Project Report
Make A Secure Website

Group 9

Minde Hansen

Stig Helle

Anders Mikalsen

Jari Kunnas

Contents

Overview

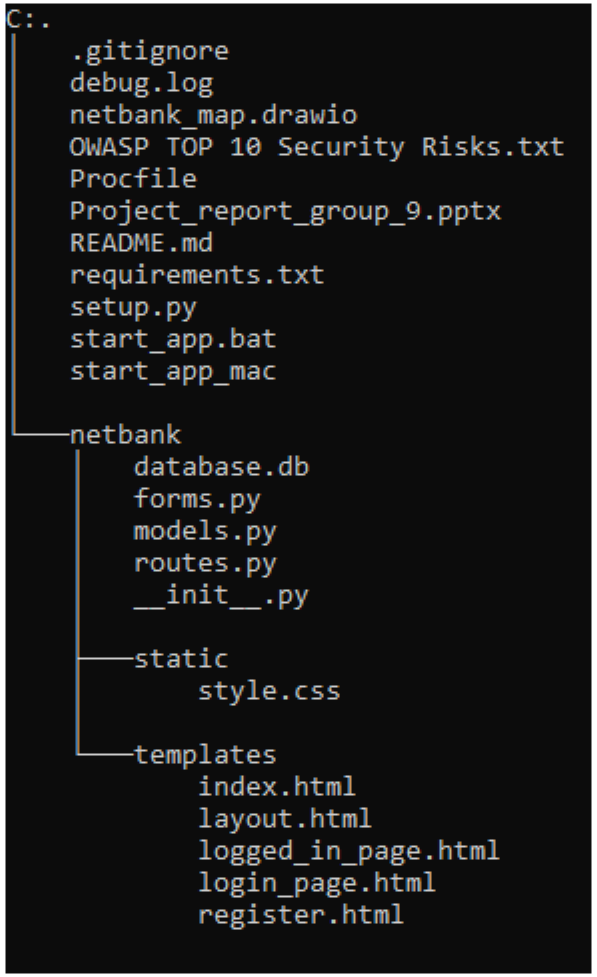
- We chose the task of creating an online banking application
- Communcation and meetings were done using Discord
- GitHub was used to create seperate branches and collaborate on the master branch for the code and other files in the project
  - **https://github.com/jkunnas58/dat250_netbank_fall_2020**
- The finished website was hosted on Heroku
  - **https://dat250-netbank.herokuapp.com**

Functionality of Netbank

- Register user with 100-1000 dollars
- Log in to personalized site.
    - View how much money you have in your account
    - View account number
    - Select account number to send to
    - Select amount to send
    - Verify sending by entering username and password again
    - Logout

