

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



SZABLON APLIKACJI WEB - UWIERZYTELNIENIE WIELOETAPOWE

*wykorzystanie mechanizmu uwierzytelnienia wieloetapowego we własnej aplikacji i analiza
wektorów ataku*

Zespół X:

BOGDANOVIČ ALBERT 307870

KOWALCZYK JAKUB 304168

RUCIŃSKI DAWID 304212

OCENIAJĄCY

dr inż. Robert Kurjata

Warszawa 2022

►Projekt finalny

Projekt został wykonany głównie przy użyciu języka Python, z wykorzystaniem biblioteki Flask. Strona “front-endowa” została przygotowana przy użyciu języka HTML oraz CSS. Ponieważ strona wizualna projektu nie jest kluczowa, użyto podstawowych technik wizualnych. Baza danych została przygotowana za pomocą MySQL, w tym też języku kierowane są zapytania do niej za pomocą modułu SQLAlchemy, co zapewnia bezpieczeństwo poprzez sanityzację danych wejściowych pobieranych od użytkownika lub potencjalnego atakującego.

Mechanizm działania:

Przy użyciu MySQL DB będzie utworzona baza danych, do której będziemy się odwoływali podczas próby rejestracji/logowania użytkownika. Do zahashowania hasła, jako szczególnie wrażliwego elementu, użyliśmy SHA-256 - o wiele bezpieczniejszego od MD5, który jest osiągalny do pokonania metodą brute-force np. poprzez znalezienie kolizji poprzez komputery domowe.

Podczas rejestracji odbywa się weryfikacja elementów wprowadzonych przez użytkownika przy użyciu modułu wtforms.validators.

Po udanej rejestracji użytkownik otrzymuje wygenerowany kod (Shared Secret), który ma wprowadzić do Google Authenticator’a, dzięki czemu możemy powiązać logującego się użytkownika z drugim faktorem autoryzacji na jego urządzeniu. Sesja użytkownika jest przechowywana w możliwie trudnym do przechwycenia ciasteczku, które wraz z sesją ma swoją datę ważności.

Przy każdej próbie logowania po poprawnym podaniu standardowych poświadczeń zostaje wyświetlony monit o podanie kodu z drugiego faktora. Naszym zamiarem jest generowanie kodów TOTP, zatem oba urządzenia muszą mieć zsynchronizowane zegary. Wymaga to zatem, by urządzenie z drugim faktorem było co jakiś czas podłączone do Internetu, by utrzymywać poprawność obliczania kodu. W przypadku zgodności zarówno standardowych poświadczeń, jak i kodu, użytkownik dostaje dostęp do aplikacji.

Biorąc pod uwagę sposób wyliczania kodu jednorazowego, krytyczne dla naszego projektu było bezpieczne przekazanie Shared Secret w obie strony, gdyż na jego podstawie jest generowany kod uwierzytelniający. Po wykradzeniu danych poprzez np. phishing, wydobyć bazy danych, czy przy pomocy innej podatności związanej z urządzeniem końcowym klienta, logowanie nieuprawnionej osoby jest w pełni możliwe. Ponieważ komunikacja jest prowadzona z użyciem HTTPS zaszyfrowanym z użyciem możliwie wysokiej wersji TLS, przekazanie kodu jest bezpieczne. Jest także jednorazowe, co ogranicza zainicjowanie generacji kodu przez atakującego.

Jako drugi faktor, po standardowych poświadczeniach logowania, został wykorzystany Google Authenticator, ze względu na dużą powszechność tej platformy i reputacji, jaką cieszy się firma tworząca to rozwiązanie, a także bogatego zaplecza w dziedzinie cyberbezpieczeństwa, zatem na obecną chwilę można stwierdzić, że to bezpieczny środek do weryfikacji dwuetapowej. Stosowane powszechnie wiadomości tekstowe są o wiele łatwiejsze do przechwycenia, poprzez ataki takie jak SIM swapping, zatem uważamy, że zaproponowana przez nas metoda sprawdzi się lepiej pod kątem cyberbezpieczeństwa, wymaga jednak ona posiadania smartfona, co może stanowić barierę dla mniej zaawansowanych technicznie osób.

► Użyte narzędzia

- elementy mikro web frameworku **Flask** do tworzenia backendu serwera
- **forms** do tworzenia formularzy rejestracji oraz logowania
- **wtforms** (StringField, PasswordField, SubmitField, BooleanField) do przesyłania danych pomiędzy formularzami a programem
- **wtforms.validators** (DataRequired, Length, Email, EqualTo) do sprawdzania czy dane podane są prawidłowe
- **flask_sqlalchemy** do komunikacji z bazą danych
- **werkzeug.security** do hashowania passwordów
- **uuid** do generacji 128 bitowego id użytkownika
- **pyotp** do generacji Shared Secret i ciasteczka
- **timedelta** do odmierzania czasu ważności sesji i ciasteczka
- podstawowe narzędzia składni języka Python

► Analiza ryzyka i przyjęte modele zabezpieczeń

Dla każdego projektu należy przeprowadzić analizę ryzyka teleinformatycznego, ocenić niezbędne zabezpieczenia, zrealizować te, które są niezbędne z punktu widzenia analizy ryzyka, opisać.

