DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Secure Coding - Phase 2

Blackbox Testing Report

Team 12
Alexander Lill
Lorenzo Donini
Florian Mauracher
Mahmoud Naser

Executive Summary

1 DogeBank- Team 27

During our intensive testing of the DogeBank application, we found many severe vulnerabilities, as well as bugs, that could easily be exploited to compromise the entire application. Under no circumstances should this web application be used productively!

For starters, directory listing is enabled and allows to browse the structure of the web application, providing knowledge about the application flow and some functionalities. Several checks are not performed, therefore it is possible to access some pages as an unauthenticated user and perform operations that should only be allowed to employees (e.g. approving registrations/transactions). This also holds for users without sufficient role privileges. Even without all these vulnerabilities, session hijacking would still be a viable option to obtain employee privileges. SQL injection and XSS are also possible on almost all pages, giving an attacker plenty of possibilities to cause damage to both customers and employees.

Furthermore any registered user may potentially upload arbitrary malicious code to the server by exploiting the batch transaction functionality. Since no checks are performed whatsoever on the extension of the uploaded file, it is perfectly feasible to upload scripts that can later be executed from just anyone. Although this issue is easy to fix, it may give an attacker full control over the web application as well as over part of the system it runs on. All database and email passwords are also accessible this way. Additionally, the file upload functionality is vulnerable to direct code injection, which will result in allowing an attacker to execute arbitrary code.

The business logic also seems to be somewhat broken, as a customer can increase his/her balance arbitrarily, by exploiting multiple flaws in the transaction functionalities. Moreover TANs are not entirely random and will be generated only once per user, allowing endless transactions after all the codes have been used up.

2 Goliath National Bank- Team 12

We found several vulnerabilities and bugs, most of them having minor impact on the web application as a whole, but a few of them were still pretty severe, as they allowed an attacker to gain root access or simply hijack sessions and gaining access to other user accounts (both clients and employees).

Contents

1 Timetracking

Alexander Lill

Task	Time
Setting up VM environment	2 h
Setting up hacking tools	1 h
Testing for SQL Injection	2 h
Writing for SQL Injection	2 h
Testing for XML Injection	1 h
Testing for Code Injection	2 h
Testing for Local File Inclusion	1 h
Testing for Remote File Inclusion	1 h
Analysis of Error Codes	1 h
Testing Business Logic Data Validation	2 h
Writing Business Logic Data Validation	2 h
Testing Ability to Forge Requests	1 h
Testing for Process Timing	1 h
Writing for Process Timing	1 h
Looking into the CVSS Score Calculation	1 h
Implementing the CVSS Score Calculation in LaTeX	2 h
Fixing problems in LaTeX for the CVSS Score Calculation	2 h
Improving the CVSS Score Calculation	2 h
Publishing the CVSS Score Calculation	1 h
Looking into reveal.js for presentation	1 h
Creating the first reveal.js presentation draft	2 h
Improving the reveal.js presentation	2 h
Rating the CVSS Scores	1 h
Sum	34 h

Lorenzo Donini

Task	Time
Setting up hacking tools & environment	2 h
Group meeting	2 h
Building custom scripts for testing	2 h
Stress testing DogeBank application	2 h
SQL Injection testing	2 h
Upload malicious code testing	2 h
Business logic testing	2 h
Session testing	2 h
Transaction testing	2 h
Command injection testing	1 h
CSRF testing	1 h
Buffer overflow testing DogeBank	2 h
Buffer overflow testing GNB	2 h
Client side testing	1 h
Report - Config and deployment	2 h
Report - Session management	2 h
Report - Business logic	2 h
Report - Data validation	2 h
Report - Vulnerabilities overview	1 h
Report - Executive summary	1 h
Report - CVSS scores	1 h
Sum	36 h

Florian Mauracher

Task	Time
Set up pentesting environment	2 h
Test basic functionality of DogeBank	2 h
Configuration and Deploy Management Testing	1 h
Authorization Testing	2 h
Data Validation Testing	2 h
Cryptography Testing	2 h
Business Logic Testing	2 h
Configure basic LaTeX template	2 h
Create LaTeX template for OWASP checklist	2 h
LaTeX table formatting and presentation	2 h
Write basic custom testing tools	2h
Write Authorization section	2 h
Write Data Validation section	2 h
Write Cryptography section	2 h
Score the vulnerabilities	1 h
Presentation	2 h
Sum	30 h

Mahmoud Naser

Task	Time
Setting up VM environment	1 h
Setting up hacking tools	1 h
Test Role Definitions	1 h
Test User Registration Process	1 h
Test Account Provisioning Process	2 h
Testing for Account Enumeration and Guessable User Account	2 h
Testing for Credentials Transported over an Encrypted Channel	1 h
Testing for default credentials	1 h
Testing for Weak lock out mechanism	1 h
Testing for bypassing authentication schema	2 h
Test remember password functionality	2 h
Testing for Browser cache weakness	2 h
Testing for Weak password policy	2 h
Testing for Browser cache weakness	2 h
Documenting bugs in LaTeX report	1 h
Working on presentation	1 h
Adding Tools to LaTeX report	1 h
Sum	24 h

2 Vulnerabilities Overview

In this chapter, the major security flaws of both the DogeBank and the Goliath National Bank applications will be briefly summarized and compared.
After testing the two web applications thoroughly, we found the following vulnerabilities to be the most serious.

2.1 DogeBank

2.1.1 Bypassing authorizations

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-AUTHZ-002

Any unauthenticated user may approve registrations or transactions using the correct GET request. It is also possible to directly register an employee without even having to login. Considering this security issue, the whole authentication process proves to be useless.

2.1.2 Privilege escalation

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-AUTHZ-003

A logged in customer is able to access employee pages without having the proper privileges, allowing therefore actions which shouldn't be possible. This is also possible the other way around, although this could be considered as a bug, rather than a vulnerability.

2.1.3 Stored XSS in all forms

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-002

Input values in forms are not validated whatsoever, allowing to store custom scripts inside the database when filling out forms. These scripts will automatically be executed by employees who view the details of the client. This is also valid for stored CSS and HTML injection.

2.1.4 SQL injection in all forms

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-005

The same concept explained in the XSS vulnerability also applies for SQL injection: since the input values in forms are not validated, it is possible to inject SQL statements. Even though multiple SQL queries are not supported, tricking the server into authenticating a user without valid credentials, or using invalid TANs for that matter, is still easy.

2.1.5 Session hijacking

- Likelihood: *low*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-SESS-004

The server exchanges the session id with the client in clear-text, allowing man in the middle attacks or social engineering techniques aimed at hijacking a session. Once the session of a user has been hijacked, an attacker could even gain employee privileges.

2.1.6 No file extension check during upload

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-BUSLOGIC-008

During the upload of batch files for multiple transactions, the file extension is not verified, therefore it is possible to upload any potential file or to use Unix commands as the name of the file.

2.1.7 Test Upload of Malicious Files

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-BUSLOGIC-009

The weak file upload policy leads to another big issue, since the uploaded file may contain malicious code. Exploiting this vulnerability allows to gain complete control of the web application.

2.1.8 Command injection

- Likelihood: *low*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-013

When uploading a file, it is possible the use bad filenames which will result in arbitrary shell command injections.

2.1.9 Stack overflow

- Likelihood: *low*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-014

Given that uploading a file allows to execute arbitrary commands, it also allows to inject arbitrary strings, which will be parsed as program arguments during a batch transaction operation. Since the length of the additional argument is not checked, this produces a stack overflow.

2.2 Goliath National Bank

2.2.1 SQL injection in all forms

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-005

Since some input values in forms are not validated, it is possible to inject SQL statements. Even though multiple SQL queries are not supported, tricking the server into authenticating a user without valid credentials, or using invalid TANs for that matter, is still easy.

2.2.2 Accessible sensitive information

- Likelihood: *high*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-CONFIG-003

Some sensitive information were left available for attackers to steal. This is the case of a README.md file which can be easily found by listing tools and contains system/root access credentials as well as other important information.

2.2.3 Bypassing authorizations

- Likelihood: *low*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-AUTHZ-002

Some key pages can be accessed without requiring any privileges. These pages (e.g. `manage_registration.php`) are supposed to be accessible only to employees, giving an attacker the highest possible privileges while accessing said pages directly. Exploiting this vulnerability proves to be somewhat tricky, since the Javascript code necessary for this purpose is not delivered directly but needs to be fetched manually.

2.2.4 Stored XSS in all forms

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-002

Input values in forms are not validated whatsoever, allowing to store custom scripts inside the database when filling out forms. These scripts will automatically be executed by employees who view the details of the client. This is also valid for stored CSS and HTML injection.

2.2.5 Session hijacking

- Likelihood: *low*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-SESS-004

The server exchanges the session id with the client in clear-text, allowing man in the middle attacks or social engineering techniques aimed at hijacking a session. Once the session of a user has been hijacked, an attacker could even gain employee privileges.

2.2.6 Command Injection

- Likelihood: *high*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-INPVAL-013

Injecting commmands was possible in the by `new_transaction_multiple.php` by inserting them directly into the filename of the file that was to be uploaded.

2.2.7 Test Upload of Malicious Files

- Likelihood: *medium*
- Implication: *high*
- Risk: *high*
- Reference: OWASP OTG-BUSLOGIC-009

Even though the GNB application checks for file extensions, it does not parse the entire filename, which allows to upload malicious code. This attack is somewhat complicated on this web application, since it requires the attacker to rename the file afterwards and find a way to execute the malicious code. This, however, can be done thanks to command injection.

2.3 Comparison

Comparing the two web applications, we found many similar vulnerabilities: both are completely exposed to stored XSS attacks, session hijacking and allow to upload malicious files as well as inject commands during the batch transaction operation. Moreover, both have some unique vulnerabilities, like the possibility to easily bypass any kind of authorization mechanism and create C buffer overflows on the DogeBank application, or reading files containing sensitive information on the GNB application. While both applications were definitely not secure, we concluded, however, that the GNB application was less vulnerable than the DogeBank application, since the complexity of the attacks carried out to exploit the GNB vulnerabilities proved to be much higher compared to DogeBank: for example bypassing authorization requires much more knowledge in the GNB case.

3 Detailed Report

3.1 Tool description

Chrome Cookie Extension

- **Used by**
  Mahmoud Naser
- **Used for**
  This tool was useful when inspecting cookie sessions, HTTP Headers and GET and POST variables
- **Useful in**
  OTG-AUTHN-004 and OTG-AUTHN-005

Acunetix Web Vulnerability Scanner

- **Used by**
  Mahmoud Naser
- **Used for**
  This tool was useful in providing a page structure to the application as well as pointing out major vulnerabilities such as Blind SQL Injection, Code execution and cross site scripting as well as other less critical errors.