Files and Folders

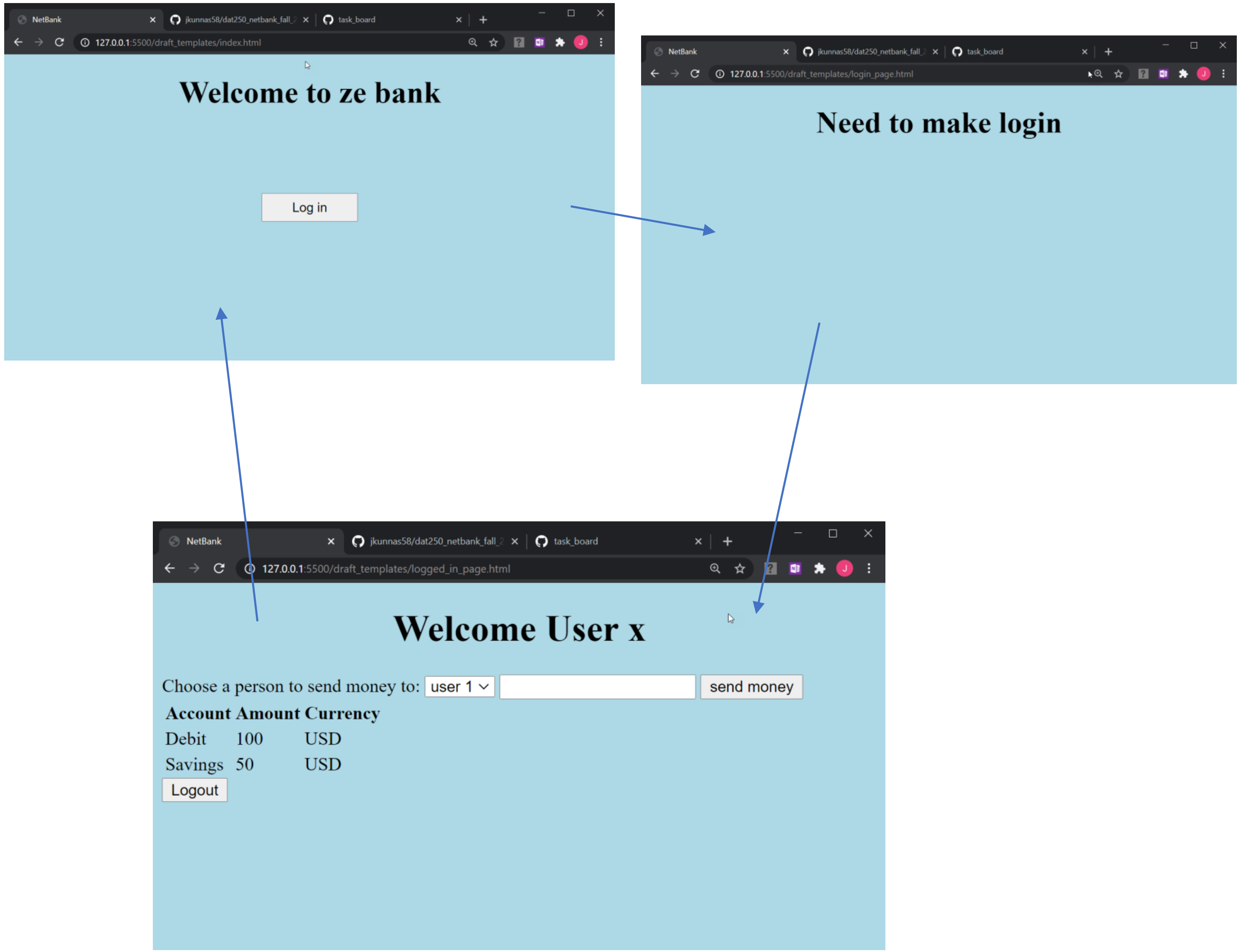
- Application can be run locally by running the “start_app.bat” file on Windows systems or use the commands in the “start_app_mac” on IOS.



```
C:.
│   .gitignore
│   debug.log
│   netbank_map.drawio
│   OWASP TOP 10 Security Risks.txt
│   Procfile
│   Project_report_group_9.pptx
│   README.md
│   requirements.txt
│   setup.py
│   start_app.bat
│   start_app_mac
│
└───netbank
    │   database.db
    │   forms.py
    │   models.py
    │   routes.py
    │   __init__.py
    │
    ├───static
    │       style.css
    │
    └───templates
            index.html
            layout.html
            logged_in_page.html
            login_page.html
            register.html
```

HTML drafts

- First HTML drafts had 3 pages. A register user page was made later when that functionality was needed.