Ryzyko: SQL Injection

Model zabezpieczenia:

Parameter binding

```
s = sqlalchemy.sql.text("SELECT * FROM users WHERE users.id = :e")
result = mysql.engine.execute(s, e=id).fetchall()
```

Innym sposobem zapobiegania iniekcjom SQL jest użycie ORM, który wykonuje oczyszczanie danych wejściowych za użytkownika. SQLAlchemy słynie z mapera obiektowo-relacyjnego (ORM), za pomocą którego można odwzorować klasy na bazę danych, co pozwala od początku rozwijać model obiektowy i schemat bazy danych w sposób czysto odspzężony.

Wartość ciągu znaków jest przekazywana jako parametr zapytania, dzięki czemu użytkownik jest chroniony przed wstrzyknięciem kodu SQL.

```
cookie = request.cookies.get('token')
user = Users.query.filter_by(sec_factor_cookie=cookie).first()
```

Ryzyko: CSRF

Model zabezpieczenia:

FlaskForm (wbudowane CSRF protection)

Zapewnia bezpieczną sesję oraz od razu implementuje ochronę CSRF (Cross-site request forgery).

Oznacza to, że program nie jest podatny na nieświadome wysyłanie żądań spreparowanych przez inne osoby.

W wypadku gdy, użytkownik zostanie oszukany i wejdzie na spreparowaną stronę i tam spróbuje się zalogować to bez tego zabezpieczenia wysłane pliki cookies wystarczyłyby do zalogowania na prawidłowej stronie. FlaskForm zabezpiecza to przez CSRF token – losowy, nie do odgadnięcia string, by za jego pomocą sprawdzać czy żądanie pochodzi z prawidłowej strony.

```
app = Flask(__name__)
app.config['SECRET_KEY'] = '5791628bb0b13ce0c676dfde280ba245'
```

```
<form method="POST" action="">
    {{ form.hidden_tag() }}
```

Chroni to przed wykorzystywaniem przez wroga osoby, niebezpiecznych żądań jak HTTP POST.

wtforms.validators:

-DataRequired:

Sprawdza, czy dane podane przez użytkownika mają wartość 'true'. Jeśli podamy pusty string DataRequired wykryje to i zatrzyma ciąg sprawdzania danych.

```
password = PasswordField('Password', validators=[DataRequired()])
```

-Length:

Sprawdza, czy dany string odpowiada minimalnym i maksymalnym wymiarom.

```
class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
```

-Email:

Sprawdza, czy adres email podany przez użytkownika spełnia wymagania (sprawdza składnię, blokuje niebezpieczne znaki i normalizuje adresy email).

```
email = StringField('Email',
                    validators=[DataRequired(), Email()])
```

-EqualTo:

Sprawdza, czy wartości dwóch pól zgadzają się (używany przy potwierdzaniu hasła)

```
confirm_password = PasswordField('Confirm Password',
                                  validators=[DataRequired(), EqualTo('password')])
```

Ryzyko: Fuzzing, kradzież sesji

Model zabezpieczenia: ustawienie czasu ważności sesji

```
@app.before_request
def make_session_permanent():
    session.permanent=True
    app.permanent_session_lifetime = timedelta(minutes=15)
```

Ryzyko: Przechwytywanie ruchu (odczytanie ciasteczek), brute force

Model zabezpieczenia: Flaga HttpOnly

```
res.set_cookie('token', value=cookie_value, secure=True, httponly=True, max_age=300)
```

Ryzyko: Hijacking session

Model zabezpieczenia: HTTPS: Zastosowanie protokołu HTTPS zapewnia szyfrowanie SSL/TLS w całym ruchu sesji. Napastnicy nie są w stanie przechwycić jawnego identyfikatora sesji, nawet jeśli ruch ofiary monitorowany.

```
app.run(ssl_context=('cert.pem', 'key.pem'))
```

Użycie cookies:

Użycie cookies zgodnie z ogólnie przyjętymi zasadami bezpieczeństwa informatycznego oraz treściami prezentowanymi na wykładzie jest minimalne. Są one przesyłane z użyciem HTTPS, a data ważności ciasteczek jest ograniczona, by utrudnić możliwe przejęcie sesji. Użytkownik po poprawnym podaniu danych z pierwszego faktora otrzymuje ciasteczko – 32 – znakowy token, który jest jednocześnie zapisywany w bazie. Na tej podstawie otrzymuje on upoważnienie do podania kodu TOTP z drugiego faktora. Długość tokenu jest na tyle duża, by zgadnięcie go metodą brute-force w ciągu pięciu minut (tyle wynosi czas ważności) było niemożliwe. Czas ten jest także dostatecznie długi, by nie powodować niepotrzebnej irytacji u użytkownika, który będzie potrzebował więcej czasu na odnalezienie kodu.

