath National Bank application.
Discovery	We inspected the source code for potential heap overflows. This was done manually and with the RATS application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.7.19 Testing for Format string (OTG-INPVAL-014-4)

	Bank System
Observation	No format string issues were discoverd in the Bank System application.
Discovery	In the process of reverse engineering the binary we were keeping track of all format string usages. Afterwards we inspected the resulting source code for potential issues with the format string. This was done manually and with the RATS application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No stack overflow was discoverd in the Goliath National Bank application.
Discovery	We inspected the source code for potential issues with the format sting. This was done manually and with the RATS application.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	This has also been rectified as a part of the Phase III requirements. The code was inspected for input sensitization and Manual testing was done to confirm vulnerability was fixed.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	See above
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

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

This vulnerability was not tested as we did not extend our testing to the Apache Web Server.

3.8 Error Handling

3.8.1 Analysis of Error Codes (OTG-ERR-001)

	Bank System
Observation	No vulnerabilities through Error Codes have been detected, although the one mentioned in OTG-IDENT-004 might be related because the error message is giving away that a valid mail address has been used. Please also refer to OTG-ERR-002 for error codes we found using the `api` functions, which also included slim stack traces.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No Error Code vulnerabilities have been found in the Goliath National Bank application.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.8.2 Analysis of Stack Traces (OTG-ERR-002)

	Bank System
Observation	The application uses REST calls between client and server, which can return stack traces whenever an error occurs in the Slim framework on the server. We managed to produce an example, as described in the discovery section below. These traces can only be seen in the REST response and not on the web interface. We didn't manage to produce any other stack traces otherwise, neither on the SCS nor on the Bank System web application. We discovered, however, that it is possible to produce buffer overflows on the C parser, although only locally and not through the web interface. For more information about this vulnerability, please refer to OTG-INPVAL-014 .
Discovery	We performed a REST request to the following link `https://[HOST]/api/uploadFile`,using the Advanced REST Client extension for Chrome. We only used the following as a request parameter: `X-XSRF-TOKEN: a206758...08aa5cb`. Since the request would require more/different parameters, the application on server side generates an error, which is then returned to the client side as a REST response.
Likelihood	The likelihood of an attacker testing the single REST APIs provided by the application is high.
Impact	These errors are never displayed on the client side directly, but can nevertheless be seen using a REST client, providing an attacker with information about the source code on the server, along with details on where each error code is generated (refer to the example in the picture).
Recommendation	The Slim framework produces stack traces automatically, hence the only way to avoid generating these is to check inside the PHP code if the expected variables are set. This can simply be done by calling the `isset($_GET['paramName'])` function.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** N **A:** N **Score:** 5.3

	Goliath National Bank
Observation	We did not manage to produce any stack traces, neither on the SCS application nor on the Goliath National Bank web application itself.
Discovery	We intensively tested the application in order to produce stack traces, with no results. The testing for this purpose was done manually.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

Figure 3.3: Example of a Slim application error

3.9 Cryptography

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

```
--> Testing protocols (via sockets except TLS 1.2 and SPDY/NPN)

SSLv2      not offered (OK)
SSLv3      offered (NOT ok)
TLS 1      offered
TLS 1.1    offered
TLS 1.2    offered (OK)
SPDY/NPN   not offered

--> Testing ~standard cipher lists

Null Ciphers                 not offered (OK)
Anonymous NULL Ciphers       not offered (OK)
Anonymous DH Ciphers         not offered (OK)
40 Bit encryption            not offered (OK)
56 Bit encryption            Local problem: No 56 Bit encryption configured in /usr/bin/openssl
Export Ciphers (general)     not offered (OK)
Low (<=64 Bit)               not offered (OK)
DES Ciphers                  not offered (OK)
Medium grade encryption      offered (NOT ok)
Triple DES Ciphers           offered (NOT ok)
High grade encryption        offered (OK)
```