  The disadvantage of using this tool is that the free version does not show where the vulnerability occurs as in which page it appears, and no ability to export results for later inspection.
- **Useful in**
  OTG-INPVAL-005 and OTG-IDENT-002

Arachni

- **Used by**
  Florian Mauracher

- **Used for**
  This tool was useful for basic application reconnaissance and identifying general issues with the application. As it had issues with the navigation scheme employed in our site, it was of limited utility on our site.
- **Useful in**
  OTG-CONFIG-003 and OTG-INPVAL-012 and OTG-CRYPST-003

Zed Attack Proxy

- **Used by**
  Florian Mauracher
- **Used for**
  This tool was used as a proxy to capture the complete communication between the browser and the application. The captured traffic was later used to identify the input vectors for the authorization section, and fuzz certain input parameters.
- **Useful in**
  OTG-AUTHZ-001 , OTG-AUTHZ-002 , OTG-AUTHZ-003 and OTG-AUTHN-004

SQLMap

- **Used by**
  Alexander Lill
- **Used for**
  This tool was useful for testing the different SQL attack vectors. It was used for fuzzing with different usernames and passwords and was able to retrieve the database structure, the current database user and some more internal mysql information.

  The disadvantage of using this tool is that it is complicated to test websites that are not simply using POST requests for the data transmission, but nested JavaScript calls.
- **Useful in**
  OTG-INPVAL-005

Skipfish

- **Used by**
  Alexander Lill

- **Used for**
  This tool was useful for finding the common vulnerabilities of the given website. This included vulnerable forms like the `/login.php`, files and directories that were referenced by the returned HTML files or which cookies were created by the website.

  The disadvantage of using this tool is that the results of the tool do overlap which all the other tools (e.g. arachni) but it includes less details.
- **Useful in**
  OTG-CONFIG-003

Burp Suite

- **Used by**
  Lorenzo Donini
- **Used for**
  This tool was essential for most of the tests involving an accurate analysis of the HTTP requests and responses, since it provided Proxy interception, several functionalities useful for information gathering as well as for stress testing the web applications and analyzing sessions. Additionally, Burp offers an intruder, which helped in automating most of the input validation tests (e.g. buffer overflow).
- **Useful in**
  OTG-SESS-002 , OTG-SESS-003 , OTG-INPVAL-005 , OTG-INPVAL-013 and OTG-INPVAL-014

suchsecure.py

- **Used by**
  Lorenzo Donini
- **Used for**
  This python script was useful for analyzing several bugs and business logic flaws inside the DogeBank application. This tool was custom-built for the purpose of performing specific HTTP requests to the server without having to strictly follow the business logic flow. This basic functionality also helped to flood requests and keeping a session state between operations.
- **Useful in**
  OTG-SESS-004 and OTG-BUSLOGIC-001 , as well as for other bugs and business logic issues, described in chapter 4.

3.2 Configuration and Deploy Management Testing

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

	DogeBank
Observation	The whole configuration allows to list the content of directories and see files which should otherwise not be visible to the outside. This made analyzing the web application a lot easier. Also, while exploring the folder structure of the server, some leftover files were found, as well as hidden folders.
Discovery	The web application structure was brute-forced thanks to the DirBuster tool. Additionally, leftover files (e.g. `/employee_registration.php`) were found later by listing the contents of some folders, as well as other files containing sensitive information. More specifically, this is the case of the .git folder.
Likelihood	Listing directories is a trivial task. Howver, in order to access the hidden .git folder an attacker must be skilled and know where to search for sensitive information.
Implication	Having access to the data contained inside the hidden .git folder allows to get access to the whole source code of the web application, also thanks to the fact that directory listing is active.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** N **A:** N **Score:** 5.3

	Goliath National Bank
Observation	Although the web application doesn't allow directory listing, it is still possible to directly open certain files. We found that some hidden folders and files containing sensitive information were left inside the folder of the web application.
Discovery	We noticed this flaw by simply observing the structure of the web application. More specifically, we found that the root folder still contained the .git folder of the project, along with a README.md file that contained the credentials to access the system as well as the database. It would also be possible to find the files contained inside the /database folder, in which the database structure as well a basic setup is stored.
Likelihood	An attacker could easily brute-force the names of files and folders contained inside the root folder of the web application (perhaps using an appropriate tool). Figuring out the existence of such vulnerability is, hence, doomed to happen.
Implication	This was the most severe vulnerability found in the GNB web application. Knowing about the existence of the .git folder doesn't help an attacker much, since directory listing is disabled (an attacker would have to brute-force the names of the objects inside of it); however, the unprotected README.md is a different matter. Once an attacker has read the credentials inside this file, he could connect to the server via ssh and get complete control over the system (root privileges included).
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 9.8

3.2.2 Test HTTP Methods (OTG-CONFIG-006)

	DogeBank
Observation	We observed that the GET, POST, HEAD and OPTIONS methods are allowed. 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 using netcat.
Likelihood	N/A
Implication	N/A
CVSS	Secure

	Goliath National Bank
Observation	We observed that the GET, POST, HEAD and OPTIONS methods are allowed.
Discovery	This discovery was made using netcat.
Likelihood	N/A
Implication	N/A
CVSS	Secure

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

	DogeBank
Observation	The application is only accessible over HTTP.
Discovery	No HTTPS is enforced, therefore all data sent between the server and client is not encrypted.
Likelihood	An attacker could perform a man in the middle attack.
Implication	Sniffing the network traffic, all data exchanged between the server and the client can be read as clear text. No confidentiality at all is supported on this end.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** H **I:** H **A:** N **Score:** 6.8

	Goliath National Bank
Observation	The application is only accessible over HTTP.
Discovery	No HTTPS is enforced, therefore all data sent between the server and client is not encrypted.
Likelihood	An attacker could perform a man in the middle attack.
Implication	Sniffing the network traffic, all data exchanged between the server and the client can be read as clear text. No confidentiality at all is supported on this end.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** H **I:** H **A:** N **Score:** 6.8

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

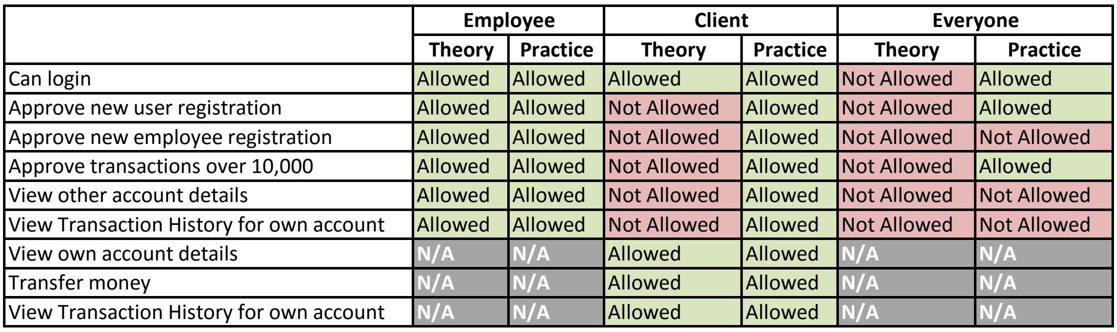
	DogeBank
Observation	The web application doesn't support additional technologies like Flash, Silverlight or Java.
Discovery	No cross-domain policy files were found.
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

	Goliath National Bank
Observation	The web application doesn't support additional technologies like Flash, Silverlight or Java.
Discovery	No cross-domain policy files were found.
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

3.3 Identity Management Testing

3.3.1 Test Role Definitions (OTG-IDENT-001)

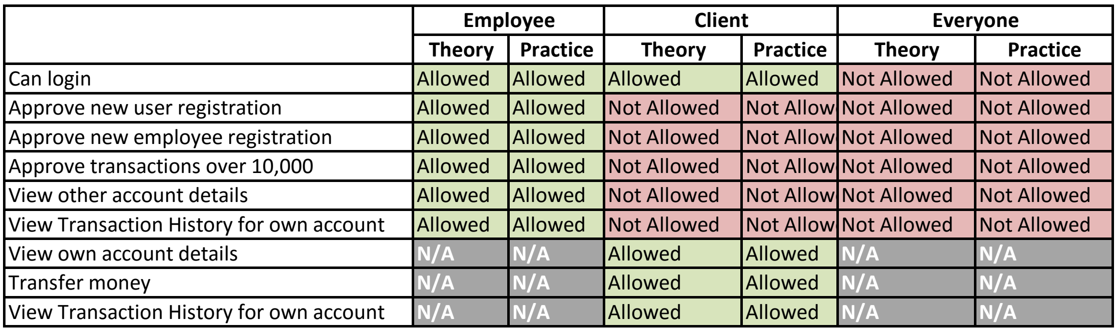
	DogeBank
Observation	Clients and non-logged in users are able to access Employee privileges, see 3.1
Discovery	This was discovered through the bug searching stage and while completing the OTG-AUTHN-004 "Testing for bypassing authentication schema" test.
Likelihood	Skill level needed to uncover this is moderate, using simple direct page access and basic trail and error with parameters is enough to exploit this bug, so the likelihood of this vulnerability is quite high
Implication	This could have tremendous damages to the bank, as adding employee accounts and transferring functions
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** L **Score:** 9.4



	Employee		Client		Everyone	
	Theory	Practice	Theory	Practice	Theory	Practice
Can login	Allowed	Allowed	Allowed	Allowed	Not Allowed	Allowed
Approve new user registration	Allowed	Allowed	Not Allowed	Allowed	Not Allowed	Allowed
Approve new employee registration	Allowed	Allowed	Not Allowed	Allowed	Not Allowed	Not Allowed
Approve transactions over 10,000	Allowed	Allowed	Not Allowed	Allowed	Not Allowed	Allowed
View other account details	Allowed	Allowed	Not Allowed	Allowed	Not Allowed	Not Allowed
View Transaction History for own account	Allowed	Allowed	Not Allowed	Allowed	Not Allowed	Not Allowed
View own account details	N/A	N/A	Allowed	Allowed	N/A	N/A
Transfer money	N/A	N/A	Allowed	Allowed	N/A	N/A
View Transaction History for own account	N/A	N/A	Allowed	Allowed	N/A	N/A

Figure 3.1: Role Definitions

	Goliath National Bank
Observation	Clients, Employees and non-logged in users all act as expected, see 3.2
Discovery	This was verified through basic function testing and security testing tools (ZAP).
Likelihood	N/A
Implication	N/A
CVSS	Secure