Po poprawnym logowaniu token jest kasowany, gdyż pełni on jedynie tymczasową rolę, użytkownik dostaje identyfikator sesji przy pomocy domyślnego, zaimplementowanego już mechanizmu. Co prawda, treść ciasteczka jest możliwa do podejrzenia i edycji (jest zakodowany w Base64), ale identyfikator użytkownika jest **128** bitową liczbą, dlatego zgadnięcie tego identyfikatora byłoby niemożliwe w sposób niepostrzeżony. W rzeczywistym środowisku należałoby wykrywać taką sytuację poprzez logowanie. Po kilkunastu próbach można by ograniczyć dostęp użytkownika. Sama sesja jest permanentna, ale ma czas ważności ograniczony do 15 minut, by ograniczyć możliwość kradzieży sesji

► Instrukcja jak kompilować, uruchomić, korzystać

1. Pobrać na PC:

- *Mysql*: <https://dev.mysql.com/downloads/installer/>
- *python3*
- *wymagane moduły*

```
from datetime import timedelta

import pyotp
import uuid
import sqlalchemy.sql
from flask import Flask, render_template, url_for, flash, redirect, request, make_response, session
from flask_sqlalchemy import SQLAlchemy

from flask_migrate import Migrate
from flask_login import LoginManager, UserMixin, login_user, current_user, logout_user, login_required
from werkzeug.security import generate_password_hash, check_password_hash
from forms import RegistrationForm, LoginForm
import getpass
from sqlalchemy_utils.functions import database_exists
```