Figure 3.4: SSL Ciphers used by Bank System application

```
--> Testing vulnerabilities

Heartbleed (CVE-2014-0160)                VULNERABLE (NOT ok)
CCS (CVE-2014-0224)                       VULNERABLE (NOT ok)
Secure Renegotiation (CVE-2009-3555)      not vulnerable (OK)
Secure Client-Initiated Renegotiation     not vulnerable (OK)
CRIME, TLS (CVE-2012-4929)                not vulnerable (OK)
BREACH (CVE-2013-3587)                    NOT ok: uses gzip HTTP compression (only "/" tested)
POODLE, SSL (CVE-2014-3566)               VULNERABLE (NOT ok), uses SSLv3+CBC (check TLS_FALLBACK_SCSV mitigation below)
TLS_FALLBACK_SCSV (RFC 7507), experim.    Local problem: /usr/bin/openssl lacks TLS_FALLBACK_SCSV support
FREAK (CVE-2015-0204)                     not vulnerable (OK) (tested with 4/9 ciphers)
LOGJAM (CVE-2015-4000), experimental      not vulnerable (OK) (tested w/ 2/4 ciphers only!), common primes not checked.
BEAST (CVE-2011-3389)                     SSL3: EDH-RSA-DES-CBC3-SHA DES-CBC3-SHA
                                          TLS1: EDH-RSA-DES-CBC3-SHA DES-CBC3-SHA
                                          -- but also supports higher protocols (possible mitigation): TLSv1.1 TLSv1.2
RC4 (CVE-2013-2566, CVE-2015-2808)        VULNERABLE (NOT ok): RC4-SHA
```

Figure 3.5: SSL Vulnerabilities used by Bank System application

	Bank System
Observation	This was tested using a script listed on the OWASP website TestSSL, this ran a list of tests on SSL, (see Figure 3.4 and Figure 3.5).
Discovery	This application reported the following main vulnerabilities: • Vulnerable to Heartbleed • Vulnerable to CCS • Vulnerable to POODLE • Vulnerable to RC4 #1 • Vulnerable to RC4 #2
Likelihood	**Heartbleed** Heartbleed while talked about it often when mentioning SSL, still requires some technical knowledge, which is readily available online with instructions and hence the likelihood is high. **CCS** In order to exploit CCS it requires the attacker to intercept and alter network traffic in real time. This reduces the risk and likelihood of this vulnerability. **POODLE** POODLE exploit tools are already in development, however this attack only works with SSL 3.0 which is being phased out at the moment. **RC4** The RC4 algorithm has many single-byte biases, making it susceptible to statistical attacks, which require some technical knowledge and hence the likelihood is not high.
Impact	With the exception on Heartbleed which allows the attackers to read server memory, the rest of the attacks impact lies in eavesdropping on encrypted packets.
Recommendation	• Install system security updates especially Apache and SSL • Disable SSLv3, Medium grade encryption, Triple DES etc.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** N **S:** C **C:** H **I:** N **A:** N **Score:** 6.8

	Goliath National Bank
Observation	See above, (see Figure 3.6 and Figure 3.7).
Discovery	This application reported the following main vulnerabilities: • Vulnerable to POODLE • Vulnerable to RC4 #1 • Vulnerable to RC4 #2
Likelihood	**POODLE** POODLE exploit tools are already in development, however this attack only works with SSL 3.0 which is being phased out at the moment. **RC4** The RC4 algorithm has many single-byte biases, making it susceptible to statistical attacks, which require some technical knowledge and hence the likelihood is not high.
Impact	Both attacks lead to eavesdropping on encrypted packets.
Recommendation	To fix the vulnerabilities above the following is recommended: • Disable SSLv3, Medium grade encryption, Triple DES and other weak ciphers
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** N **A:** N **Score:** 3.1

```
--> Testing protocols (via sockets except TLS 1.2 and SPDY/NPN)

 SSLv2      not offered (OK)
 SSLv3      offered (NOT ok)
 TLS 1      offered
 TLS 1.1    offered
 TLS 1.2    offered (OK)
 SPDY/NPN   not offered

--> Testing ~standard cipher lists

 Null Ciphers                 not offered (OK)
 Anonymous NULL Ciphers       not offered (OK)
 Anonymous DH Ciphers         not offered (OK)
 40 Bit encryption            not offered (OK)
 56 Bit encryption            Local problem: No 56 Bit encryption configured in /usr/bin/openssl
 Export Ciphers (general)     not offered (OK)
 Low (<=64 Bit)               not offered (OK)
 DES Ciphers                  not offered (OK)
 Medium grade encryption      offered (NOT ok)
 Triple DES Ciphers           offered (NOT ok)
 High grade encryption        offered (OK)
```