	Employee		**Client**		**Everyone**	
	Theory	**Practice**	**Theory**	**Practice**	**Theory**	**Practice**
Can login	Allowed	Allowed	Allowed	Allowed	Not Allowed	Not Allowed
Approve new user registration	Allowed	Allowed	Not Allowed	Not Allow	Not Allowed	Not Allowed
Approve new employee registration	Allowed	Allowed	Not Allowed	Not Allow	Not Allowed	Not Allowed
Approve transactions over 10,000	Allowed	Allowed	Not Allowed	Not Allow	Not Allowed	Not Allowed
View other account details	Allowed	Allowed	Not Allowed	Not Allow	Not Allowed	Not Allowed
View Transaction History for own account	Allowed	Allowed	Not Allowed	Not Allow	Not Allowed	Not Allowed
View own account details	N/A	N/A	Allowed	Allowed	N/A	N/A
Transfer money	N/A	N/A	Allowed	Allowed	N/A	N/A
View Transaction History for own account	N/A	N/A	Allowed	Allowed	N/A	N/A

Figure 3.2: Role Definitions

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

	DogeBank
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. Identities are not verified nor checked at this stage due to application limitations, email format verification is missing from the form. No CAPTCHA or similar tests available to test for human accounts
Discovery	The email verification test was discovered through trail and error while registering. Lack of human testing was found through simple observation.
Likelihood	likely, due to intentional or unintentional mistyping. Robot accounts are less likely to occur and it depends on the skill level of the attacker.
Implication	No serious impact as TAN codes are not sent to the email. Robot accounts can be used to perform a DOS attack.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** N **A:** L **Score:** 5.3

	Goliath National Bank
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. Identities are not verified nor checked at this stage due to application limitations. No CAPTCHA or similar tests available to test for human accounts.
Discovery	Lack of human testing was found through simple observation.
Likelihood	Robot accounts are less likely to occur and it depends on the skill level of the attacker.
Implication	Robot accounts can be used to perform a DOS attack.
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)

	DogeBank
Observation	Provisioning clients is an easy process with no effective mechanisms to verify or vet clients besides a manual approval process. Vulnerabilities with creating the client account have been discussed in the previous section OTG-IDENT-002 , but using OTG-AUTHZ-002 an non-authenticated user is able to approve a client user request. Provisioning Employees is set up to only be possible by other employees, but using OTG-AUTHZ-001 , a non-authenticated user is able to create an employee account.
Discovery	This was discovered through trail and error in the bug discovery phase.
Likelihood	This vulnerability requires some basic knowledge of the PHP pages and variable names, which can be obtained using basic testing tools, so while this vulnerability is likely to be discovered by an attacker with some basic skills.
Implication	This presents some serious impact, as creating an employee account will give the attacker full access to the application and administrative functions such as creating accounts and approving money transfers.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** L **A:** N **Score:** 5.3

	Goliath National Bank
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	Robot accounts are less likely to occur and it depends on the skill level of the attacker.
Implication	If the DOS attack is severe enough it could stop the application from from working.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** N **I:** N **A:** N **Score:** 0

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

	DogeBank
Observation	Errors provided from enumerating through the different cases are as follows : • Valid username with correct password : Expected result of logging in • Valid username with incorrect password : Returns "Username or Password incorrect! Please try again!" • Invalid username : Returns "Username or Password incorrect! Please try again!"
Discovery	This was found through trail and error on different combinations of credentials.
Likelihood	N/A
Implication	N/A
CVSS	Secure

	Goliath National Bank
Observation	Errors provided from enumerating through the different cases are as follows : • Valid username with correct password: Expected result of logging in • Valid username with incorrect password: Returns "Invalid login credentials!" • Invalid username : Returns "Invalid login credentials!"
Discovery	This was found through trail and error on different combinations of credentials.
Likelihood	N/A
Implication	N/A
CVSS	Secure

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

	DogeBank
Observation	No username policy applied.
Discovery	N/A
Likelihood	N/A
Implication	N/A
CVSS	N/A

	Goliath National Bank
Observation	Username has to be in valid email address format.
Discovery	Through trail and error.
Likelihood	N/A
Implication	N/A
CVSS	Secure

3.4 Authentication Testing

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

	DogeBank
Observation	Due to the fact, that no encryption is used when accessing the application, credentials transported over an encrypted channel are vulnerable. Relevant information regarding unencrypted communication has already been mainly covered in section OTG-CRYPST-003 and in OTG-CONFIG-007 .
Discovery	Through observation.
Likelihood	Given that a simple network sniffing tracking tool would pick up the credentials, this vulnerability is likely to be exploited.
Implication	The attacker is able access the said account, and perform actions on behalf of said user, and it the account is an employee account, the attacker would gain access to the administrative tasks
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** H **I:** H **A:** N **Score:** 6.8

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Implication	See above
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** H **I:** H **A:** N **Score:** 6.8

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

	DogeBank
Observation	The Administrator had a predictable username "employee" and an easy to guess short password "pass".
Discovery	Credentials were provided in the team report.
Likelihood	If an attacker is performing a basic dictionary attack, this vulnerability would very likely be discovered.
Implication	Gaining access to the employee account would allow the attacker to perform administrative tasks on the application.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** L **Score:** 9.4

	Goliath National Bank
Observation	Administrator(Employee) and client users did not have predictable credentials.
Discovery	Credentials were provided in the team report.
Likelihood	N/A
Implication	N/A
CVSS	Secure

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

	DogeBank
Observation	No lockout mechanism is deployed.
Discovery	Through trail and error.
Likelihood	This allows for brute force attacks on accounts if the attacker has a username.
Implication	Gaining access to an employee or client account, which allow the attacker to perform that accounts tasks.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** N **Score:** 6.5

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Implication	See above
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** N **Score:** 6.5

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

	DogeBank
Observation	Through direct page access (see OTG-AUTHZ-002 and OTG-AUTHZ-004), and SQL injection (see 2.1.4) a non-authenticated user is able to gain access to both client and employee functions without being logged in. For more details on what privileges can a non-authenticated user can exploit, please refer to 3.1. • Direct page access example : the url `/downloadTans.php?userid=1` allows a non authorized user to download the TAN codes for the user with the user_id of 1. • SQL Injection example : by using the username *'OR 1=1 AND user_name=‘employee'* # an attacker can gain access to the admin account ‘employee’
Discovery	This was discovered through trail and error in the bug discovery phase.
Likelihood	This vulnerability requires some basic knowledge of the PHP pages and variable names, which can be obtained using basic testing tools, so while this vulnerability is likely to be discovered by an attacker with some basic skills.
Implication	This presents some serious impact, as creating an employee account will give the attacker full access to the application and administrative functions such as creating accounts and approving money transfers.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** L **Score:** 9.4

	Goliath National Bank
Observation	The PHP Session ID cookie variable is not destroyed after logout, and even though the user is unable to use the "back" option on the browser, the PHP Session ID does not change though making it extremely predictable after logging
Discovery	This was done through cookie inspection using the chrome and Firefox Developer extensions.
Likelihood	The skill level required for exploiting this is minimal, Knowledge needed for this attack is basic cookie inspection and injection and Network sniffing.
Implication	Overtake an existing user session and gain access to that users privileges.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

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

	DogeBank
Observation	Passwords in this application are sent in clear text.
Discovery	Through HTTP Header inspection
Likelihood	The skill level required for exploiting this is minimal, Knowledge needed for this attack is basic HTTP Header inspection and Network sniffing.
Implication	Gaining access to an employee or client account, which allow the attacker to perform that accounts tasks.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Implication	See above
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

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

	DogeBank
Observation	Browser Cache settings are set up correctly, so ’Back’ on the browser does not work.
Discovery	Though observation.
Likelihood	N/A
Implication	N/A
CVSS	Secure

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Implication	See above
CVSS	Secure

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

	DogeBank
Observation	No password policy is implemented.
Discovery	Though observation.
Likelihood	The skill level required for exploiting a week password is minimal, either through password cracking tools or social engineering.
Implication	Gaining access to an employee or client account, which allow the attacker to perform that accounts tasks.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** N **Score:** 6.5

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Implication	See above
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** N **Score:** 6.5

3.5 Authorization Testing

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

	DogeBank
Observation	No user defined input vectors to include additional files were discovered during testing.
Discovery	After capturing the requests and responses of all pages available in the application with Zed Attack proxy, the recorded traffic was checked for possible input vectors. No input vectors referencing files were found.
Likelihood	N/A
Implication	N/A
CVSS	N/A

	Goliath National Bank
Observation	User defined input vectors were observed on all pages of the application which require authentication. These input vectors don't seem to be direct references to files in the web application, thus an indirect mapping of the specified names to included files is assumed. Due to this indirect mapping accessing arbitrary files on the server is not possible by modifying these input vectors. Nevertheless an authenticated attacker is able to use this mechanism to get access to all included pages as described in OTG-AUTHZ-002
Discovery	After capturing the requests and responses of all pages available in the application with Zed Attack proxy, the recorded traffic was checked for possible input vectors. Multiple input vectors referencing sections of the site were identified, but manual testing revealed that none of these referred to filenames directly.
Likelihood	N/A
Implication	N/A
CVSS	Secure

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

	DogeBank
Observation	All employee pages are fully accessible for any authenticated user e.g. client.
Discovery	Manually accessing the address of the employee pages while being logged in as user allowed full access to these sites. `/employee_home.php`
Likelihood	To exploit this vulnerability an attacker needs to be aware of the address of the employee pages and have an valid client account.
Implication	This vulnerability allows bypassing all authorization mechanisms put in place, granting the attacker the highest privileges available in the application.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 8.8

	Goliath National Bank
Observation	Direct access to the employee page is not possible as client. However by using the file inclusion technique described in OTG-AUTHZ-001 an authenticated attacker is able to access all included pages of the application.

Discovery	Manual testing direct access to the employee pages (`/employee/employee.php`) did not grant any positive results.
	Using the information gathered in OTG-AUTHZ-001 it was possible to access the employee pages by modifying the POST parameters of the client page to reference the employee pages. Page: `/client/client.php`
	Original POST parameters:
	`section=my_accounts&frame=account_overview&account=10000002`
	Updated POST parameters:
	`section=employee_area&frame=manage_registration`
	This opened the page with pending registrations which should only be visible as employee. Additional effort is required to start any further actions from this page (e.g. approving users) as the required javascript is missing and all references lead to the wrong page. The same applies for all other employee pages.

