Ochrona Danych - Podsumowanie Zadań 1-5

Przedmiot: Ochrona Danych

Data: 27 listopada 2025

Student: Paweł

Cel Projektu

Zbudować bezpieczną architekturę web aplikacji z:

- Szyfrowaniem HTTPS (SSL/TLS)
- Nginx jako reverse proxy
- Flask + Gunicorn (production WSGI server, **nie dev server**)
- Obsługą uprawnień procesów
- Detekcją rzeczywistego IP klienta

Zadanie 1: Wygenerować Certyfikat Samopodpisany

Co robić:

```
bash generate_cert.sh
```

Wynik:

- `certs/server.crt` - 1302 bytes (certyfikat)
- `certs/server.key` - 1704 bytes (klucz prywatny)
- Ważny przez 365 dni
- RSA 2048-bit, SHA256

Znaczenie: Szyfrowanie HTTPS wymaga certyfikatu. Samopodpisany wystarczy do testów.

Zadanie 2: Skonfigurować Nginx dla HTTPS

Architektura:

```
Klient (HTTPS :443) ←→ Nginx (SSL terminator)
       ↓
HTTP :80 → redirect 301 → HTTPS :443
```

Kroki:

Terminal 1 (WSL):

```
sudo killall nginx
sudo bash run_nginx.sh
```

Terminal 2 (PowerShell):

```
# Test HTTP → HTTPS redirect (301)
curl -I http://localhost/
# → HTTP/1.1 301 Moved Permanently
# → Location: https://localhost/

# Test HTTPS (będzie 502, bo Flask nie działa)
curl -k https://localhost/
# → <html><h1>502 Bad Gateway</h1></html>
```

Znaczenie:

- Flaga `-k` = ignoruj warning o samopodpisanym certyfikacie
- 301 status = permanent redirect
- 502 = nginx czeka na backend (normalnie na tym etapie)

Konfiguracja:

- Port 80: HTTP redirect do HTTPS
- Port 443: SSL/TLS, proxy do `127.0.0.1:8000`
- Sertyfikaty: `./certs/server.crt` i `./certs/server.key`

Zadanie 3: Nginx jako Proxy do Flask/Gunicorn

WAŻNE: Flask musi być na Gunicorn** (production WSGI), NIE na dev serverze!

Architektura:

```
Klient → Nginx (SSL, reverse proxy) → Gunicorn (127.0.0.1:8000) → Flask (app)
```

Kroki:

Terminal 1 (WSL) - Nginx:

```
sudo bash run_nginx.sh
```

Terminal 2 (WSL) - Flask:

```
export PATH="/home/pawel/.local/bin:$PATH"
cd '/mnt/d/stud/sem 5/OchronaDanych/7'
bash run_flask.sh
```

Terminal 3 (PowerShell) - Test:

```
# Endpoint główny
curl -k https://localhost/

# Health check
curl -k https://localhost/health
```

Wynik - JSON:

```
{
  "status": "ok",
  "https": true,
  "message": "Flask application running with Gunicorn through Nginx SSL proxy",
  "client_ip": "127.0.0.1",
  "process_info": {...},
  "headers": {...},
  "server_info": {...}
}
```

Znaczenie:

- Gunicorn to production server (stabilny, skalujący)
- Flask dev server NIE powinien być w production
- Nginx oddala Flask od dostępu bezpośredniego (bezpieczeństwo)

Zadanie 4: Uprawnienia Procesu Flask

Polecenie:

```
curl -k https://localhost/process-info
```

Wynik:

```
{
  "uid": 1000,
  "gid": 1000,
  "username": "pawel",
  "current_user": "pawel",
  "home": "/home/pawel",
  "groups": [4,20,24,25,27,29,30,44,46,116,1000]
}
```

Co to oznacza:

| Pole | Wartość | Znaczenie |

|------|---------|-----------|

| `uid` | 1000 | Zwykły user (NOT root/0) |

| `gid` | 1000 | Grupa użytkownika |

| `username` | pawel | Nazwa użytkownika |

| `groups` | [...] | Grupy do których należy |

Znaczenie:

- Flask działa jako zwykły user `pawel`, **nie** jako `root`
- To bezpieczne - ogranicza szkody w razie breakscha
- Root (uid=0) ma nieograniczone uprawnienia - ryzyko!