Figure 3.6: SSL Ciphers used by Goliath National Bank application

```
--> Testing vulnerabilities

 Heartbleed (CVE-2014-0160)                not vulnerable (OK) (timed out)
 CCS (CVE-2014-0224)                       not vulnerable (OK)
 Secure Renegotiation (CVE-2009-3555)      not vulnerable (OK)
 Secure Client-Initiated Renegotiation     not vulnerable (OK)
 CRIME, TLS (CVE-2012-4929)                not vulnerable (OK)
 BREACH (CVE-2013-3587)                    NOT ok: uses gzip HTTP compression (only "/" tested)
 POODLE, SSL (CVE-2014-3566)               VULNERABLE (NOT ok), uses SSLv3+CBC (check TLS_FALLBACK_SCSV mitigation below)
 TLS_FALLBACK_SCSV (RFC 7507), experim.    Downgrade attack prevention supported (OK)
 FREAK (CVE-2015-0204)                     not vulnerable (OK) (tested with 4/9 ciphers)
 LOGJAM (CVE-2015-4000), experimental      not vulnerable (OK) (tested w/ 2/4 ciphers only!), common primes not checked.
 BEAST (CVE-2011-3389)                     SSL3: ECDHE-RSA-DES-CBC3-SHA EDH-RSA-DES-CBC3-SHA
                                                 DES-CBC3-SHA
                                           TLS1: ECDHE-RSA-DES-CBC3-SHA EDH-RSA-DES-CBC3-SHA
                                                 DES-CBC3-SHA
                                           -- but also supports higher protocols (possible mitigation): TLSv1.1 TLSv1.2
 RC4 (CVE-2013-2566, CVE-2015-2808)        VULNERABLE (NOT ok): ECDHE-RSA-RC4-SHA RC4-SHA
```

Figure 3.7: SSL Vulnerabilities used by Goliath National Bank application

3.9.2 Testing for Padding Oracle (OTG-CRYPST-002)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.9.3 Testing for Sensitive information sent via unencrypted channels (OTG-CRYPST-003)

	Bank System
Observation	All request to the site are done over an encrypted HTTPS connection. No unencrypted messages were observed.
Discovery	After capturing the traffic of normal application use with wireshark, it was verified that all requests made were encrypted.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	All request to the site are done over an encrypted HTTPS connection. No unencrypted messages were observed.
Discovery	After capturing the traffic of normal application use with wireshark, it was verified that all requests made were encrypted.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.10 Business Logic Testing

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

	Bank System
Observation	While testing the whole application, we discovered that only valid data is accepted, not allowing to exploit the application in any way by inserting invalid data.
Discovery	We statically analyzed the PHP source code using RIPS, RATS and later on went through the code manually, in order to find additional vulnerabilities not detected by the previously mentioned tools. RIPS and RATS also produced some false positives, like the `$tan` field inside the multiple transaction callback function (/api/index.php line 105): in this example the variable was marked as a vulnerability, although the data is validated by the application at a later moment in time.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	While testing the whole application, we discovered that only valid data is accepted, not allowing to exploit the application in any way by inserting invalid data. As described in chapter 4 (SCS analysis) however, after a specific point in time, user input will no longer be validated correctly, because of a timestamp overflow bug in the Java code. This cannot be considered a vulnerability, but will pose a problem for clients in the future (in 22 years time).
Discovery	We statically analyzed the PHP source code using RIPS, RATS and later on went through the code manually, in order to find additional vulnerabilities not detected by the previously mentioned tools. Also, RIPS and RATS mainly produced false positives.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	No vulnerabilities in this area were found for the Bank System application.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No vulnerabilities in this area were found for the Goliath National Bank application.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.10.3 Test Integrity Checks (OTG-BUSLOGIC-003)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

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

	Bank System
Observation	We observed that the Bank System application has a process timing vulnerability in the login function. The login page would reply within 50ms if the mail address was valid and within around 20ms if the mail address was not valid.
Discovery	We observed that by looking at the source code for the login (see `api/index.php` lines 168-304) and confirming our observations by black box testing.
Likelihood	It is quite likely that this vulnerability is used for finding valid mail addresses, although in this particular case another vulnerability (see OTG-IDENT-004) is much easier to exploit for this attack.
Impact	N/A
Recommendation	N/A
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** N **A:** N **Score:** 5.3

	Goliath National Bank
Observation	No Process Timing vulnerability was found in the Goliath National Bank application.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	While reviewing the project documentation and testing the single functionalities we did not find any possible application misuse/abuse cases that would allow a user to execute a function more than the allowable number of times.
Discovery	We performed manual tests on all application features.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The same observations made for the Bank System application hold.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	The follow cases were tested based on the report submittid from team 7 in Phase II • Customer ability to access employee portal. • Customer approving his own transaction. • Customer accessing account before account being approved. • Customer bypassing approval for large transaction. By inspecting the code, proper checks have been put in place that disallow circumvention in the above cases.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	
Discovery	Vulnerabilities in this seciton were rectified as part of the requirements for phase III.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