Likelihood	To access these sites an attacker has to be authenticated and aware of the names of the employee pages. As the client area pages doesn't contain the required javascript code for the employee area, additional knowledge and effort is required to perform actions on the pages accessed by this method.
Implication	This vulnerability allows bypassing all authorization mechanisms put in place, granting the attacker the highest privileges available in the application.
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 7.5

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

	DogeBank
Observation	None of the direct object references that were observed in the application appears to have additional authorization or authentication checks.
Discovery	After capturing the requests and responses of all pages available in the application with Zed Attack proxy, the recorded traffic was checked for direct references to objects supplied by the user. Manual testing of these revealed that no authorization or authentication was in place when accessing these objects. • Download transaction history of an arbitrary user `/downloadTransaction.php?userid=1` • Download tans of an arbitrary user `/downloadTans.php?userid=1` • Approve an arbitrary transaction without being logged in `/approvetransaction.php?transid=2` • Approve an arbitrary user who just registered `/approveuser.php?userid=2`
Likelihood	To exploit this vulnerability an attacker needs to know the address of the pages containing the direct object references. The sequentially increasing user IDs starting with 1 make it easy to obtain this information for all users of the bank.
Implication	Trough this vulnerability it is possible to directly access all functionality an employee has available. As this is possible even for an unauthenticated user this vulnerability is also referenced in OTG-AUTHN-004 .
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** L **Score:** 9.4

	Goliath National Bank
Observation	No direct object references were observed.
Discovery	N/A
Likelihood	N/A
Implication	N/A
CVSS	Secure

3.6 Session Management Testing

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

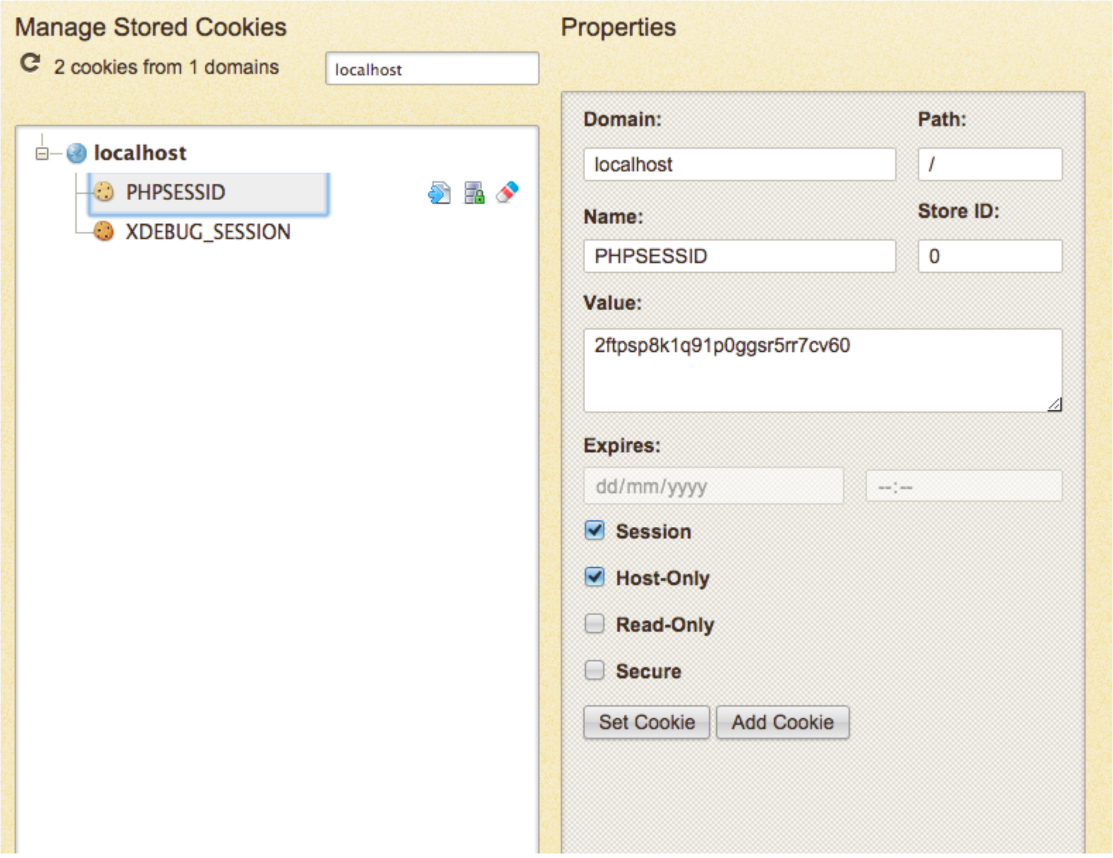
	DogeBank
Observation	When accessing the application, a randomly generated PHPSESSID session cookie is set. The cookie doesn't have an expiration date nor is it tagged as secure. Apparently the session cookie is already set before logging into the application. This cookie is simply replaced by a new one once the user logs out of the application. No other cookies are set. Also if the cookie is tampered with, the server automatically generates a new cookie, containing a new session ID.
Discovery	The PHPSESSID cookie has been discovered while intercepting HTTP requests/responses using Burp. The same cookie details were later on confirmed using the Cookies plugin for browser.
Likelihood	N/A
Implication	Since the only used cookie only contains the session ID, even though it is easy to change the value of the cookie, no other session values are exposed to the user. It is still possible to hijack another session by changing the entire value of the cookie with the one associated to another user (see OTG-SESS-004).
CVSS	N/A

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	The PHPSESSID cookie was analyzed using the Cookies plugin for browser.
Likelihood	N/A
Implication	See above
CVSS	N/A

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

	DogeBank
Observation	We found that the cookie generated by the application does NOT set the following attributes: • Secure • HttpOnly • Expires The application also sets the domain attribute very loosely, since the path is set to the root directory "/".
Discovery	The attributes were analyzed using the Cookies plugin for browser.
Likelihood	It is easy to access the cookies from Javascript, as long as the browser supports client-side scripting.
Implication	Weak protection for cookies means that these can be accessed via Javascript to perform XSS attacks.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	The attributes were analyzed using the Cookies plugin for browser.
Likelihood	It is easy to access the cookies from Javascript, as long as the browser supports client-side scripting.
Implication	Weak protection for cookies means that these can be accessed via Javascript to perform XSS attacks.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5



Manage Stored Cookies

2 cookies from 1 domains

localhost

- localhost
  - PHPSESSID
  - XDEBUG_SESSION

Properties

Domain: localhost

Path: /

Name: PHPSESSID

Store ID: 0

Value: 2ftpsp8k1q91p0ggsr5rr7cv60

Expires: dd/mm/yyyy --:--

☑ Session

☑ Host-Only

☐ Read-Only

☐ Secure

Figure 3.3: PHPSESSID cookie for the DogeBank application

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

	DogeBank
Observation	After careful observation and testing we found that once a session ID has been set, this will not be changed until a user logs out. More specifically, the session ID will not be invalidated before a login operation, remaining the same after having logged in (a session ID is generated on the login page already).
Discovery	The session ID generation was thoroughly observed thanks to a Proxy that intercepted all GET/POST requests and responses to/from the server. For this the Burp Suite tool was used.
Likelihood	Setting up a possible attack is theoretically easy, but it requires a victim to be tricked by the attacker, making the attack less likely depending on the victim.
Implication	An attacker could generate a session ID for himself, then force the same ID onto a user, hijacking that users' session in case of successful authentication with the server. This implicates full access to the account of a user.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

	Goliath National Bank
Observation	Unlike the DogeBank case, the session ID, once generated, never changes. Even though the server destroys all session variables associated to a user, after a logout operation, the session ID is not unset. The session ID is first generated on login page, same as for the DogeBank application.
Discovery	This discovery was made while analyzing the session cookies thanks to the Cookies browser extension.
Likelihood	The same likelihood described for the DogeBank application applies.
Implication	The same implications mentioned for the DogeBank application apply.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

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

	DogeBank
Observation	After observing requests and responses between the client and the server, we observed that session IDs are always sent in HTTP headers. Although the session ID is never explicitly passed in URLs, no encryption is provided whatsoever and the session ID does not change until a user explicitly logs out. No other session variables are generated, therefore only the session ID is affected.
Discovery	This observation was made when analyzing session management. Refer to sections OTG-SESS-001 and OTG-SESS-003 .
Likelihood	As long as an attacker can sniff the network traffic and read the session ID of a user, it is very easy to hijack a session. This approach makes it even easier than hijacking a session through social engineering.
Implication	An attacker can perform a man in the middle attack, read the unencrypted HTTP messages exchanged between a user and the server, in order to impersonate the user and hijack an existing session. This implicates full access to the account of a user.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	The discovery was made thanks to GET/POST requests interception.
Likelihood	The same likelihood described for the DogeBank application applies.
Implication	The same implications mentioned for the DogeBank application apply.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

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

	DogeBank
Observation	Having observed different functionalities offered by the server, we noticed that no encryption is used at all; furthermore the session ID is stored as a cookie in the browser of the user, which makes things a lot easier. In order to trick a user into execute specific operations, however, we found out that some knowledge of the web application was required. Given this knowledge, it proves easy to compromise the entire web application. We found a list of pages potentially subject to CSRF: • approveuser.php: can be exploited without privileges • approvetransaction.php: can be exploited without privileges • downloadTans.php: can be exploited without privileges • register_employee.php: only accessible to employees. The existence of this page has to be known to an attacker beforehand • tran.php: although this page is easy to exploit, an attacker would need to have access to the TANs of a user. This could be done beforehand by downloading from `/downloadTans.php`
Discovery	All of the observations were made while navigating the website and brute-forcing different combinations of attacks. Other helpful tools helpful were DirBuster and the DogeBankHack custom script, since these allowed to gain a better understanding of the website's structure.
Likelihood	In theory, performing a CSRF attack would be easy, since session IDs are stored in browsers and sent over unencrypted channels. However, in this specific case, the attack complexity increases, since the attacker requires additional knowledge, like the ID of a user and the names of the vulnerable pages. In most of the above mentioned cases, getting access to the ID of a user proves to be trivial, since it is contained in pages shown to the user (easy to exploit via Javascript).
Implication	An attacker in possession of the previously described information could be able to trick a victim into executing operations predetermined by the attacker himself, like starting transactions to arbitrary accounts or registering arbitrary employees. The impact in the former case would compromise the whole bank account of a user, or grant a privilege escalation in the latter case.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** H **A:** L **Score:** 6.4