NetBank | jkunnas58/dat250_netbank_fall | task_board

127.0.0.1:5500/draft_templates/index.html

Welcome to ze bank

Log in

NetBank | jkunnas58/dat250_netbank_fall | task_board

127.0.0.1:5500/draft_templates/login_page.html

Need to make login

NetBank | jkunnas58/dat250_netbank_fall | task_board

127.0.0.1:5500/draft_templates/logged_in_page.html

Welcome User x

Choose a person to send money to: user 1 send money

Account	Amount	Currency
Debit	100	USD
Savings	50	USD

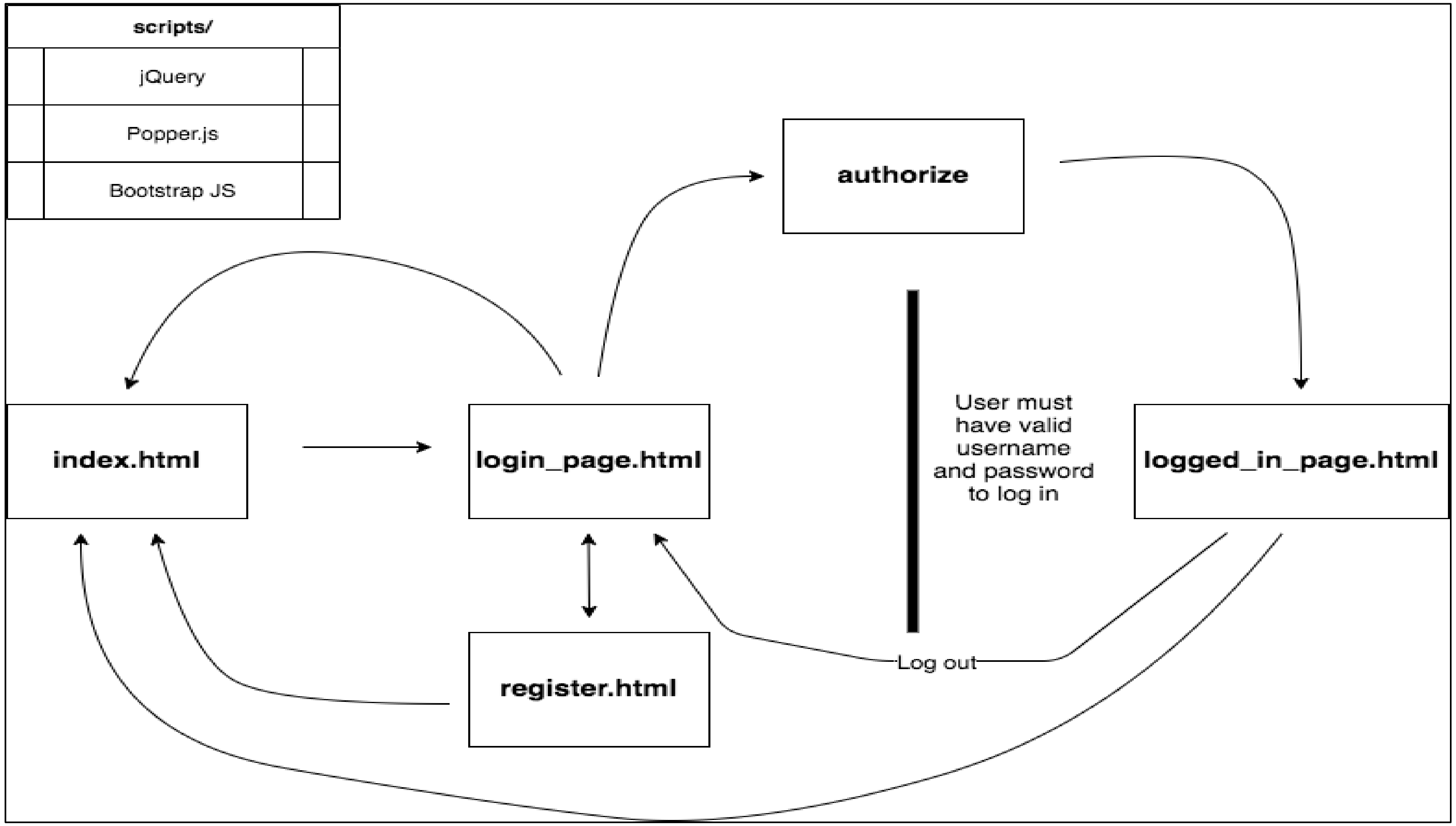
Logout

Database

- The database used is SQLite and handled via the SQLAlchemy python module.
- “database.db” is stored under the netbank folder, added to the project in the __init__.py and initiated in the routes.py script.

- For production the SQLite database was swapped with PostgreSQL because Heroku is not persistent for SQLite, but is for PostgreSQL
  - This was performed following the tutorial from Magnus Book

Site map


scripts/
jQuery
Popper.js
Bootstrap JS
authorize
index.html
login_page.html
User must have valid username and password to log in
logged_in_page.html
register.html
Log out

Threat model

- A threat model, is defined as ’’a process that reviews the security of any connected system, identifies problem areas, and determines the risk associated with each area.’’
- For our webpage, we did an initial threat model, and then continuously improved the model as new potential issues came up while developing new features.

