

# Zadanie 7

I-UPB

29. novembra 2021

## Obsah

<b>1</b>	<b>Tvorba WSGI servera</b>	<b>3</b>
<b>2</b>	<b>Webová aplikácia</b>	<b>3</b>
<b>3</b>	<b>Konfigurácia</b>	<b>3</b>
3.1	Spustenie . . . . .	3
3.2	Spustenie servera . . . . .	4
<b>4</b>	<b>Sledovanie komunikácie</b>	<b>4</b>
<b>5</b>	<b>Poznámky</b>	<b>5</b>
<b>6</b>	<b>Zdroje</b>	<b>5</b>

## Tvorba WSGI servera

Server je vytvorený pomocou nástroja [gunicorn](#).

## Webová aplikácia

Použil som formulár na registráciu z predošlého zadania, takto si môžem odchytiť aj nejaký zaujímavejší traffic. Aplikácia je jednoduchá, používateľ sa zaregistruje, aplikácia skontroluje záznam v databáze pri prihlásení.

## Konfigurácia

Pred inštaláciou dotiahnuť pip packages:

```
1 pip install -r requirements.txt
```

Keďže som sa rozhodol pre self-signed certifikáty, pred spustením ich treba vytvoriť. Urobím tak jednoducho cez [openssl](#).

```
1 openssl req -x509 -newkey rsa:4096 -nodes -out  
2   cert.pem -keyout key.pem -days 365
```

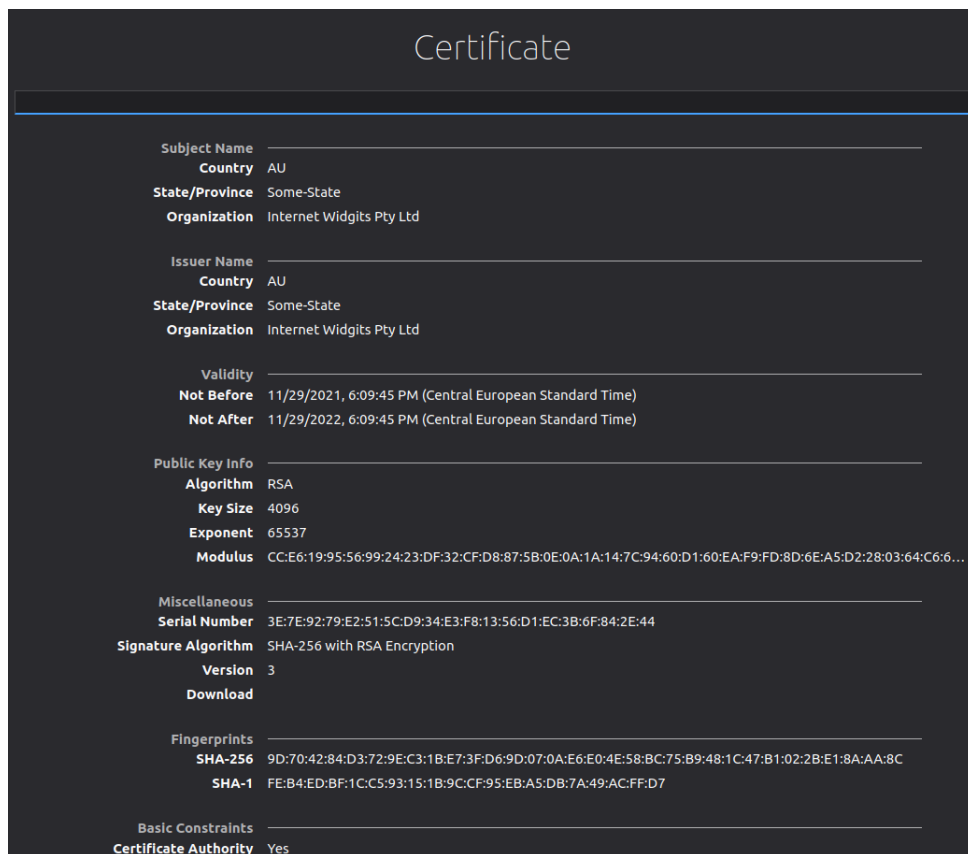
Vysvetlenie príkazu:

