Podstawy komunikacji HTTPS

Bezpieczne systemy internetowe

---

1. Ze względu na ograniczenia przeglądarki Chrome (od wersji 58) nie będziemy mogli wykorzystać do ćwiczeń prostych autocertyfikatów (self-signed certificates) **openssl**. Musimy stworzyć minimalną hierarchię z własnym CA. Dodatkowo Chrome wymaga, aby nazwa domeny, dla której wystawiamy certyfikat, była wpisana nie tylko w **Common Name**, ale również w rozszerzeniu **Subject Alternative Name**.
2. Założyć katalog **sec** na klucze i certyfikaty. Podczas generowania kluczy i certyfikatów przeanalizować wszystkie używane ustawienia.
3. W katalogu **sec** wygenerować klucz i certyfikat naszego urzędu certyfikacji
    a. utworzyć plik **ca.conf** zawierający:

```
[req]
distinguished_name = req_distinguished_name
prompt = no
[req_distinguished_name]
C = PL
ST = Zachodniopomorskie
L = Szczecin
O = ZUT
OU = WI
CN = BSI
```

    b. utworzyć klucz urzędu

```
openssl genrsa -out ca.key 2048
```

    c. utworzyć certyfikat urzędu (w jednym kroku powstaje żądanie certyfikatu i certyfikat)

```
openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.pem
-config ca.conf
```

    d. zaimportować certyfikat w Chrome (ustawienia / Zarządzaj certyfikatami / Urzędy / Importuj), zaznaczając, że ufamy mu przy identyfikacji witryn internetowych.
4. Wygenerować klucz i certyfikat dla serwera w domenie **<login>.org**
    a. utworzyć plik **<login>.org.conf** (np. **wmackow.org.conf**) zawierający:

```
[req]
distinguished_name = req_distinguished_name
prompt = no
[req_distinguished_name]
C = PL
ST = Zachodniopomorskie
L = Szczecin
O = ZUT
OU = WI
CN = <login>.org
```

    b. utworzyć plik **<login>.org.ext** zawierający:

```
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
```

```
[alt_names]
DNS.1 = <login>.org
```

c. utworzyć klucz serwera

```
openssl genrsa -out <login>.org.key 2048
```

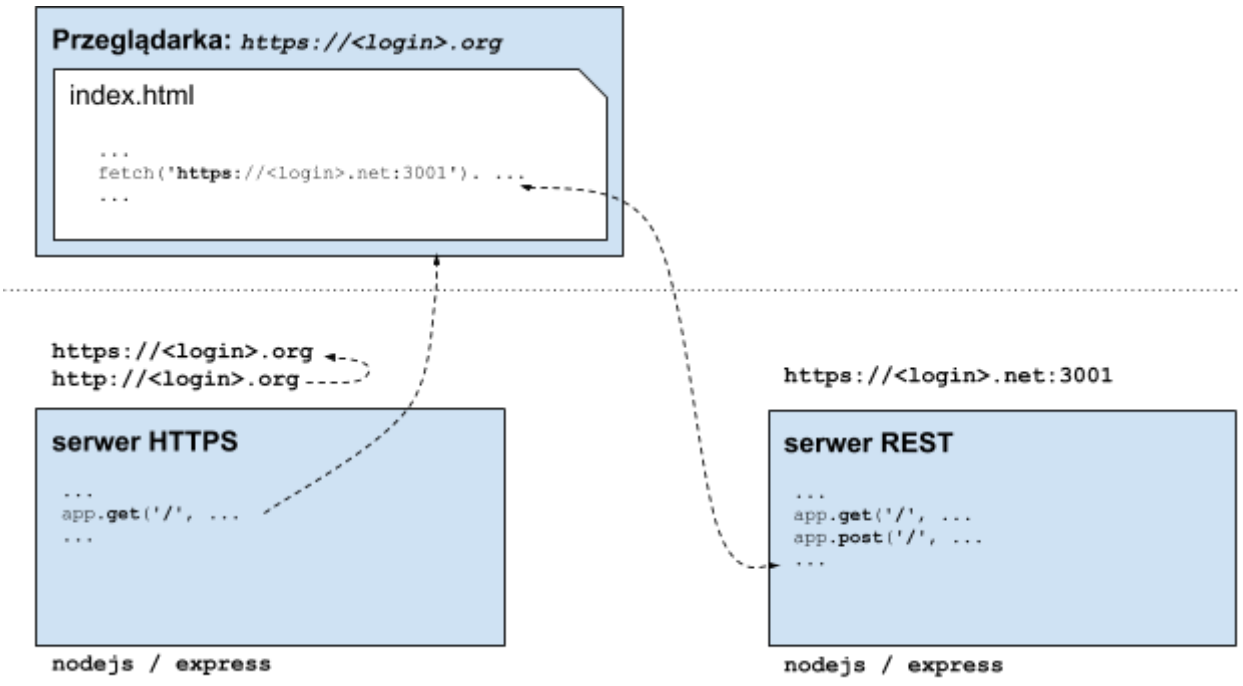
d. utworzyć żądanie certyfikatu serwera

```
openssl req -new -config <login>.org.conf -key <login>.org.key -out
<login>.org.csr
```