Securing the website

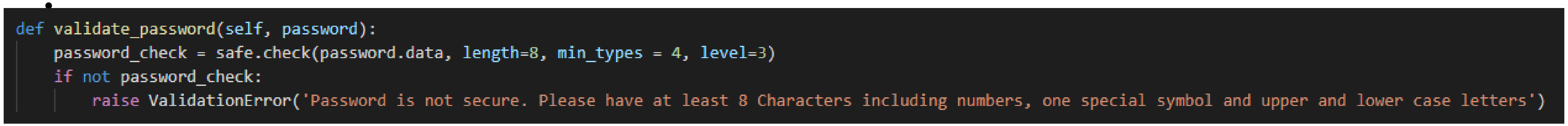
- In addition to the web application having the desired funcionality, the goal was to secure the web application against as many of the OWASP Top Ten Application Security Risks as possible
- The next slides cover the topics and how our page is protected or not against the top ten and other potential information and software security risks

OWASP A1 - Injection

- SQLAlchemy was selected as a ORM for this project. The benefit of this is that you don’t need to pass raw sql statements to the database to update and register users. SQLAlchemy will sanitize the inputs if used correctly.
- The WTForms package does not necessarily sanitize all the inputs, but the way it is set up it will limit some possibilities of what can be entered and sent to the SQLAlchemy methods.
- This project is set up to get/set user objects from the database, update those and then commit those objects to the database. This should prevent sql injection as an option to compromise this website.

OWASP A2 - Broken Authentication

- Passwords are hashed in the database using the bcrypt python module.
- Passwords have a minimum 8 character length and demand characters(lower and upper case), special symbol and numbers.
- A user can only access the login page 5 times in 5 minutes. If they try more than 5 times, they are locked out for the remaining 5 minutes. This stops brute force attacks at the login page.
- The user will be logged out if inactive for 2 minutes
- There is no notification that will let you know if you have entered a correct username, but a wrong password. This will limit the chance of brute force attacks.
- !! The website doesn't have multifactor authentication implemented. This would also decrease the risk of Broken Authentication, for this project it was deemed an extra feature we did not prioritize spending time learning how to implement!!
- We used the python safe package(https://github.com/lepture/safe) that checks for password strength=STRONG, minimum length=8 and mixed number, alpha and (not (marks or alpha)) :
    - How it works
    - **Safe** will check if the password has a simple pattern, for instance:
    1. password is in the order on your QWERT keyboards.
    2. password is simple alphabet step by step, such as: abcd, 1357
    - **Safe** will check if the password is a common used password. Many thanks to Mark Burnett for the great work on 10000 Top Passwords.
    - **Safe** will check if the password has mixed number, alphabet, marks.
    - This verification might flag some sql injection checks as unsafe, but the inputs still go through WTForms, bcrypt hashing and flask-sqlalchemy database writing so should be sql injection safe.
-



```python
def validate_password(self, password):
    password_check = safe.check(password.data, length=8, min_types = 4, level=3)
    if not password_check:
        raise ValidationError('Password is not secure. Please have at least 8 Characters including numbers, one special symbol and upper and lower case letters')
```

OWASP A3 - Sensitive Data Exposure

- Heroku has a paid option to allow for TLS on the hosted sites there. We opted not to pay for this service, but are aware of it and know it will mitigate the risk of hijacking a session or sniffing.
- We don’t share usernames of other users to the logged in user. They need to enter the account number (we use the user primary key as the identifier on where to send the money)
- We use bcrypt to hash the passwords in the database. We have limited amount of sensitive data in the database so did not see the need of encrypting the other entries, but username and account/money for a user might be an idea to encrypt.
- Additional steps that could be taken to further secure against sensitive data exposure is HTTP Strict Transport Security (HSTS).

OWASP A4 - XML External Entities (XXE)

- N/A, we don’t use any XML or have any option to insert any XML in our website. Should be protected from this security risk

OWASP A5 - Broken Access Control

- SQLinjection is not possible for logged in users either.
- Displayed information is only from the current user.

OWASP A6 - Security Misconfiguration

- Our application is quite minimalistic, with no unnecessary features or default accounts/passwords. This limits the chance of security misconfiguration.

OWASP A7 - Cross Site Scripting (XSS)

- In Flask, Jinja2 is automatically configured to escape all values unless explicitly told otherwise. This should in general rule out all XSS issues caused in templates. However, there are still other areas where one needs to be vigilant.
- Additional steps that could be taken to further secure against XSS is HTTP Strict Transport Security (HSTS).