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

	Bank System
Observation	No mechanisms to prevent against application mis-use are in place except the lockout-functionality (see OTG-AUTHN-003). No critical functionalities are disabled and no logs are kept.
Discovery	These observations were made after reading the source code and testing the accuracy of the observations with some manual testing.
Likelihood	The likelihood of exploiting this vulnerability is high because all standard fuzzying tools are able to exploit this vulnerability, even making it possible for so-called "script-kiddies" to try their luck.
Impact	Given the other observations about the Bank System application the impact of this vulnerability is not really high as this will probably not lead to a security breach.
Recommendation	We recommend to implement a simple way to detect malicious requests and prevent further requests for some time after a certain treshold is reached.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 7.3

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Impact	See above
Recommendation	See above
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 7.3

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

	Bank System
Observation	Only files with the MIME-type `text/plain` are allowed. The file is renamed on the server side to a random string with a `.txt` file extension. The file is deleted directly after processing.
Discovery	We obtained the allowed file types by inspecting the code and manually verifying the results by testing the file upload afterwards.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The files are renamed on upload to the sessionid without extention. The c parser ignores files with only invalid transaction details afterwards. The file is deleted directly after processing.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	Additionally filter for mime type and/or file extention before accepting the file upload.
CVSS	Secure

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

	Bank System
Observation	We discovered that it is possible to upload malicious files using the batch transaction functionality: these files won't be recognised by the target system as malwares, but these files cannot bring any harm to the application. This is because the uploaded files are automatically renamed and removed from the system after executing the C parser. Since the C parser reads the the uploaded files as simple text files, nothing bad can happen.
Discovery	We tried uploading test malwares found in the Internet, as well as custom PHP/bash scripts. Since the application only allows text extensions, so we had to rename them first, before uploading them. All files were accepted by the server, but the transactions simply failed as a result.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	The same observations made for the Bank System application hold.
Discovery	See above
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.11 Client Side Testing

3.11.1 Testing for DOM based Cross Site Scripting (OTG-CLIENT-001)

	Bank System
Observation	see OTG-INPVAL-001
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.11.2 Testing for JavaScript Execution (OTG-CLIENT-002)

Please refer to OTG-INPVAL-001 and OTG-INPVAL-002 .

3.11.3 Testing for HTML Injection (OTG-CLIENT-003)

	Bank System
Observation	We discovered that it is not possible to perform HTML injection attacks on the Bank System application, although some user input isn't properly sanitized by the server (e.g. Description field in a transaction). This does not pose a threat, however, since the application is based on the AngularJS framework, which sanitizes the user input automatically on the client (see image below). Refer to OTG-INPVAL-002 for more details.
Discovery	We manually analyzed the client-side code of the Bank System application, looking for `.innerHTML` and `document.write` potential vulnerabilities, not finding any.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	We didn't find any HTML injection vulnerability, since all user input is sanitized by the server for each GET/POST request.
Discovery	We manually analyzed the client-side and server-side code, looking for potential HTML injection vulnerabilities.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

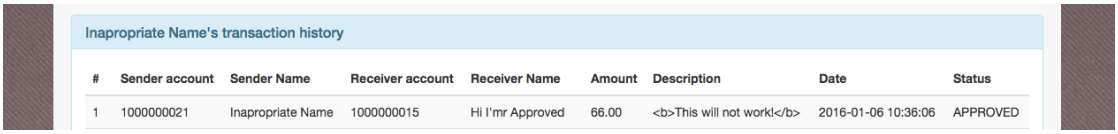


Figure 3.8: HTML injection attempt inside a description field

3.11.4 Testing for Client Side URL Redirect (OTG-CLIENT-004)

	Bank System
Observation	While analyzing the Javascript code, we did not find any URL redirection vulnerabilities. The Bank System application exploits the AngularJS framework and moves between pages using static strings, never using any unsanitized user input as a target URL. The backend also never performs redirection using a GET/POST parameter as the target URL.
Discovery	We accurately inspected all the client-side code and the PHP server-side code where URL redirections take place.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	While testing the Javascript code, we did not find any URL redirection vulnerabilities. Whenever a page is redirected using the `window.location` object, a static string is directly assigned to it, never using any unsanitized user input as a variable. The backend also never performs redirection using a GET/POST parameter as the target URL.
Discovery	We accurately inspected all the client-side code and the PHP server-side code where URL redirections take place. In the latter case, the Goliath National Bank application performs redirections only exploiting the `resource_mappings.php` file and not redirecting directly to a URL contained in a parameter.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.11.5 Testing for CSS Injection (OTG-CLIENT-005)

	Bank System
Observation	This vulnerability is not available as no code is injectable in the CSS files or styles in HTML or PHP files.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.11.6 Testing for Client Side Resource Manipulation (OTG-CLIENT-006)