e. utworzyć certyfikat serwera (wydany przez nasz urząd CA)

```
openssl x509 -req -in <login>.org.csr -CA ca.pem -CAkey ca.key
-CAcreateserial -out <login>.org.crt -days 365 -sha256 -extfile
<login>.org.ext
```

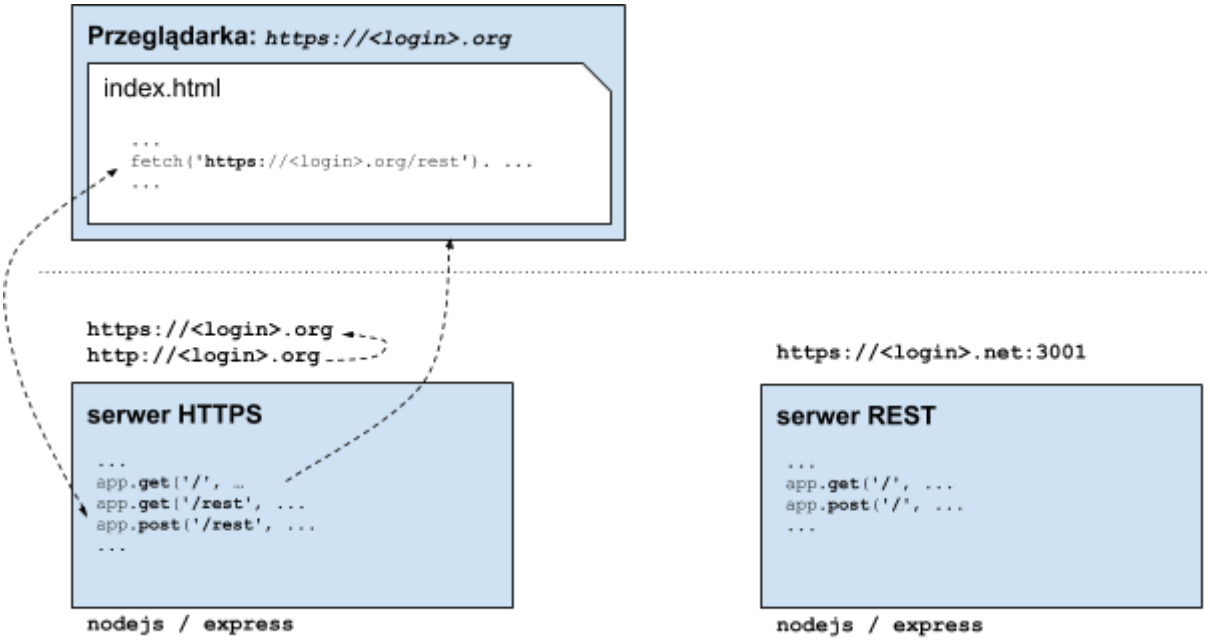
5. Powtórzyć czynności z punktu 3 dla domeny **<login>.net**.
6. Zmodyfikować serwer z poprzednich zajęć, który służył do udostępniania plików z katalogu **public**. Serwer ma być uruchamiany domyślnie na porcie 443 i obsługiwać protokół HTTPS (skorzystać z klucza i certyfikatu wygenerowanych w kroku 3)[1]. Dodatkowo serwer ma przyjmować połączenia HTTP na porcie 80 i forwardować je na HTTPS[2]. Uruchomić serwer i przetestować jego działanie w przeglądarce Chrome, w debugerze otworzyć zakładkę **Security** i prześledzić udostępnione tam informacje. Czy uda się z tak uruchomionej strony (HTTPS) pobrać dane z serwera REST po HTTP?
7. Zmodyfikować serwer REST z poprzednich zajęć tak, aby udostępniał dane po protokole HTTPS (użyć klucz i certyfikat wygenerowane w kroku 4). Odpowiednio zmodyfikować konfigurację CORS.
8. Zmodyfikować plik HTML z poprzednich zajęć uwzględniając nowy adres serwisu REST. Przetestować działanie po wprowadzeniu wszystkich wcześniejszych zmian.



Rys.1 Schemat komunikacji pomiędzy przeglądarką a serwerami (wykonanie ćwiczenia do kroku 8).