OWASP A8 - Insecure Deserialization

- When an object is converted into a byte stream is called serialization. When the byte stream is converted into an object is called deserialization.
- JSON (JavaScript Object Notation) handles deserialization in Flask.
- The best way to avoid insecure deserialization is to avoid deserializing untrusted data. If you cannot verify and validate the data, it should not be deserialized.

OWASP A9 - Using Components with Known Vulnerabilities

- To our knowledge the components we are using, don’t have known vulnerabilities that we have misused.
- The Flask framework, Flask packages, SQLAlchemy, WTForms and safe that we have chosen is to aid in security and are commonly used for that purpose. We are not aware of any vulnerabilities with those packages if they are used properly.

OWASP A10 - Insufficient Logging & Monitoring

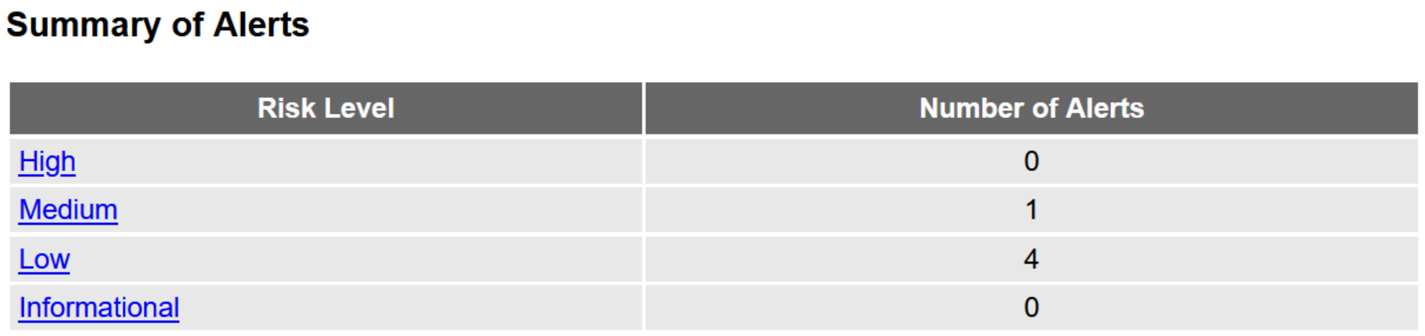
- We don’t do any explicit logging and monitoring in our application.
- Heroku, where the page is deployed, has a range of runtime logs that can be used, such as app logs, system logs, API logs and add-on logs. These are accessed through commands in the CLI.
- For paid version of Heroku hosted sites (hobby or professional dynos), there are extended metrics available.
- Also, having a transactions table that stores every movement of money would be a proper thing to have in a real bank.

Other security threats?

- Cross-Site Request Forgery (CSRF): For registration and login, we use WTForms. Flask WTForms provides among other things CSRF protection. CSRF was on the OWASP top ten in 2013, but was not included in the 2017 version.

Security testing of our application

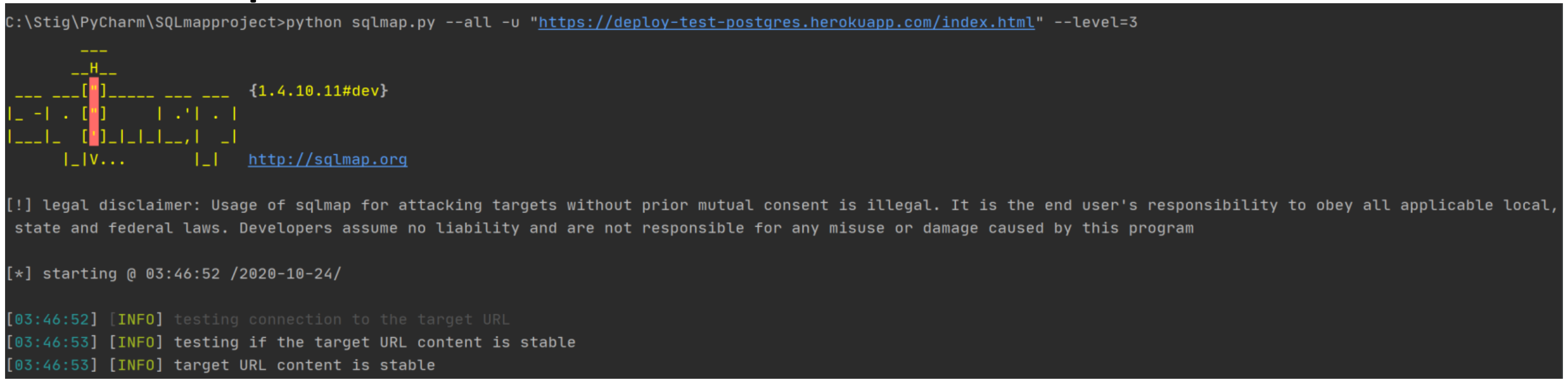
- To test security, we tried to attack our site in various ways:
- ZAP (Zed Attack Proxy) testing tool



Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	4
Informational	0

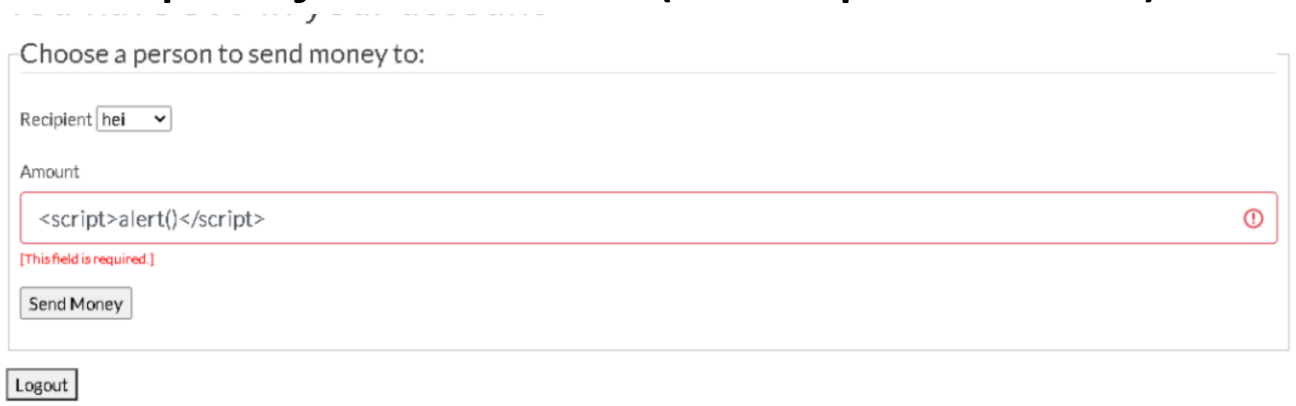
- SQLmap



```
C:\Stig\PyCharm\SQLmapproject>python sqlmap.py --all -u "https://deploy-test-postgres.herokuapp.com/index.html" --level=3
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.4.10.11#dev}
|_ -| . ["]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local,
 state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 03:46:52 /2020-10-24/

[03:46:52] [INFO] testing connection to the target URL
[03:46:53] [INFO] testing if the target URL content is stable
[03:46:53] [INFO] target URL content is stable
```

Security testing of our application

- Script injection test (example of test)



Choose a person to send money to:

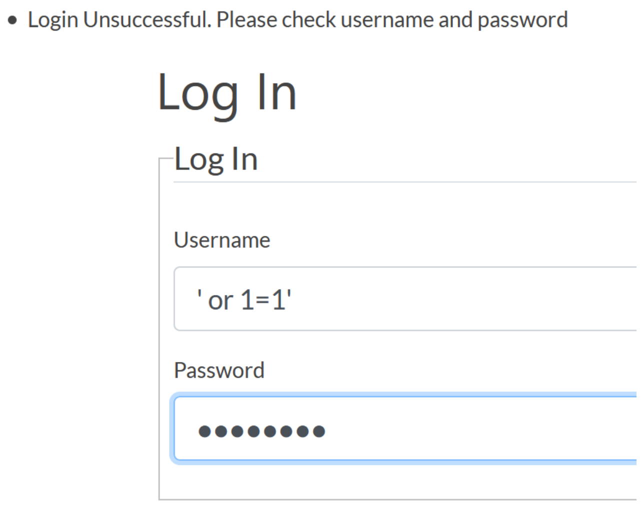
Recipient hei

Amount

<script>alert()</script>

[This field is required.]

