TLS w Nginx

Poziom: średni **Dotyczy: Nginx 1.18+ z OpenSSL 1.1.1+** **Czas: ~20 min**

Certyfikat i klucz to tylko dwa z kilku parametrów, które decydują o tym, jak bezpieczne i jak zgodne z klientami jest połączenie TLS — tę stronę czytamy razem ze stroną Certyfikaty SSL: wystawienie, konwersja, weryfikacja, która opisuje samo pozyskanie certyfikatu.

Skróty użyte na tej stronie (10)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
HSTS	HTTP Strict Transport Security: nagłówek każący przeglądarce łączyć się tylko po HTTPS
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
OCSP	Online Certificate Status Protocol: sprawdzanie na żywo, czy certyfikat nie został unieważniony
SNI	Server Name Indication: pole w pierwszym komunikacie TLS, w którym klient podaje nazwę serwera, z którym chce rozmawiać, co pozwala HAProxy wybrać backend bez rozszyfrowywania ruchu
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

Fiszka

TYLKO ODCZYT

Wskazanie certyfikatu i łańcucha pośredniego: ssl_certificate

```
ssl_certificate /etc/nginx/ssl/NazwaDomeny/fullchain.pem;
```

> **Pojęcie: fullchain.** Plik `ssl_certificate` musi zawierać nie tylko certyfikat serwera, ale też certyfikat lub certyfikaty pośrednie urzędu certyfikacji (chain) sklejone jeden pod drugim w tym samym pliku, w kolejności od certyfikatu serwera do certyfikatu pośredniego najbliższego korzeniowi. Podanie samego certyfikatu serwera bez łańcucha pośredniego jest jednym z najczęstszych powodów, dla których przeglądarka desktopowa akceptuje stronę bez ostrzeżenia (bo ma łańcuch pośredni w swoim własnym magazynie), a klient mobilny albo skrypt sprawdzający certyfikat przez `openssl verify` zgłasza błąd niekompletnego łańcucha.

TYLKO ODCZYT

Wskazanie klucza prywatnego: ssl_certificate_key

```
ssl_certificate_key /etc/nginx/ssl/NazwaDomeny/privkey.pem;
```

Wskazujemy plik z kluczem prywatnym odpowiadającym certyfikatowi z `ssl_certificate` — jeśli klucz i certyfikat nie pasują do siebie, `nginx -t` przechodzi (składnia jest poprawna), ale usługa nie startuje albo od razu po starcie zwraca błąd w logu. Sposób sprawdzenia zgodności klucza z certyfikatem opisujemy na stronie Certyfikaty SSL: wystawienie, konwersja, weryfikacja.

SSL = Secure Sockets Layer: dawna nazwa szyfrowania, dziś formalnie TLS

TYLKO ODCZYT

Dozwolone wersje protokołu: ssl_protocols

```
ssl_protocols TLSv1.2 TLSv1.3;
```

Ograniczamy protokół do wersji uznawanych obecnie za bezpieczne — TLS 1.0 i TLS 1.1 są uznane za przestarzałe i większość skanerów bezpieczeństwa (oraz część klientów korporacyjnych) traktuje ich obecność jako błąd konfiguracji, nawet jeśli technicznie serwer nadal je obsługuje.

TLS = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

TYLKO ODCZYT

Dozwolone zestawy szyfrów: ssl_ciphers

```
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384;
ssl_prefer_server_ciphers on;
```

> **Pojęcie: zestaw szyfrów (cipher suite).** Kombinacja algorytmów użytych do wymiany klucza, szyfrowania i integralności danych w danej sesji TLS. `ssl_prefer_server_ciphers on` każe serwerowi narzucać własną kolejność preferencji zamiast akceptować pierwszy zestaw zaproponowany przez klienta — istotne, bo starsze klienty potrafią proponować jako pierwsze słabsze zestawy.