	Goliath National Bank
Observation	The same predisposition showed by the DogeBank application holds true, i.e. no encryption is used in client-server message exchanges and the session ID is stored as a cookie in the browser of the user. Similar observations regarding the knowledge required to perform a CSRF attack also hold. In particular, these pages proved to be vulnerable: • verify_transaction.php: as for the DogeBank case, an attacker would need to have access to the TANs of a user in order to exploit this page. Getting access to these TANs, however, is only possible by either accessing the database directly or intercepting the confirmation email sent to a user; • manage_registration.php: requires knowledge of the ID of the user that we want to approve/reject. Other than that, exploiting this page proved to be trivial and just required a proper analysis of the client-side code; • manage_transfer.php: this page is also easy to exploit, since it only requires knowledge of the pending transactions IDs, which can be found on the same page; • manage_clients.php: although this page can be accessed directly, its existence has to be known to the attacker. It is important to stress that, given the layout of the pages, the above mentioned pages can simply be used as section parameters inside a POST request to the employee.php and client.php pages, without the need to copy additional Javascript from the container pages.
Discovery	These observations were made while navigating the website and brute-forcing different combinations of attacks.
Likelihood	As for the DogeBank case, although a CSRF attack would be easy, the complexity increases in this specific case, since an attacker needs some knowledge about the layout of the pages (i.e. the names of the above mentioned vulnerable php sections). This information is harder to acquire than in the previous case however, as it required a thorough client-side code analysis.
Implication	Assuming an attacker is capable of obtaining the required information for such an attack to work, this would have the same implications described in the DogeBank case.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

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

	DogeBank
Observation	We observed that the logout functionality is working properly, as the logoutAction.php destroys an existing session, creating a new one. Trying to access a page that requires authentication, after having logged out, fails and the server responds with a "PHPSESSID=deleted" cookie, redirecting the browser to the login page. We also observed that there is no logout timer, allowing a user to be logged in indefinitely, as long as the logout is not manually triggered.
Discovery	These observations were made using the Burp Repeater tool.
Likelihood	N/A
Implication	Since there is no session timeout policy whatsoever, this could prove to be a vulnerability. This is analyzed in more details in OTG-SESS-007 .
CVSS	Secure

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply. Additionally, we found out that the logout functionality does not create a new session ID for the user, but rather destroys all of the server-side variables associated to that user only.
Discovery	These observations were made using the Burp Repeater tool.
Likelihood	N/A
Implication	The same implications described for the DogeBank application apply.
CVSS	Secure

3.6.7 Test Session Timeout (OTG-SESS-007)

	DogeBank
Observation	We observed that no session timeout policy was implemented, neither on server side nor on client side. Although the only sensitive information stored on client side is the session ID, we proved that it was possible to reuse the same session any number of times.
Discovery	These observations were made using the Burp Repeater tool.
Likelihood	A session hijacking is easy to perform, as long as the attacker can either sniff the traffic between the victim and the server, or has access to the device from which the victim logged in.
Implication	The lack of a session timeout gives a potential attacker indefinite time to perform a session hijacking. Once a session has been hijacked, an attacker has complete access over a user's account.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 5.6

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	These observations were made using the Burp Repeater tool.
Likelihood	The same likelihood described for the DogeBank application applies.
Implication	The same implications mentioned for the DogeBank application apply.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 5.6

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

	DogeBank
Observation	Considering that the only session variable set by the application is the session ID, which we observed is randomly generated by the server, there isn't really any margin for session variable overloading.
Discovery	These observations were made using the Burp suite tool.
Likelihood	N/A
Implication	N/A
CVSS	Secure

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	These observations were made using the Burp suite tool.
Likelihood	N/A
Implication	N/A
CVSS	Secure

3.7 Data Validation Testing

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

	DogeBank
Observation	All pages showing user defined input are vulnerable to stored cross site scripting. No input is validated after it is entered by the user as described in OTG-BUSLOGIC-001 .
Discovery	Issues on multiple pages were discovered during manual testing of the input vectors. For example: • Site: `/register.php` • Fields: `First Name` `Last Name` • Input example: `<script>alert(1);</script>` • Shown on: – `/customer_home.php` – `/tran.php` – `/employee_home.php` Javascript code can be written to the database using forms. It will later on be automatically executed on client side (e.g. comments in transactions)
Likelihood	Really easy to perform
Implication	Total control over the affected sites
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 9.8

	Goliath National Bank
Observation	All pages showing user defined input are vulnerable to stored cross site scripting. No input is validated after it is entered by the user as described in OTG-BUSLOGIC-001 .
Discovery	Issues on multiple pages were discovered during manual testing of the input vectors. For example: • Site: `/registration.php` • Fields: `First Name` `Last Name` • Input example: `<script>alert(1);</script>` • Shown on: – `/employee/employee.php` in the manage_registration section – `/client/client.php`
Likelihood	See above
Implication	See above
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 9.8

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

	DogeBank
Observation	We observed several effective SQL Injection vulnerabilities in the DogeBank site. The following could be accomplished: • Timing attacks, e.g. using `username=miWm' AND (SELECT * FROM (SELECT(SLEEP(5)))lqlT)#` • Logging in without valid user credentials, e.g. using `username=-6505' OR 7530=7530- -`
Discovery	This was observed using the tool sqlmap and by manual trial and error.
Likelihood	This has a high likelihood as this is covered by almost all simple tools for penetration testing and can be easily done by hand.
Implication	You can log in as the first user in the database without valid login credentials.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 9.8

	Goliath National Bank
Observation	We observed several effective SQL Injection vulnerabilities in the Goliath National Bank site. The following could be accomplished: • Timing attacks, e.g. using `username=miWm' AND (SELECT * FROM (SELECT(SLEEP(5)))lqlT)#`
Discovery	See above
Likelihood	See above
Implication	This allows to send commands to the SQL database and retrieve some information, but not to log in the user.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** H **I:** H **A:** H **Score:** 9.8

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

	DogeBank
Observation	This vulnerability is not applicable due to the fact that none of the pages of the DogeBank allows to provide a parameter that will be executed. See OTG-INPVAL-013 for a similiar attack using the filename of the uploaded batch transactions file.
Discovery	This was observed using a Proxy that intercepted all GET/POST requests and responses to/from the server. For this the Burp Suite tool was used.
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

	Goliath National Bank
Observation	This vulnerability is not applicable due to the fact that none of the pages of the Goliath National Bank allows to provide a parameter that will be executed.
Discovery	This was observed using a Proxy that intercepted all GET/POST requests and responses to/from the server. For this the Burp Suite tool was used.
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

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

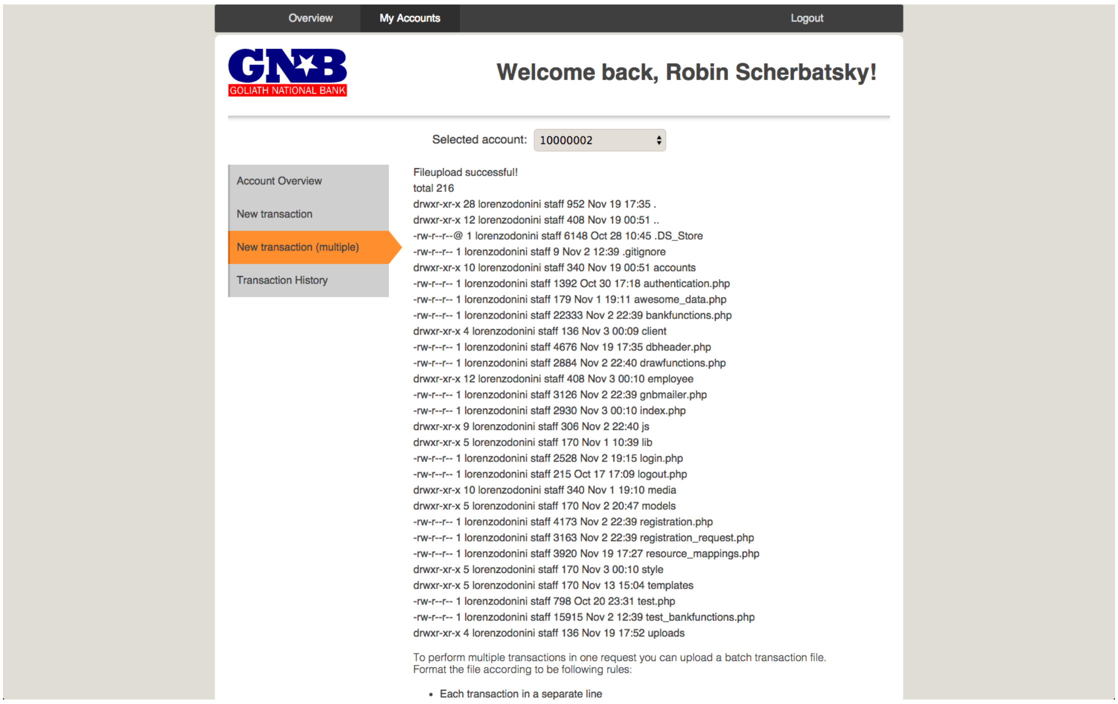
	DogeBank
Observation	This vulnerability is not applicable due to the fact that none of the pages of the DogeBank allows to provide a parameter specifying files.
Discovery	This was observed using a Proxy that intercepted all GET/POST requests and responses to/from the server. For this the Burp Suite tool was used.
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

	Goliath National Bank
Observation	This vulnerability is not applicable due to the fact that none of the pages of Goliath National Bank allows to provide a parameter specifying files. The Goliath National Bank can be exploited though by messing with the internal lookup that gets the php filename from a keyword. See OTG-AUTHZ-001 for a more detailed description.
Discovery	This was observed using a Proxy that intercepted all GET/POST requests and responses to/from the server. For this the Burp Suite tool was used.
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

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

	DogeBank
Observation	We found that this vulnerability was strictly related to ORG-BUSLOGIC-009 , since we managed to exploit this vulnerability inside the `tran.php` page, when uploading a batch transaction file. We observed that it was possible to make the server execute arbitrary commands by injecting them inside the filename of the batch file. The results from stdout were also clearly visible inside the `tran.php` page.
Discovery	In order to execute arbitrary commands we had to insert them in the name of the uploaded file. Here is an example: ;cat /etc/passwd;# The content of the file were not really relevant, since we just wanted to execute code on the machine.
Likelihood	The likelihood of an attacker attempting a command injection through a file upload is very high.
Implication	This vulnerability proves as really severe, since an attacker can execute arbitrary commands on the webserver. The results are not too devastating, as long as the attacker does not have access to the root user; however all the source code is visible this way, which can aid the attacker in attempting to exploit further vulnerabilities.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 9.9

	Goliath National Bank
Observation	Similarly to the case of the DogeBank application, we observed that on the new_transaction_multiple.php page it was indeed possible to inject commands by inserting them directly into the filename of the file that was to be uploaded.
Discovery	In comparison to the DogeBank case, we had to append the commands after an apostrophe, since the call to the C parser would take parameters inside apostrophes. We also had to camouflage the file itself as a .txt/.csv file, since the web application performs checks on the file extension. Here is an example of the filename use for a successful attack: hack’;cd ..;ls -la;#.txt
Likelihood	The likelihood of an attacker attempting a command injection through a file upload is very high.
Implication	The same implications mentioned for the DogeBank case apply. However, since in this case an attacker could have gotten root access (refer to OTG-CONFIG-001), the implications are much worse.
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 8.5



Welcome back, Robin Scherbatsky!