- SQL injection
- Example of test



- Login Unsuccessful. Please check username and password

Log In

Log In

Username

' or 1=1'

Password

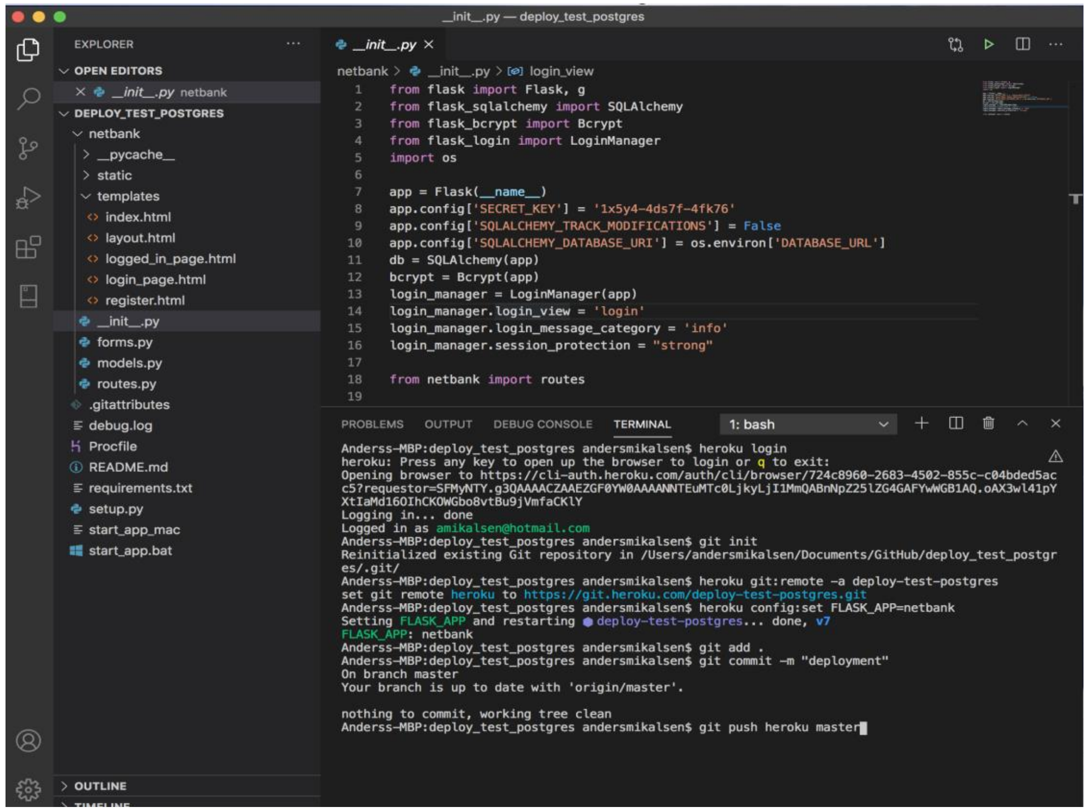
●●●●●●●●

Reflections concerning security

- We ran various security tests that all resulted positive as in being secure.
- We have also gone through all the steps and functionality we have in our page to see if there are any weakness.
- We have used well known packages and modules that protects against the most common security challenges. They were chosen from the start so we could follow the "build security in" thinking.
- We made sure we started the coding and development with the security in mind for every extra feature that was added.