Zadanie 5: Odczytanie Rzeczywistego IP Klienta

Problem: W architekturze proxy, Flask widzi IP = localhost, ale rzeczywisty klient jest zdalnie.

Rozwiązanie: Nginx dodaje nagłówek `X-Forwarded-For` z rzeczywistym IP.

Polecenie:

```
curl -k https://localhost/client-ip
```

Wynik:

```
{
  "client_ip": "127.0.0.1",
  "remote_addr": "127.0.0.1",
  "x_forwarded_for": "127.0.0.1",
  "x_real_ip": "127.0.0.1"
}
```

Co to oznacza:

| Pole | Wartość | Źródło |

|------|---------|--------|

| `client_ip` | 127.0.0.1 | Detected z `X-Forwarded-For` |

| `remote_addr` | 127.0.0.1 | Połączenie do Flaska (localhost) |

| `x_forwarded_for` | 127.0.0.1 | Nagłówek od Nginx'a |

| `x_real_ip` | 127.0.0.1 | Nagłówek od Nginx'a |

Znaczenie:

- `remote_addr` = IP który się podłączy do Flaska (tu: localhost bo proxy)
- `X-Forwarded-For` = rzeczywisty IP klienta (przekazany przez Nginx)
- **ProxyFix middleware** w Flasku parsuje te nagłówki
- W produkcji: `client_ip` będzie rzeczywistym IP z Internetu

W nginx.conf:

```
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

Technologia Stack

| Komponent | Rola | Port |

|-----------|------|------|

| **Nginx** 1.18.0 | Reverse proxy, SSL terminator | 80, 443 |

| **Flask** 3.0.0 | Web framework | - |

| **Gunicorn** 21.2.0 | WSGI application server | 8000 |

| **Werkzeug** 3.0.0 | ProxyFix middleware | - |

| **OpenSSL** | SSL certificate generation | - |

| **WSL** | Linux environment na Windows | - |

Security Features

HTTPS Only - HTTP auto-redirect do HTTPS (301)

TLS 1.2 + 1.3 - Modern encryption

Self-signed Certificate - RSA 2048-bit

Regular User - Flask uid=1000 (NOT root)

Reverse Proxy - Flask nie dostępny bezpośrednio

Security Headers - HSTS, X-Frame-Options, itp.

Real IP Detection - X-Forwarded-For support

Checklist Weryfikacji

- [x] Certyfikat wygenerowany (`certs/server.crt` , `server.key`)
- [x] Nginx running na portach 80/443
- [x] HTTP redirect do HTTPS (301 status)
- [x] HTTPS dostępny (bez 502 po uruchomieniu Flaska)
- [x] Gunicorn binding na 127.0.0.1:8000
- [x] Flask accessible przez proxy
- [x] `/process-info` pokazuje uid=1000 (zwykły user)
- [x] `/client-ip` pokazuje X-Forwarded-For headers
- [x] Pełny flow: Klient → Nginx → Gunicorn → Flask

Kluczowe Punkty do Zaprezentowania

1. **Certyfikat samopodpisany** - potrzebny do HTTPS, ważny 365 dni

2. **Nginx + reverse proxy** - oddala Flask od sieci, obsługuje SSL

3. **Gunicorn nie Flask dev** - production-ready, skaluje się

4. **uid=1000 (regular user)** - bezpieczeństwo, nie root

5. **X-Forwarded-For headers** - rzeczywisty IP mimo proxy

Gotowy do rozmowy z prowadzącym!

Gotowy do rozmowy z prowadzącym!

Dokument wygenerowany: 27.11.2025 | Wersja: 1.0