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, BooleanField
from wtforms.validators import DataRequired, Length, Email, EqualTo
```

2. Na smartfona pobrać:

- *Google Authenticator*:

<https://play.google.com/store/apps/details?id=com.google.android.apps.authenticator2&hl=pl&gl=US>

albo

- *Microsoft Authenticator*:

<https://play.google.com/store/apps/details?id=com.azure.authenticator&hl=pl&gl=US>

3. Uruchomić skrypt app.py

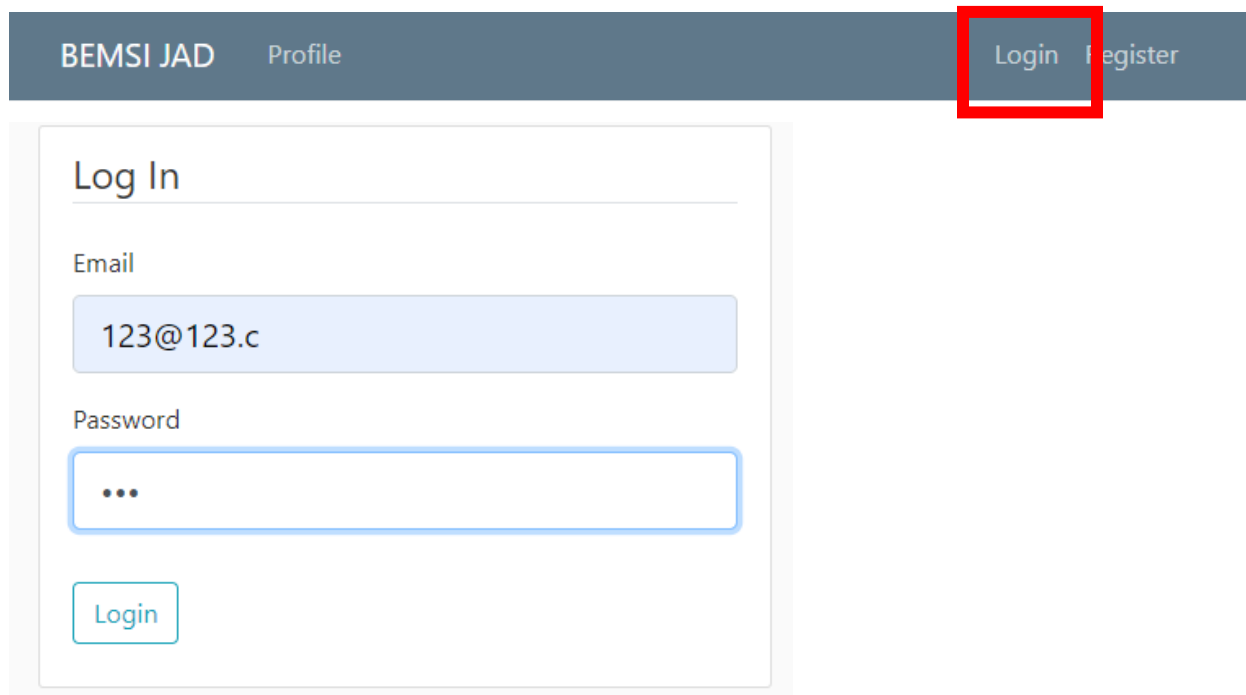
- Do terminalu wpisać hasło lokalnego mysql serwera
- W przeglądarce wywołać <https://127.0.0.1:5000>

4. W zakładce Register zarejestrować swego użytkownika:

The screenshot shows a web application interface for user registration. At the top, a dark blue navigation bar contains the text 'BEMSI JAD' and 'Profile' on the left, and 'Login' and 'Register' on the right. The 'Register' link is highlighted with a red rectangular border. Below the navigation bar is a light gray box containing the registration form. The form is titled 'Join Today' in a large, dark font. It consists of four input fields: 'Username' with the value '123', 'Email' with the value '123@123.c', 'Password' with three dots indicating a masked password, and 'Confirm Password' also with three dots. A blue 'Sign Up' button is located at the bottom left of the form area.

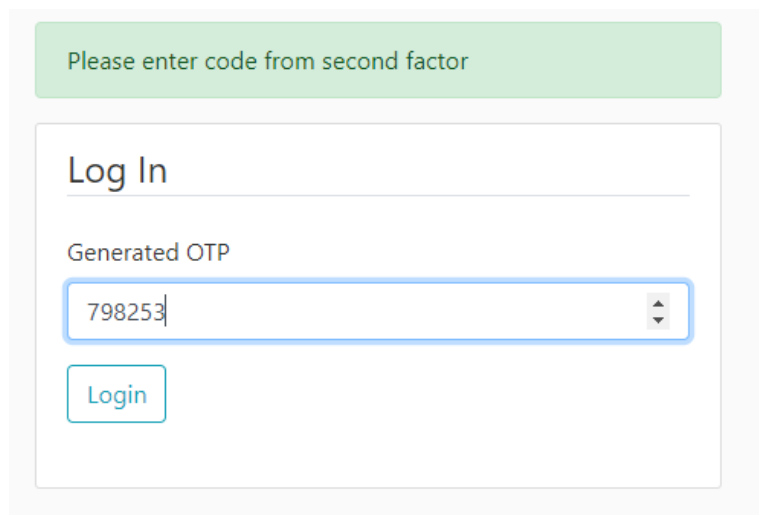
PO UDANEJ REJESTRACJI ZOSTANIE WYGENEROWANY GA TOKEN. NALEŻY GO WPROWADZIĆ DO AUTHENTICATORA.

5. W zakładce login zalogować swego użytkownika:



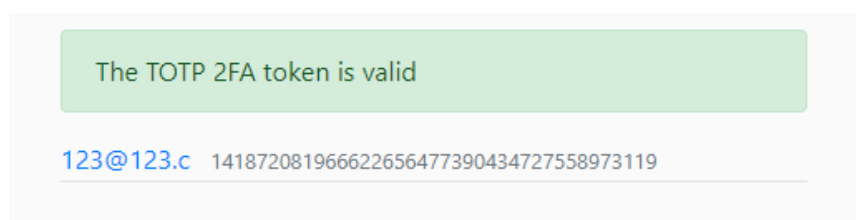
The screenshot shows a web interface with a dark blue header bar. On the left, it says 'BEMSI JAD' and 'Profile'. On the right, there are two buttons: 'Login' and 'Register'. The 'Login' button is highlighted with a red rectangular box. Below the header, there is a white box titled 'Log In'. Inside this box, there are two input fields: 'Email' with the value '123@123.c' and 'Password' with three dots indicating a masked password. Below these fields is a blue 'Login' button.

Zostanie poproszony kod OTP generowany w Autentyfikatorze:



The screenshot shows a green notification bar at the top that says 'Please enter code from second factor'. Below it is the same 'Log In' box as in the previous screenshot. The 'Email' field still contains '123@123.c'. The 'Generated OTP' field now contains the number '798253'. The 'Password' field is still masked with dots. The blue 'Login' button is still present at the bottom.

Jeżeli kod zostanie wprowadzony poprawnie użytkownik zostanie przekierowany do swego profilu z mailem i swoim 128 bitowym id:



The screenshot shows a green notification bar at the top that says 'The TOTP 2FA token is valid'. Below it, the user's email '123@123.c' is displayed in blue text, followed by a long 128-bit ID '141872081966622656477390434727558973119'.