□□ **TLS** = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS

□□ TYLKO ODCZYT

Wymuszenie HTTPS przez HSTS: add_header Strict-Transport-Security

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
```

> **Pojęcie: HSTS (HTTP Strict Transport Security).** Nagłówek, który każe przeglądarce zapamiętać, że dana domena ma być odwiedzana wyłącznie przez HTTPS przez czas podany w sekundach (`max-age`) — po pierwszej wizycie przeglądarka sama, bez pytania serwera, zamienia każdą próbę wejścia po `http://` na `https://`, zanim jeszcze wyśle jakiekolwiek żądanie sieciowe. Flaga `always` każe dodawać ten nagłówek do każdej odpowiedzi, także tych z kodami błędów.

Uważaj: ustawienie długiego `max-age` na domenie, która później miałaby wrócić do HTTP (np. przy migracji), oznacza, że przeglądarki użytkowników, którzy już odwiedzili stronę, będą odmawiać połączenia po HTTP przez cały zadeklarowany okres — HSTS jest łatwo włączyć i trudno cofnąć.

□□ **HTTPS** = HTTP zabezpieczony szyfrowaniem TLS · **HSTS** = HTTP Strict Transport Security: nagłówek każący przeglądarce łączyć się tylko po HTTPS · **HTTP** = Hypertext Transfer Protocol: protokół stron WWW

□□ TYLKO ODCZYT

Przekierowanie ruchu z portu 80 na 443

```
server {
    listen 80;
    server_name NazwaDomeny;
    return 301 https://$host$request_uri;
}
```

Osobny blok `server` na porcie 80, którego jedynym zadaniem jest przekierowanie każdego żądania na wersję HTTPS tej samej ścieżki, zachowując oryginalny host i pełną ścieżkę wraz z parametrami zapytania (`$request_uri`).

HTTPS = HTTP zabezpieczony szyfrowaniem TLS

TYLKO ODCZYT

OCSP stapling: ssl_stapling

```
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /etc/nginx/ssl/NazwaDomeny/chain.pem;
resolver 1.1.1.1 8.8.8.8 valid=300s;
```

> **Pojęcie: OCSP stapling.** Zamiast każąc przeglądarce klienta samodzielnie odpytywać urząd certyfikacji o status odwołania certyfikatu (co spowalnia pierwsze połączenie i ujawnia urzędowi certyfikacji, jakie strony odwiedza dany klient), serwer sam okresowo pobiera taką odpowiedź od urzędu certyfikacji i „dolepia" ją (staple) do własnego handshake TLS. Wymaga wskazania osobno pliku z certyfikatem pośrednim (`ssl_trusted_certificate`, bez klucza prywatnego) oraz działającego resolvera DNS, bo Nginx musi sam wykonać zapytanie do serwera OCSP urzędu certyfikacji.

OCSP = Online Certificate Status Protocol: sprawdzanie na żywo, czy certyfikat nie został unieważniony · **TLS** = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS · **DNS** = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP

Z życia wzięte

Z ŻYCIA WZIĘTE

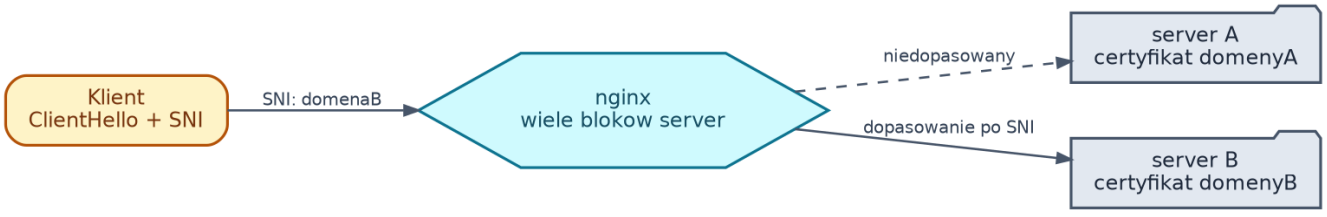
> Pan Wojtek podczas corocznego audytu sprawdza w piątek certyfikat sklepu B2B i zgłasza, że jego skaner widzi TLS 1.0 wciąż włączony na `docker01`, mimo że zespół był pewny, że go wyłączył pół roku temu. Okazuje się, że ktoś podmienił certyfikat przy odnowieniu, ale zapomniał o `nginx -s reload`, więc proces roboczy wciąż działał ze starą konfiguracją sprzed zmiany `ssl_protocols`. Po przeładowaniu `openssl s_client -tls1` faktycznie odmawia połączenia, tak jak powinno być od dawna.

Sprawdzenie, co serwer faktycznie podaje

Napisanie poprawnej konfiguracji nie gwarantuje, że serwer faktycznie ją stosuje — literówka w ścieżce do certyfikatu, stary proces, który nie przeładował się po zmianie, albo konfiguracja w złym bloku `server` potrafią sprawić, że klient dostaje zupełnie inny certyfikat niż się spodziewamy.