- [req](#) - spracovanie requestu pre certifikát, tvorba self-signed certifikátu
- `newkey` - generuj nový kľúč
- [nodes](#) - možnosť pri vytvorení cert.pem aj key.pem v jednom riadku
- `out` - výstupný súbor
- `days` - dĺžka validácie kľúča

## Spustenie

```
1 flask run --cert=cert.pem --key=key.pem
```

Skontrolujeme platnosť certifikátu na adrese <https://127.0.0.1:5000/> nasledovné:



Obr. 1: Príklad vygenerovaného certifikátu

## Spustenie servera

```
1 gunicorn --certfile cert.pem --keyfile key.pem -w 4 -b 0.0.0.0:8080 app:app
```

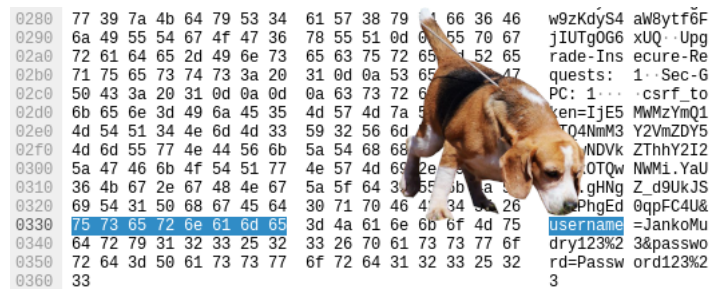
Server je nakonfigurovaný na porte **8080** a posledným arguntom je modul `app`, teda prepojenie s Flaskom. Certifikát ako aj key sú definované aj pri spustení `app.py`.

## Sledovanie komunikácie

Prihlásim sa ako **JankoMudry123!** a moje heslo bude **Password123!** (nechodil do školy na UPB, tak nevie že to nieje dobré heslo). Skúsime túto správu odchytiť cez [wireshark](#). Záznam už existuje v `database.db`.

286	355.369343389	127.0.0.1	127.0.0.1	TCP	66	58980 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=10606254...
287	355.369506291	127.0.0.1	127.0.0.1	HTTP	865	POST /login HTTP/1.1 (application/x-www-form-urlencoded)
288	355.369516441	127.0.0.1	127.0.0.1	TCP	66	8080 → 58980 [ACK] Seq=1 Ack=800 Win=64768 Len=0 TSval=106062...
289	355.382759331	127.0.0.1	127.0.0.1	TCP	587	8080 → 58980 [PSH, ACK] Seq=1 Ack=800 Win=65536 Len=521 TSval=...
290	355.382776900	127.0.0.1	127.0.0.1	TCP	66	58980 → 8080 [ACK] Seq=800 Ack=522 Win=65024 Len=0 TSval=1060...
291	355.382793413	127.0.0.1	127.0.0.1	HTTP	293	HTTP/1.1 302 FOUND (text/html)

Obr. 2: Zachytenie loginu pomocou wireshark



Obr. 3: Odhalený HTTP login packet

Vidím v plain texte prihlasovacie meno ako aj heslo.

## Poznámky

Celý projekt je dostupný aj na <https://github.com/behindbone/upb-08>.

## Zdroje

- <https://www.educba.com/flask-https/>
- <https://stackoverflow.com/questions/62649807/serve-flask-application-with-https>
- <https://www.digicert.com/kb/ssl-support/openssl-quick-reference-guide.htm>
- <https://blog.didierstevens.com/2020/12/14/decrypting-tls-streams-with-wire>
- <https://www.datasciencelearner.com/how-to-deploy-flask-app-using-nginx-and-https/>