	Bank System
Observation	No Client Side Resource Manipulation vulnerability was found in the Bank System application.
Discovery	This was observed by manually looking through the source code as none of the used tools covered this area.
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

	Goliath National Bank
Observation	No Client Side Resource Manipulation vulnerability was found in the Goliath National Bank application.
Discovery	See above
Likelihood	See above
Impact	See above
Recommendation	See above
CVSS	Secure

3.11.7 Test Cross Origin Resource Sharing (OTG-CLIENT-007)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.11.8 Testing for Cross Site Flashing (OTG-CLIENT-008)

	Bank System
Observation	This vulnerability doesn't apply for the tested applications, since Flash is not used.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This vulnerability doesn't apply for the tested applications, since Flash is not used.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.11.9 Testing for Clickjacking (OTG-CLIENT-009)

	Bank System
Observation	This vulnerability was meant to be fixed as part of the Phase III requirements.
Discovery	The Bank System Application has no protection (X-Frame-Options header) against Click-Jacking attacks. This vulnerability was found using w3af.
Likelihood	This is a relatively easy attack, not much technical knowledge is needed.
Impact	Depending on the skill of the attacker has, but can potentially take control of certain aspects of the victims computer such as camera and microphone.
Recommendation	• Sending the proper X-Frame-Options HTTP response headers that instruct the browser to not allow framing from other domains • Employing defensive code in the UI to ensure that the current frame is the most top level window
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** C **C:** L **I:** L **A:** N **Score:** 4.7

	Goliath National Bank
Observation	This vulnerability was fixed as part of the phase 3 requirements.
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	Secure

3.11.10 Testing WebSockets (OTG-CLIENT-010)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.11.11 Test Web Messaging (OTG-CLIENT-011)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

3.11.12 Test Local Storage (OTG-CLIENT-012)

	Bank System
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

	Goliath National Bank
Observation	This functionality has not been implemented / is not applicable in this application
Discovery	N/A
Likelihood	N/A
Impact	N/A
Recommendation	N/A
CVSS	N/A

4 Reverse Enginering

In this chapter we will present the results obtained and the observations made while reverse engineering the code of the Bank System application. We will focus on the Smart Card Simulator Java application as well as the C batch transaction parser in order to analyze potential vulnerabilities, reverse engineer the TAN generation algorithm and ultimately make a direct comparison with the Goliath National Bank application.

4.1 Smart Card Simulator

The SCS application provided by Bank System requires the user to input his PIN, then allows to either insert the details (destination account, amount) for a single transaction or load a batch file for a multiple transaction.
After having decompiled the code using the JavaDecompiler tool, we run the FindBugs plugin for IntelliJ IDEA on the source code of both the Bank System application and the GNB application.

4.1.1 FindBugs

The Bank System static code analysis didn't return any specific vulnerabilities, as visible in 4.1.
One performance issue arised because of String concatenation inside a loop, which isn't really significative. There were also some dodgy code warnings regarding the usage of uninitialized variables (graphical components), which are irrelevant for this specific case, since Bank System uses a JavaFX application that binds the variables automatically thanks to the .fxml files. The warnings regarding unwritten variables are also uninmportant, for the same reason.

The static code analysis of the Goliath National Bank application on the other hand returned significantly more issues, most of which derived from the IntelliJ UI Designer APIs. The exact results are visible in 4.2.
Among the relevant warnings for the GNB implementation were:

- internalization (reliance on default encoding) could pose a problem in the case of

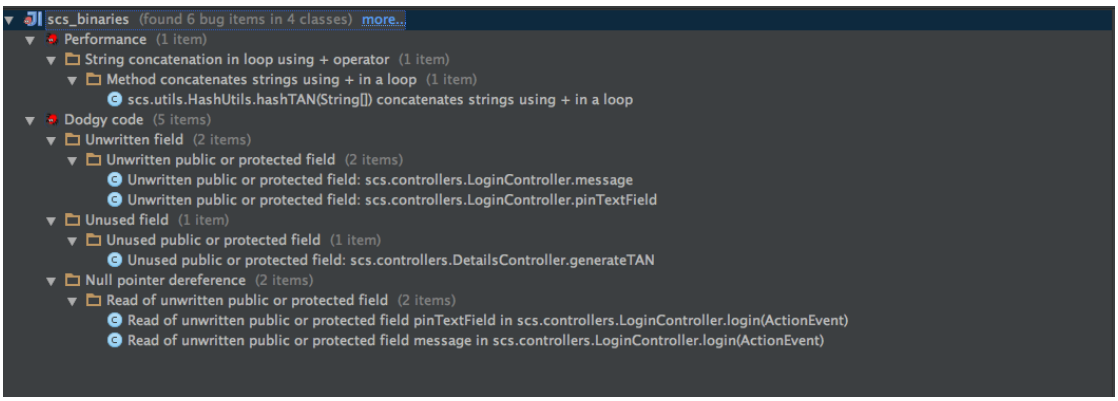


Figure 4.1: Analysis results for Bank System

non-default encodings on different platforms, hence it would be recommended to force a specific Charset when converting from bytes to Strings and vice-versa;