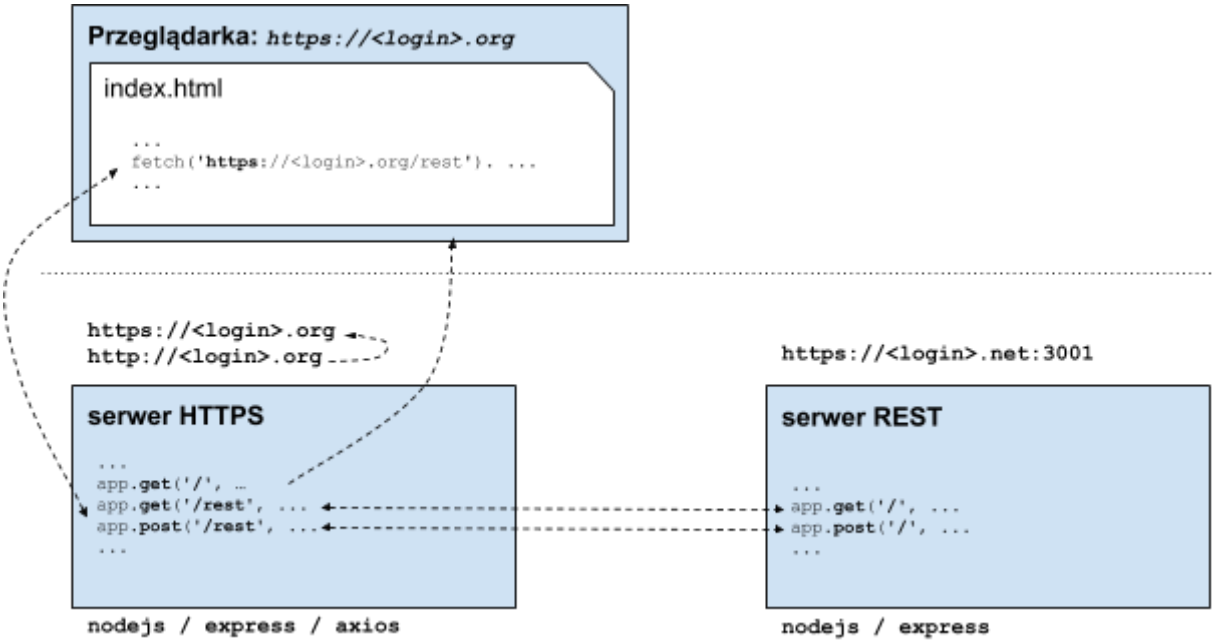
[1] https://timonweb.com/posts/running-expressjs-server-over-https/
[2] https://contextneutral.com/story/creating-an-https-server-with-nodejs-and-express

9. Do serwera HTTPS dodajemy obsługę ścieżki **/rest** (dla metod POST i GET). W pliku HTML zapytania **rest** generowane po naciśnięciu przycisków zmieniamy z adresu **https://<login>.net** na **https://<login>.org/rest**. Wartości zwracane przez obydwie metody nie mają na tym etapie znaczenia (można zwracać przypadkowe odpowiedzi).



Rys.2 Schemat komunikacji pomiędzy przeglądarką a serwerami (wykonanie ćwiczenia do kroku 9).

10. Zainstalować **axios** (npm install axios --save), którego będziemy używali w serwerze HTTPS jako klienta do komunikacji z serwerem REST. Zmodyfikować serwer HTTPS tak, aby zapytanie kierowane na **/rest** (zarówno GET jak i POST) były przekierowywane na serwer REST (przez wywołania **axios**, a nie bezpośrednio przez **redirect** w express)[3] [4].



Rys.3 Schemat komunikacji pomiędzy przeglądarką a serwerami (wykonanie ćwiczenia do kroku 10).

[3] http://nikolay.rocks/2016-02-21-microservices-with-axios
[4] http://codeheaven.io/how-to-use-axios-as-your-http-client/

Ponieważ serwer REST działa obecnie na **https**, to musimy skonfigurować **axios** tak, aby był w stanie podjąć szyfrowaną komunikację. W tym celu musi on posłużyć się certyfikatem serwera REST. Tworzymy tzw. agenta https na bazie certyfikatu serwera REST i przekazujemy go jako ostatni argument wywołania get lub post w **axios**:

```
...
const httpsAgent = new https.Agent({
   rejectUnauthorized: false,
   cert: fs.readFileSync('<login>.net.crt')
 })
...
app.get('/rest', (req, res) => {
   axios.get('https://<login>.net:3001', { httpsAgent }). …
...
```

11. Przetestować działanie aplikacji w przeglądarce analizując jednocześnie przebieg komunikacji za pomocą **Wireshark**.