Selected account: 10000002

```
Fileupload successful!
total 216
drwxr-xr-x 28 lorenzodonini staff 952 Nov 19 17:35 .
drwxr-xr-x 12 lorenzodonini staff 408 Nov 19 00:51 ..
-rw-r--r--@ 1 lorenzodonini staff 6148 Oct 28 10:45 .DS_Store
-rw-r--r-- 1 lorenzodonini staff 9 Nov 2 12:39 .gitignore
drwxr-xr-x 10 lorenzodonini staff 340 Nov 19 00:51 accounts
-rw-r--r-- 1 lorenzodonini staff 1392 Oct 30 17:18 authentication.php
-rw-r--r-- 1 lorenzodonini staff 179 Nov 1 19:11 awesome_data.php
-rw-r--r-- 1 lorenzodonini staff 22333 Nov 2 22:39 bankfunctions.php
drwxr-xr-x 4 lorenzodonini staff 136 Nov 3 00:09 client
-rw-r--r-- 1 lorenzodonini staff 4676 Nov 19 17:35 dbheader.php
-rw-r--r-- 1 lorenzodonini staff 2884 Nov 2 22:40 drawfunctions.php
drwxr-xr-x 12 lorenzodonini staff 408 Nov 3 00:10 employee
-rw-r--r-- 1 lorenzodonini staff 3126 Nov 2 22:39 gnbmailer.php
-rw-r--r-- 1 lorenzodonini staff 2930 Nov 3 00:10 index.php
drwxr-xr-x 9 lorenzodonini staff 306 Nov 2 22:40 js
drwxr-xr-x 5 lorenzodonini staff 170 Nov 1 10:39 lib
-rw-r--r-- 1 lorenzodonini staff 2528 Nov 2 19:15 login.php
-rw-r--r-- 1 lorenzodonini staff 215 Oct 17 17:09 logout.php
drwxr-xr-x 10 lorenzodonini staff 340 Nov 1 19:10 media
drwxr-xr-x 5 lorenzodonini staff 170 Nov 2 20:47 models
-rw-r--r-- 1 lorenzodonini staff 4173 Nov 2 22:39 registration.php
-rw-r--r-- 1 lorenzodonini staff 3163 Nov 2 22:39 registration_request.php
-rw-r--r-- 1 lorenzodonini staff 3920 Nov 19 17:27 resource_mappings.php
drwxr-xr-x 5 lorenzodonini staff 170 Nov 3 00:10 style
drwxr-xr-x 5 lorenzodonini staff 170 Nov 13 15:04 templates
-rw-r--r-- 1 lorenzodonini staff 798 Oct 20 23:31 test.php
-rw-r--r-- 1 lorenzodonini staff 15915 Nov 2 12:39 test_bankfunctions.php
drwxr-xr-x 4 lorenzodonini staff 136 Nov 19 17:52 uploads
```

To perform multiple transactions in one request you can upload a batch transaction file.
Format the file according to be following rules:

- Each transaction in a separate line

Figure 3.4: Output of a simple `ls -l` command injection inside the GNB application

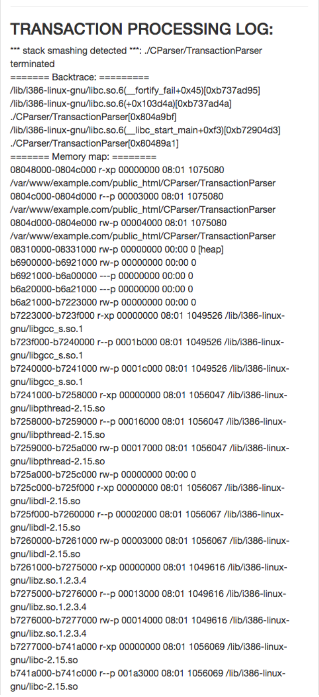
3.7.6 Testing for Buffer overflow (OTG-INPVAL-014)

	DogeBank
Observation	While testing the DogeBank application for C vulnerabilities, we found the actual code to be difficult to exploit, as inserting any kind of value inside the transaction file tags would not result in segmentation fault or other visible errors. In order to produce overflows and break the program, we had exploit the filename vulnerability, as it allowed us to pass in custom parameters to the C program. It is also important to stress that the batch transaction functionality offers a transaction log, which is displayed on client side after a request and proved to be the direct output of the C parser. This, however, includes by default only stdout messages and not stderr messages. To bypass this, we had to tamper with the transaction filename, in order to redirect stderr messages on stdout and get some feedback. More detailed observations about the found vulnerabilities are presented inside the following sections: OTG-INPVAL-014-2 , OTG-INPVAL-014-3 and OTG-INPVAL-014-4 .
Discovery	Finding out these vulnerabilities required multiple attempts. In order to find these vulnerabilities we performed manual attacks and used the Fuzz functionality of the Zed Attack Proxy tool.
Likelihood	The complexity of these attacks proved to be much higher than any other while stress-testing this application. Since we were doing Blackbox testing, we had to come up with possible combinations of strings in order to find weaknesses inside the C code. Hence an attacker could find vulnerabilities through brute-force, which makes an attack less likely.
Implication	N/A
CVSS	This vulnerability is not applicable in the application

	Goliath National Bank
Observation	The C Parser seems to be robust, as e did not find any buffer overflow vulnerability, neither due to bad string formats nor due to stack/heap overflow.
Discovery	We tried several combinations of brute force attacks, both manually and using the Fuzz functionality of the Zed Attack Proxy tool. We even tried SQL injection inside the fields of the transaction file, with no success.
Likelihood	N/A
Implication	N/A
CVSS	Secure

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

	DogeBank
Observation	We observed that it was possible to produce stack overflows by tampering with the filenames and calling the C program using custom arguments. This way we were able to generate two different kinds of messages: the program either crashed before starting the correct execution (segmentation fault) or produced a stack smashing error at some point during the transaction processing. Although in both cases a stack overflow is generated, it definitely happens in a different part of the code, since the error messages are different.
Discovery	We discovered this issue by brute-forcing arbitrary values inside the filename. More specifically, we found out that the C program would take in other arguments besides the filename (user/account id most likely). Being this parameter of fixed size, we managed to inject longer values, which could result into stack smashing. Here are two examples of filenames that generated two different errors (probably due to difference in length): • `testinput.txt hackhackhackhackhack 2>&1 #` • `textinput.txt ’0 UNION ALL SELECT tan6 from tans_lists’ 2>&1 #`
Likelihood	The attack proved to be rather difficult and exploiting the vulnerability properly also requires some skills.
Implication	Stack overflows allow an attacker to modify the program flow, for example overwriting return addresses.
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 8.5



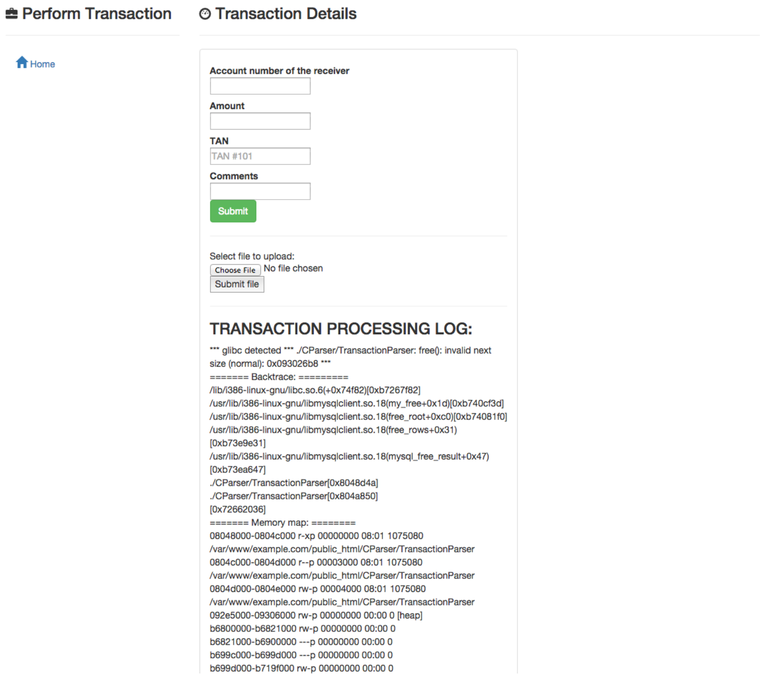
TRANSACTION PROCESSING LOG:

```
*** stack smashing detected ***: ./CParser/TransactionParser
terminated
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(__fortify_fail+0x45)[0xb737ad95]
/lib/i386-linux-gnu/libc.so.6(+0x103d4a)[0xb737ad4a]
./CParser/TransactionParser[0x804a9bf]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3)[0xb72904d3]
./CParser/TransactionParser[0x80489a1]
======= Memory map: ========
08048000-0804c000 r-xp 00000000 08:01 1075080
/var/www/example.com/public_html/CParser/TransactionParser
0804c000-0804d000 r--p 00003000 08:01 1075080
/var/www/example.com/public_html/CParser/TransactionParser
0804d000-0804e000 rw-p 00004000 08:01 1075080
/var/www/example.com/public_html/CParser/TransactionParser
08310000-08331000 rw-p 00000000 00:00 0 [heap]
b6900000-b6921000 rw-p 00000000 00:00 0
b6921000-b6a00000 ---p 00000000 00:00 0
b6a20000-b6a21000 ---p 00000000 00:00 0
b6a21000-b7223000 rw-p 00000000 00:00 0
b7223000-b723f000 r-xp 00000000 08:01 1049526 /lib/i386-linux-
gnu/libgcc_s.so.1
b723f000-b7240000 r--p 0001b000 08:01 1049526 /lib/i386-linux-
gnu/libgcc_s.so.1
b7240000-b7241000 rw-p 0001c000 08:01 1049526 /lib/i386-linux-
gnu/libgcc_s.so.1
b7241000-b7258000 r-xp 00000000 08:01 1056047 /lib/i386-linux-
gnu/libpthread-2.15.so
b7258000-b7259000 r--p 00016000 08:01 1056047 /lib/i386-linux-
gnu/libpthread-2.15.so
b7259000-b725a000 rw-p 00017000 08:01 1056047 /lib/i386-linux-
gnu/libpthread-2.15.so
b725a000-b725c000 rw-p 00000000 00:00 0
b725c000-b725f000 r-xp 00000000 08:01 1056067 /lib/i386-linux-
gnu/libdl-2.15.so
b725f000-b7260000 r--p 00002000 08:01 1056067 /lib/i386-linux-
gnu/libdl-2.15.so
b7260000-b7261000 rw-p 00003000 08:01 1056067 /lib/i386-linux-
gnu/libdl-2.15.so
b7261000-b7275000 r-xp 00000000 08:01 1049616 /lib/i386-linux-
gnu/libz.so.1.2.3.4
b7275000-b7276000 r--p 00013000 08:01 1049616 /lib/i386-linux-
gnu/libz.so.1.2.3.4
b7276000-b7277000 rw-p 00014000 08:01 1049616 /lib/i386-linux-
gnu/libz.so.1.2.3.4
b7277000-b741a000 r-xp 00000000 08:01 1056069 /lib/i386-linux-
gnu/libc-2.15.so
b741a000-b741c000 r--p 001a3000 08:01 1056069 /lib/i386-linux-
gnu/libc-2.15.so
```