- unclosed file stream at the end of the `parseFile` method inside the Presenter class. This could potentially lead in a file descriptor leak, but is easy to fix;
- exception is caught when exception is not thrown inside the `parseFile` method. Although this practice may hide other bugs that would otherwise throw a RuntimeException, this warning is not relevant in this particular case, since only file parsing exceptions might occur.

All other bugs were found in the library used for designing the GUI, hence cannot be fixed. Seeing as there are some malicious code vulnerabilities in it, it may be advisable to use a different graphic library.

Not having found severe vulnerabilities through static code analysis, we did some further manual analysis on both applications.

While analyzing the Bank System code we noticed that the application does not perform any user input validation, i.e. a user could insert arbitrary strings inside the PIN, destination and amount fields. This, however, does not pose any problems, as a TAN generated through invalid data will simply be rejected by the server.

The GNB application, on the other hand, performs strict checks upon all user inputs, preventing any user from inserting invalid data. Pin, account number and transfer amount formats are already checked by the SCS application, discouraging the user from entering invalid data inside the web application at a later moment in time.

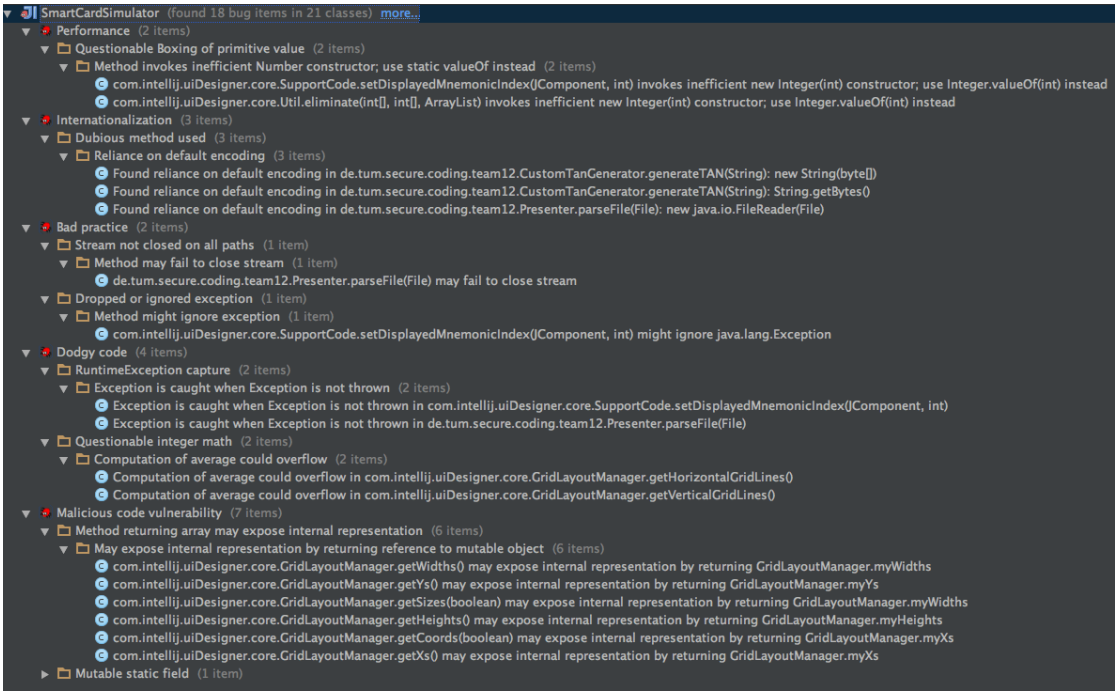


Figure 4.2: Analysis results for Goliath National Bank

4.1.2 TAN generation algorithm

What follows is an analysis of the algorithm used by the Bank System application for generating TAN codes on the Java-based SCS application. This TAN then needs to be recognised by the web application without any interaction between the two.
The algorithm can be found inside the `HashUtils.java` file and works as follows:

1. Pin, destination account and amount to be transfered are concatenated inside a String in case of single transactions. In case of a batch transaction, the Pin and the whole content of the batch file are concatenated inside a String;
2. A 5-digit nonce is randomly generated (i.e. an Integer between 10000 and 99999);
3. The nonce is appended to the String obtained in the first step;
4. The whole string is hashed using the SHA-256 function;
5. The result of the hash operation (bytes) is converted into a BigInteger;
6. The Hexadecimal String representation of the BigInteger is generated;
7. As long as the length of the generated String is less than 64 characters, zeros are prepended;
8. The first 10 characters of the String are taken and the previously generated nonce (5 characters) is appended;
9. The resulting String has now a length of 15 characters and is returned to the user as the TAN.