```
openssl s_client -connect NazwaDomeny:443 -servername NazwaDomeny </dev/null 2>/dev/null |
openssl x509 -noout -subject -issuer -dates
```

> **Pojęcie: SNI (Server Name Indication).** Parametr `-servername` w powyższej komendzie jest konieczny, gdy jeden adres IP i port obsługuje wiele domen z osobnymi certyfikatami (typowa sytuacja dla reverse proxy) — bez niego serwer nie wie, dla której domeny zwrócić certyfikat, i zwraca certyfikat domyślnego bloku `server` (albo pierwszego zdefiniowanego, jeśli żaden nie jest oznaczony `default_server`), co bywa mylące przy diagnozowaniu problemu.



Klient wysyła nazwę domeny już w ClientHello, a nginx na podstawie SNI wybiera blok server z pasującym certyfikatem zamiast domyślnego (kliknięcie otwiera pełny rozmiar)

Przykład:

```
subject=CN = app01.zielona.example
issuer=C = US, O = Let's Encrypt, CN = R11
notBefore=Aug 12 10:03:11 2026 GMT
notAfter=Nov 10 10:03:10 2026 GMT
```

Sprawdzenie, jakie wersje protokołu i zestawy szyfrów serwer faktycznie oferuje, negocjując po kolei każdą wersję:

```
openssl s_client -connect NazwaDomeny:443 -servername NazwaDomeny -tls1_2 </dev/null 2>&1 |
grep -E 'Protocol|Cipher'
```

Przykład:

```
Protocol  : TLSv1.2
Cipher    : ECDHE-RSA-AES256-GCM-SHA384
```

Odpowiedź `Protocol: TLSv1.2` przy próbie wymuszenia `-tls1` (TLS 1.0) oznacza, że serwer odmówił tej wersji i wynegocjował coś innego albo zerwał połączenie — dokładnie tego oczekujemy po ograniczeniu `ssl_protocols` do wersji 1.2 i 1.3.

Częste błędy

Certyfikat bez łańcucha pośredniego. Strona działa poprawnie w przeglądarce testującej z komputera administratora (bo system ma łańcuch pośredni w swoim magazynie zaufania z wcześniejszych wizyt na innych stronach tego samego urzędu), ale klient mobilny albo integracja API zgłasza błąd `unable to get local issuer certificate` — naprawą jest sklejenie certyfikatu pośredniego do pliku `fullchain.pem` w `ssl_certificate`.

Nieprzeładowany proces po podmianie certyfikatu. Certyfikat na dysku jest już nowy, ale `openssl s_client` nadal pokazuje stary — Nginx trzyma certyfikat wczytany do pamięci procesu roboczego i nie zauważa podmiany pliku bez `nginx -s reload`, dokładnie tak samo jak przy każdej innej zmianie konfiguracji, opisanej na stronie Architektura Nginx: procesy, konteksty, pliki konfiguracyjne.

Brak resolvera przy włączonym OCSP stapling. Usługa startuje, ale klienci dostają ostrzeżenie o nieważnym stapled response albo Nginx w logu zgłasza błąd pobrania odpowiedzi OCSP — `resolver` jest wymagany, bo bez niego proces nie ma jak wykonać zapytania DNS do serwera OCSP urzędu certyfikacji.

Ćwiczenia

> Ćwiczenia zakładają dowolny certyfikat testowy, choćby samopodpisany, zainstalowany w lokalnym Nginksie.

Ćwiczenie 1 — porównanie wyniku dla różnych wersji TLS

```
for v in tls1 tls1_1 tls1_2 tls1_3; do
  echo "== $v =="
  openssl s_client -connect localhost:443 -servername NazwaDomeny -$v </dev/null 2>&1 | grep -E 'Protocol|error'
done
```

Sprawdzamy, które wersje protokołu serwer faktycznie akceptuje po ograniczeniu `ssl_protocols` do `TLSv1.2 TLSv1.3`.

Odpowiedź

Dla `tls1` i `tls1_1` połączenie powinno zakończyć się błędem negocjacji (np. `no protocols available`), a dla `tls1_2` i `tls1_3` komenda zwraca poprawny `Protocol:` z nazwą wynegocjowanej wersji.

Powiązane: Certyfikaty SSL: wystawienie, konwersja, weryfikacja · Reverse proxy i równoważenie obciążenia · Architektura Nginx: procesy, konteksty, pliki konfiguracyjne

Źródła: dokumentacja nginx.org, rozdział „ngx_http_ssl_module" · `man 1 openssl-s_client`

Revision #37
Created 2026-09-25 07:40:14 UTC by Tomasz Konieczny
Updated 2026-10-26 14:16:14 UTC by Tomasz Konieczny