Figure 3.5: Stack smashing error generated on the DogeBank `tran.php` page by uploading the file `testinput.txt hackhackhackhackhack 2>&1 #`

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

	DogeBank
Observation	Similarly as for the stack overflow, we managed to get a heap overflow by tampering with the filenames and calling the C program using custom arguments. Differently from the stack overfow, we tried using a longer string as a parameter and simultaneously performing SQL injection through it. Although we can't be sure if the SQL injection actually worked, the resulting error was caused by a free() operation. We also noticed that the one filename produced different results in time: `textinput.txt '0 UNION ALL SELECT tan6 from tans_lists' 2>&1 #` cause both a stack overflow and a heap overflow.
Discovery	Discovery made by fuzzing the values inside the batch transaction file and comparing the results we got back by the server. We also tried several argument strings manually.
Likelihood	The attack proved to be rather difficult and exploiting the vulnerability properly also requires some skills.
Implication	Heap overflows allow an attacker to modify the program flow, leading to arbitrary code execution.
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 8.5



Account number of the receiver

Amount

TAN

TAN #101

Comments

Select file to upload:

No file chosen

TRANSACTION PROCESSING LOG:

```
*** glibc detected *** ./CParser/TransactionParser: free(): invalid next
size (normal): 0x093026b8 ***
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(+0x74f82)[0xb7267f82]
/usr/lib/i386-linux-gnu/libmysqlclient.so.18(my_free+0x1d)[0xb740cf3d]
/usr/lib/i386-linux-gnu/libmysqlclient.so.18(free_root+0xc0)[0xb74081f0]
/usr/lib/i386-linux-gnu/libmysqlclient.so.18(free_rows+0x31)
[0xb73e9e31]
/usr/lib/i386-linux-gnu/libmysqlclient.so.18(mysql_free_result+0x47)
[0xb73ea647]
./CParser/TransactionParser[0x8048d4a]
./CParser/TransactionParser[0x804a850]
[0x72662036]
======= Memory map: ========
08048000-0804c000 r-xp 00000000 08:01 1075080
/var/www/example.com/public_html/CParser/TransactionParser
0804c000-0804d000 r--p 00003000 08:01 1075080
/var/www/example.com/public_html/CParser/TransactionParser
0804d000-0804e000 rw-p 00004000 08:01 1075080
/var/www/example.com/public_html/CParser/TransactionParser
092e5000-09306000 rw-p 00000000 00:00 0 [heap]
b6800000-b6821000 rw-p 00000000 00:00 0
b6821000-b6900000 ---p 00000000 00:00 0
b699c000-b699d000 ---p 00000000 00:00 0
b699d000-b719f000 rw-p 00000000 00:00 0
```

Figure 3.6: Heap error generated on the DogeBank `tran.php` page by uploading the file `textinput.txt ’0 UNION ALL SELECT tan6 from tans_lists’ 2>&1 #`

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

	DogeBank
Observation	The only issue we found when working with string formats was regarding to the length of a string. When passed a huge string, the program will simply not work as intended, probably due to buffer limitations. However, no relevant error messages were returned, hence the file is definitely not read using the *gets* function. Inserting null characters inside the file also makes the program not work as intended, but it is not possible to exploit the actual transaction functionality by doing this.
Discovery	Most of these discoveries were made by fuzzing the values inside the batch transaction file and comparing the results we got back by the server.
Likelihood	This does not prove to be an actual attack, since it doesn't break the C program by allowing to exploit the code.
Implication	N/A
CVSS	This vulnerability is not applicable in the application

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

	DogeBank
Observation	Since it is possible to upload files permanently onto the webserver and the application is already vulnerable to XSS attacks and SQL injections, incubated attacks are definitely an option.
Discovery	This vulnerability depends on the possibility to store malicious code on the server, hence it is a consequence of discovering other vulnerabilities like the upload of malicious files.
Likelihood	The likelihood of such an attack is high, although in some cases social engineering may be required. For example, performing a stored XSS attack during a registration process will result in every employee executing that particular script.
Implication	Tricking a user into executing malicious code can prove to be a very serious issue.
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 5

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	The discovery was made the same way it was for DogeBank case.
Likelihood	The same likelihood described for the DogeBank application applies.
Implication	The same implications mentioned for the DogeBank application apply.
CVSS	**AV:** N **AC:** H **PR:** L **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 5

3.7.11 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 Cryptography

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

	DogeBank
Observation	Due to the fact, that the application is only accessible via HTTP and no SSL/TLS encryption is used no testing for weak ciphers could be performed. Relevant information leakage resulting from the unencrypted communication has already been covered in OTG-CONFIG-007 and OTG-CRYPST-003 .
Discovery	N/A
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

	Goliath National Bank
Observation	The same observations made for the DogeBank application . Relevant information leakage resulting from the unencrypted communication has already been covered in OTG-CONFIG-007 and OTG-CRYPST-003 .
Discovery	N/A
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

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

	DogeBank
Observation	Due to the fact, that no encryption is used when accessing the application, no padding of information is used. Relevant information leakage resulting from the unencrypted communication has already been covered in OTG-CONFIG-007 and OTG-CRYPST-003 .
Discovery	N/A
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply. Relevant information leakage resulting from the unencrypted communication has already been covered in OTG-CONFIG-007 and OTG-CRYPST-003 .
Discovery	N/A
Likelihood	N/A
Implication	N/A
CVSS	This vulnerability is not applicable in the application

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

	DogeBank
Observation	Sensitive information is sent over unencrypted channels in every request the user performs as the application is only accessible via HTTP and no SSL/TLS encryption is used.
Discovery	As documented in OTG-CONFIG-007 , the site is not available via HTTPS.
Likelihood	Every requests gets sent over an unencrypted channel automatically.
Implication	The same implications as in OTG-CONFIG-007 apply. By sniffing the network traffic, all data exchanged between the server and the client can be read as clear text. No confidentiality at all is supported on this end.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** H **I:** H **A:** N **Score:** 6.8

	Goliath National Bank
Observation	See above
Discovery	See above
Likelihood	See above
Implication	See above
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** H **I:** H **A:** N **Score:** 6.8

3.9 Business Logic Testing

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

	DogeBank
Observation	We observed several flaws in the input validation on the DogeBank site. This includes the following fields from a business logic perspective: • Transaction Destination Account number This field is not validated. This means money can be transferred to any given account, even if it does not exist. This also enables transferring money to the same account it is coming from. This would not be an issue if this worked properly - but the amount does not get deducted from the source account in this case. Any user may generate infinite money this way. • Transaction Amount This field is not validated. This means the transaction amount can even be negative, leading to the specified amount being subtracted from the destination account. This enables "stealing" money from other people's accounts. • Transaction Number This field is not validated. The TAN is not checked for its format which always leads to a check for the given TAN in the database. In combination with the issue that there are no new TANs generated once the 100 given ones are used this leads to the strange vulnerability that the 101st transaction and all following ones do not need a TAN in order to succeed. This means that all transactions accept an empty TAN once all 100 TANs are used. There furthermore is an issue with the logic behind the approval of transactions: Transactions with an amount above 10.000 EUR are only deducted once they are approved. This means it is possible to create additional transactions during the transaction is unapproved, even if the account balance would be negative after the first transaction. Additionally transactions leading to increased negative balance can be done once the account balance is negative.
Discovery	Manual testing
Likelihood	High
Implication	This allows an attacker to increase his own account balance without limit and create transactions without the necessary TANs.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** U **C:** N **I:** H **A:** L **Score:** 7.1

	Goliath National Bank
Observation	• Transaction Destination Account number This field is not validated. This means money can be transferred to any given account, even if it does not exist. This also enables transferring money to the same account it is coming from. This would not be an issue if this worked properly - but the amount does not get deducted from the source account in this case. Any user may generate infinite money this way.
Discovery	Manual testing
Likelihood	High
Implication	This allows an attacker to increase his own account balance without limit.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** U **C:** N **I:** H **A:** N **Score:** 6.5

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

	DogeBank
Observation	We did not observe any process timing vulnerabilities at the DogeBank.
Discovery	We used a custom script to test the response time given invalid and valid usernames and calculated the median response time.
Likelihood	N/A
Implication	N/A
CVSS	Secure

	Goliath National Bank
Observation	We observed a timing attack at the Goliath National Bank. The response times differ depending on the existence of the provided mail address in the database.
Discovery	See above
Likelihood	N/A
Implication	N/A
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** N **A:** N **Score:** 5.3

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

	DogeBank
Observation	After thorough testing and observation we concluded that no mechanisms to prevent against application mis-use are in place. No critical functionalities are disabled and no logs are kept.
Discovery	These observations were made after using several different tools and manually stress-testing the application.
Likelihood	N/A
Implication	This vulnerability implicates that an attacker will be able to attempt countless attacks and abuse functionalities without any repercussion.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 7.3

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	These observations were made after using several different tools and manually stress-testing the application.
Likelihood	N/A
Implication	The same implications mentioned for the Doge application apply.
CVSS	**AV:** N **AC:** L **PR:** N **UI:** N **S:** U **C:** L **I:** L **A:** L **Score:** 7.3