The generation of the 5-digit nonce provides randomness, hence two consecutives TANs will always look different, even though the input (i.e. PIN, account and amount) may be the same. The adopted solution also does not allow replay attacks: even though the nonce is random, making it theoretically possible to always use the same TAN for two identical transactions, this cannot happen since the server keeps track of the used TANs. This mechanism is visible inside the `/api/index.php` file (lines 130-140) and inside the `/api/asset.php` file (lines 511-524).
Since the last 5 characters of each TAN, generated by the SCS, will always be numbers (i.e. the 5-digit nonce), it is easy for an attacker to guess the used algorithm. This however, doesn't prove the solution to be insecure, as a potential attacker can obtain the data inserted by the victim only by brute-forcing all possible combinations of Pin, account and amount. This is because the used function is a one-way hash function.
This solution hence proves to be, in theory, secure against external attacks. Nonetheless, the application could incur into the following problems:

- The random nonce could occur more than once for two identical transactions made by the same client. The resulting TAN would then be rejected by the server, since it has already been marked as used;
- Although very improbable, there could be hash collisions between two completely different transactions, leading to the same problem mentioned above (the nonce needs to be the same in this case as well);

The GNB Smart Card Simulator uses an approach that is very similar to the one we just analyzed for the Bank System application: the details of the transaction are hashed along with the Pin of the user and a 5 characters long timestamp (in base64). To avoid communicating with the server directly, the timestamp then replaces the last 5 characters of the resulting TAN, in order to allow the server to recompute the hash. Although it is harder to detect at first glimpse, since the timestamp has the same base64 format as the rest of the TAN, the used algorithm can still be guessed after careful observation. The principle, however, remains the same as the one previously described for the System Bank. The main difference is that no collision is possible in the GNB application, since strictly increasing timestamps are used instead of random nonces. Even in case of a hash collision, the resulting TAN would still be different than any other used before, hence unique.
The GNB application presents a serious limitation though: since the timestamp used is only 5 characters long and is second-based (for better precision), the maximum timespan that can be covered in total is of $64\hat{5}$ seconds $\bar{3}4$ years. This means that, in our case, the generated timestamps will overflow in approximately 22 years. After that date, the server will not accept further TANs anymore, since the timestamp will be lower than previously inserted ones. This bug is very similar to the year 2038 bug and should be fixed (i.e. handled differently) for long-term support.

4.2 C Parser

4.2.1 Reverse engineering

As the C binary for the batch processing was not included in the SourceAndBinary archive submitted by the other team, we obtained the binary directly from the VM by logging in with the supplied credentials. The binary file we obtained was located at `/var/www/api/parser/a.out` on the VM and has a sha256 sum of `ab53c91c387118a4d21f54026829d60d05786a131455c90a96c5925535054a15`.
By using the radare2 reverse engineering framework we obtained a disassembly of all functions of the binary and a list of all symbols used in the binary. Based on this information and with the visual support of the control flow graph of the main function (see excerpt in Figure 4.3) we created an equivalent program included in the deliverables.

4.2.2 Extracted credentials

As expected the C code contained the credentials required for a connection to the database as it is required in order to perform the actual transactions.

- **Database host** localhost
- **Database user** root
- **Database pass** crazypassword
- **Database name** Banking

No further credentials or hidden keys were discovered. The SQL statements used in the binary can be viewed in the submitted reverse engineered c source code.

4.2.3 Remarks on the C parser

The whole program logic resides in the main function

- This leads to a single block of procedural instructions with over 750 lines of assembler code in the disassembly and over 250 lines in the reverse engineered c code.
- Functions of this size make maintenance and testing harder and prohibit code reuse.