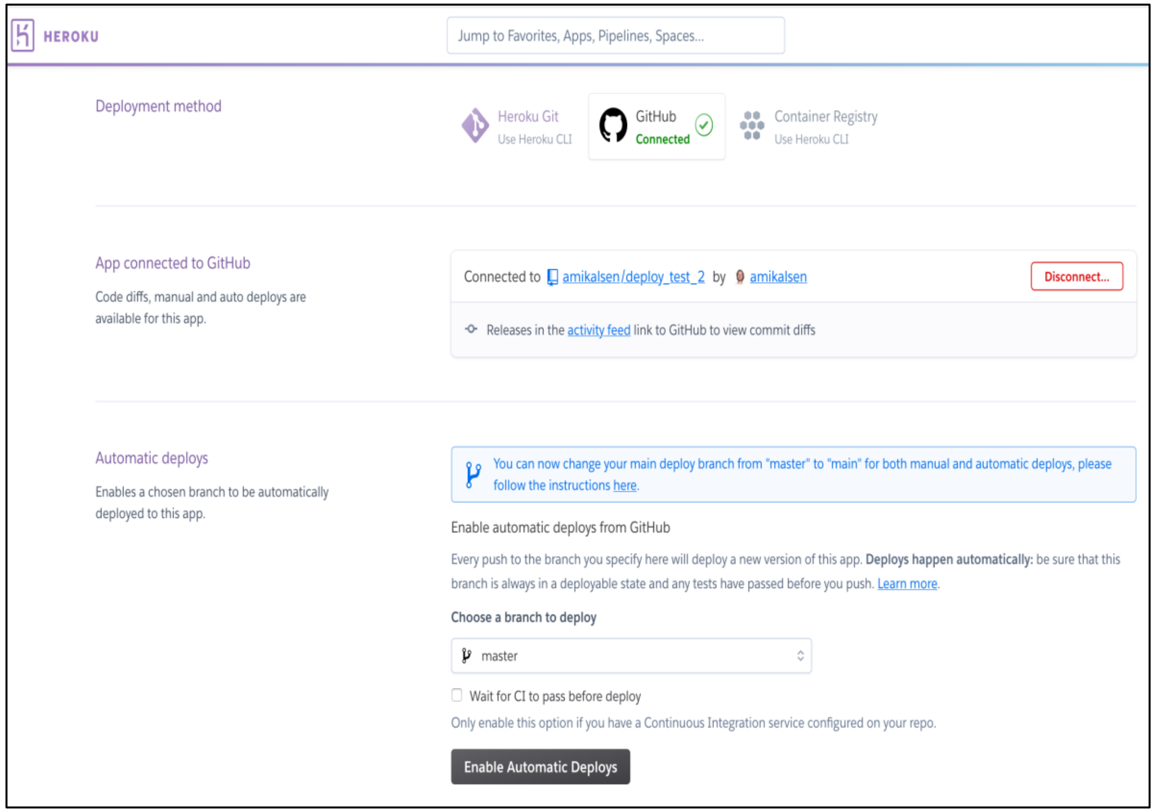
Deployment to Heroku

Testing deployment using both Heroku CLI and connecting to GitHub on Heroku.com. Final deployment was done using CLI.



```python
from flask import Flask, g
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt
from flask_login import LoginManager
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = '1x5y4-4ds7f-4fk76'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ['DATABASE_URL']
db = SQLAlchemy(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'login'
login_manager.login_message_category = 'info'
login_manager.session_protection = "strong"

from netbank import routes

```

```
Anderss-MBP:deploy_test_postgres andersmikalsen$ heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/724c8960-2683-4502-855c-c04bded5ac
c5?requestor=SFMyNTY.g3QAAAACZAAEZGF0YW0AAAANNTEuMTc0LjkyLjI1MmQABnNpZ25lZG4GAFYwWGB1AQ.oAX3wl41pY
XtIaMd16OIhCKOWGbo8vtBu9jVmfaCKlY
Logging in... done
Logged in as amikalsen@hotmail.com
Anderss-MBP:deploy_test_postgres andersmikalsen$ git init
Reinitialized existing Git repository in /Users/andersmikalsen/Documents/GitHub/deploy_test_postgr
es/.git/
Anderss-MBP:deploy_test_postgres andersmikalsen$ heroku git:remote -a deploy-test-postgres
set git remote heroku to https://git.heroku.com/deploy-test-postgres.git
Anderss-MBP:deploy_test_postgres andersmikalsen$ heroku config:set FLASK_APP=netbank
Setting FLASK_APP and restarting ⬢ deploy-test-postgres... done, v7
FLASK_APP: netbank
Anderss-MBP:deploy_test_postgres andersmikalsen$ git add .
Anderss-MBP:deploy_test_postgres andersmikalsen$ git commit -m "deployment"
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
Anderss-MBP:deploy_test_postgres andersmikalsen$ git push heroku master
```



Deployment method

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to amikalsen/deploy_test_2 by amikalsen

Releases in the activity feed link to GitHub to view commit diffs

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions here.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

master

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.