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

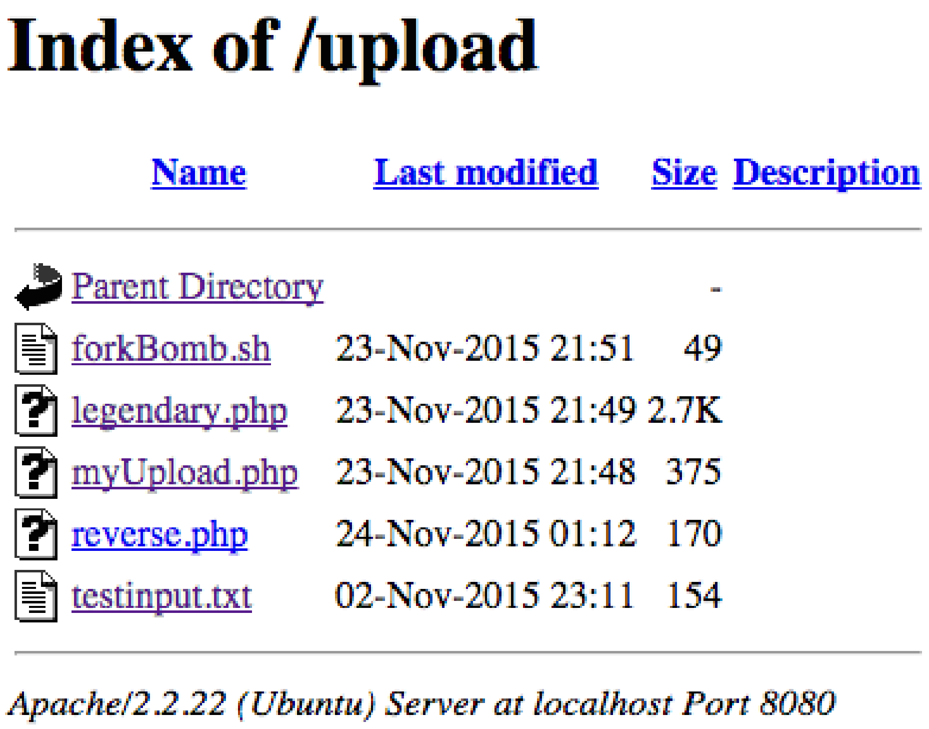
	DogeBank
Observation	We discovered that the tran.php page allows to upload any kind of file, without performing extension checks on it. However, it seems that the server accepts only files with a limited size, making it impossible to generate more than 3 transactions at once, or uploading huge files for that matter. Regardless, as long as the filesize stays below 500 bytes, any file will be accepted by the server and stored forever inside the /uploads folder.
Discovery	It is important to stress that no file format was described on the documentation nor on the transaction page. Nonetheless, after having found out the structure of the web application (using DirBuster and Burp), we found a sample batch transaction file in txt format. Afterwards we simply tried uploading files with different extensions to see the outcome.
Likelihood	The likelihood of a an attacker uploading a file with a bad filename or a non-expected extension is very high.
Implication	This was by far the most severe vulnerability we found, since all uploaded files are kept inside a well known folder on the web server. An attacker is this way able to upload any file, as well as custom scripts and programs to the server. This issue is analyzed in depth in section 009.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 9.9

	Goliath National Bank
Observation	The only page which allows to upload a file to the server is the new_transaction_multiple.php (loaded as a frame under the my_accounts.php section inside the client.php file). When uploading a file, the server performs an explicit check, eventually accepting only .csv and .txt files.
Discovery	We tried uploading several files with different file extensions, leading to the result described above.
Likelihood	N/A
Implication	N/A
CVSS	Secure

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

	DogeBank
Observation	Once discovered that the tran.php page allowed to upload any kind of file, we did multiple tests and finally observed that it was indeed possible to upload malicious code. This vulnerability was later on used to execute custom PHP scripts. Although eventually we gained full access to the application, including credentials, database access and source code, we couldn't tamper too much with the operating system since we didn't have root access.
Discovery	This discovery depends directly on the one made in section OTG-BUSLOGIC-008 . We tried multiple attacks in order to exploit this vulnerability, all of which worked without flawlessly: • Upload a php script which allowed us to start a reverse shell attack. • Upload an interactive php script which could allow us to enter commands directly or perform specific operations. • Upload a php script which could make us tamper with the database. Once uploaded, each script could simply be executed by opening the page `/upload/SCRIPTNAME`
Likelihood	Once an attacker asserted the possibility of uploading any kind of file, the likelihood of such an attack becomes very high.
Implication	As stated in the OTG-BUSLOGIC-008 section, this was the most severe vulnerability found, since it allows to get full control over the application.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 9.9

	Goliath National Bank
Observation	Although the application performs a check on the extension of the file uploaded via the new_transaction_multiple.php page, it does not check the content nor the basename of the file. Combined with the fact code injection is possible due to the same vulnerability (discussed in INPVAL 013), we observed that it is perfectly feasible to upload malicious files and rename them afterwards.
Discovery	This discovery was made thanks to manual attempts to perform code injection. More specifically, we could upload a malicious file and execute by following these steps: 1. Upload a file named `maliciousFile.txt`; 2. Upload a second file named `test';mv maliciousFile.txt maliciousFile.php;#.txt` 3. Knowing that the uploaded files are stored inside the /uploads folder, the maliciousFile could be executed opening the page `http://HOST/gnb/project/uploads/maliciousFile.php`
Likelihood	The likelihood of an attacker attempting a code injection through a file upload is very high.
Implication	The implications of a code injection attack are very severe, since an attacker could execute arbitrary code on the webserver. Even reading source code becomes possible.
CVSS	**AV:** N **AC:** L **PR:** L **UI:** N **S:** C **C:** H **I:** H **A:** H **Score:** 9.9



Index of /upload

Name	Last modified	Size	Description
Parent Directory		-	
forkBomb.sh	23-Nov-2015 21:51	49	
legendary.php	23-Nov-2015 21:49	2.7K	
myUpload.php	23-Nov-2015 21:48	375	
reverse.php	24-Nov-2015 01:12	170	
testinput.txt	02-Nov-2015 23:11	154	

Apache/2.2.22 (Ubuntu) Server at localhost Port 8080

Figure 3.7: List of uploaded files accessible via the `/upload` folder, after having uploaded some custom scripts

3.10 Client Side Testing

This section was prioritized as low, therefore the client side was not tested in depth. Furthermore, as stated in the OWASP testing guide, black box testing of the client side is usually not performed, since access to the source code is always available, as it needs to be sent to the client to be executed.

3.10.1 Testing for Clickjacking (OTG-CLIENT-009)

	DogeBank
Observation	We observed that it is entirely possible to load all of the pages inside
Discovery	This discovery required manual testing: an html with a simple iframe was included, that could contain any of the pages of the the web application.
Likelihood	Considering there is not protection against clickjacking attacks whatsoever, this kind of attack could prove to be quite easy.
Implication	The Doge web application is entirely vulnerable to clickjacking attacks and an attacker could handle all of the actions started on the php pages in a malicious way.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

	Goliath National Bank
Observation	The same observations made for the DogeBank application apply.
Discovery	This discovery required manual testing: an html with a simple iframe was included, that could contain any of the pages of the the web application.
Likelihood	The same likelihood described for the DogeBank application applies.
Implication	The GNB web application is entirely vulnerable to clickjacking attacks and an attacker could handle all of the actions started on the php pages in a malicious way.
CVSS	**AV:** N **AC:** H **PR:** N **UI:** R **S:** U **C:** L **I:** L **A:** L **Score:** 5

3.10.2 Testing WebSockets (OTG-CLIENT-010)

	DogeBank
Observation	The Doge Web Application does not make use of any asynchronous operation, neither using AJAX nor using WebSockets.
Discovery	We asserted that there was no WebSockets communication at all while surfing the pages of the application and testing out all of the functionalities. This was done using Google Chrome's Developer Tools.
Likelihood	N/A
Implication	N/A
CVSS	Secure

	Goliath National Bank
Observation	Although the application makes use of asynchronous requests for the client search functionality, this is done using traditional AJAX and not HTML5 WebSockets. Hence, the application is secure against attacks on WebSockets.
Discovery	To prove our observation, we used Google Chrome's Developer Tools to assert there was no ongoing WebSocket communication when executing search requests.
Likelihood	N/A
Implication	N/A
CVSS	Secure

4 Bugs

4.1 Transaction Batch format not documented

Location	`/tran.php`
Vulnerabilities	N/A
Discription	There is no reference on how to use the transaction batch file upload. Neither the required file extension nor the format of the file contents are mentioned on the page. It is only possible to access this information by browsing through the directories of the web application.
How to trigger	N/A

4.2 Visible TAN Code

Location	`/tran.php`
Vulnerabilities	Referenced Vulnerabilities
Discription	When inserting an invalid TAN using the batch file upload, the server will respond with an error message in which the expected valid TAN is sent back to client in clear text.
How to trigger	Upload a transaction batch file with an invalid TAN code, and you will be able to find the required TAN code.

Location	`/downloadTans.php`
Vulnerabilities	N/A
Discription	TAN codes appear to be randomly generated. However, when generating TAN codes, the user_id and the number of the code are perpended to the string. So, if a user ID was of length 5 and the number of a code of length 2, the actual length of the code, which was randomly generated, would be 15-5-2=8. An attacker could register an arbitrary number of users to make the user ID counter go up inside the database (auto-increment is used). Once the counter is reasonably high, the TAN code complexity would be reduced drastically, making the application more susceptible to brute-force attacks.
How to trigger	By writing a script that will continue to create users, and once users hit the limit, TAN codes will be predictable.

4.3 Predictable TAN Codes

4.4 Limited batch file size

Location	`/downloadTans.php`
Vulnerabilities	N/A
Discription	The size limit of a batch transaction file is of 500 bytes. This size is only sufficient to cover 3 transactions at most, and this is a non-acceptable number for this application.
How to trigger	N/A

4.5 Redirection to incorrect PHP

Location	`/employee_home.php`
Vulnerabilities	N/A
Discription	Clicking on 'Home' while logged in as an employee will redirect the user to `http:/HOST/employee_home` without the '.php', which results in a 404 error.
How to trigger	Click on 'Home' while logged in as an employee.

4.6 No more TAN codes generated after the first 100

Location	`/tran.php`
Vulnerabilities	N/A
Discription	Once a customer uses all 100 allocated TAN Codes, the application does not support generating more TAN codes for the customer, the TAN codes are only generated at the registration phase.
How to trigger	Use all 100 allocated TAN Codes.