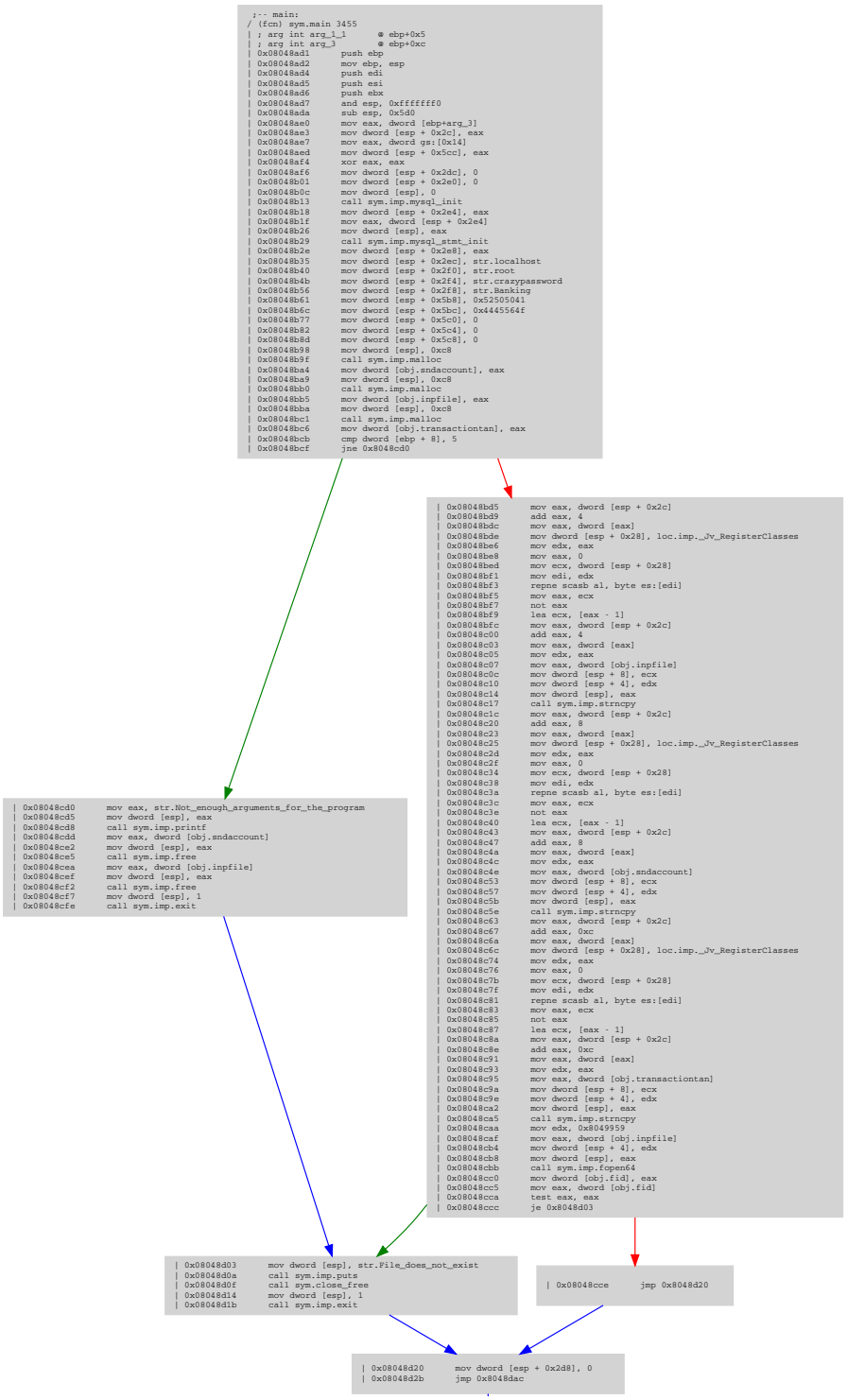


Figure 4.3: First part of the control flow graph

- The increased maintenance difficulty is visible on multiple occasions throughout the program. For example only 2 of 3 malloced strings are freed on error, probably missed as it seems like the third string was added to a later date and the function size makes it harder to figure out all data dependencies.
- Separating the different parts of the program in individual functions with clearly defined interfaces is recommended.

Heap overflow

- The variables `sndaccount, inpfile and transactiontan` are initialized with a pointer to 200 bytes of allocated heap memory each.
  `sndaccount = malloc(200);`
- This size limitation is not correctly enforced when copying data to these areas. The size of the data copied is only limited to the size of the string being copied instead of the size of the destination buffer.
- `strncpy(dst, src, strlen(src))` is used instead of
  `strncpy(dst, src, 200)`
- This results in a heap overflow by specifying a parameter longer than 200 chars to the script. The consequences of this overflow are documented in OTG-INPVAL-014-3 .

Error handling

- On multiple occasions return values are not checked for errors. For example none of the malloc calls check if the memory allocation was successful.
- The MySQL connection is not properly closed in case of error. Although this happens shortly before the end of the program, this might result in an increase amount of concurrent connections to the database before the connections are cleaned of automatically.