Sieci VPN. Zdalna praca i bezpieczeństwo danych

Autor: Marek Serafin
ISBN: 978-83-246-1521-6
Format: B5, stron: 160

Poznaj i wykorzystaj w praktyce metody korzystania z sieci VPN

- Na czym opiera się standard SSL?
- Jak zestawiać tunele VPN w systemach Windows i Linux?
- Jak połączyć oddziały firm za pomocą tunelu IPSec?

Serwery plików i baz danych spotykamy niemal w każdej firmie. Architektura klient-serwer umożliwia dostęp do aplikacji nie tylko wewnątrz firmy, ale także z dowolnego innego miejsca. Rozwój sieci pozwolił wielu organizacjom na sprawną komunikację i otworzył perspektywy dla tych pracowników, którzy z różnych względów wykonują swoje obowiązki poza biurem. Niestety – zdalny dostęp do firmowej infrastruktury IT niesie ze sobą także zagrożenia związane z możliwością utraty, uszkodzenia lub wydostania się na zewnątrz cennych danych. Rozwiązaniem tego problemu są łącza szyfrowane, nazywane VPN.

Książka „Sieci VPN. Zdalna praca i bezpieczeństwo danych" to praktyczny przewodnik dla administratorów sieci firmowych, którzy zajmują się wdrażaniem rozwiązań umożliwiających pracę na odległość. Opisuje wszystkie aspekty konfigurowania tuneli VPN z wykorzystaniem protokołów SSL (OpenVPN) i IPSec (OpenSWAN) w systemach Linux i Windows. Czytając ją, poznasz standard SSL, zasady generowania certyfikatów oraz metody implementacji sieci VPN. Analizując zawarte w książce przykłady, nauczysz się otwierać zdalny dostęp do sieci korporacyjnej, łączyć oddziały firmy za pomocą IPSec i uruchamiać tunele VPN w urządzeniach mobilnych.

- Zagrożenia wynikające z konstrukcji protokołu TCP/IP
- Przesyłanie danych z wykorzystaniem SSL
- Zapewnianie pracownikom zdalnego dostępu do zasobów firmy
- Generowanie kluczy
- Tworzenie tuneli SSH
- Instalacja i konfiguracja programu OpenVPN
- Tunele VPN w urządzeniach mobilnych
- Implementacja IPSEC/L2TP w systemie Linux
- Konfiguracja VPN w systemie Windows

Zabezpiecz dostęp do swojej sieci.
Skorzystaj z wiedzy doświadczonego administratora

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 3.

SSL jako standard bezpiecznego przesyłania danych

3.1. Historia i znaczenie protokołu SSL

W odpowiedzi na problemy związane z brakiem zabezpieczeń w popularnych protokołach internetowych firma Netscape Communications Corporation w latach 90. ubiegłego wieku opracowała protokół SSL. Obecnie najpopularniejsza jest wersja trzecia protokołu — SSL 3, ponieważ wcześniejsze implementacje zawierały kilka błędów. W 1996 roku za sprawą IETF (ang. *Internet Engineering Task Force*) powstała grupa robocza TLS (ang. *Transport Layer Security*), która ma za zadanie rozwijać oraz publikować standard SSL.

W założeniach SSL powstał jako zabezpieczenie do protokołu http dla potrzeb usług e-commerce. Jednak dzięki jego uniwersalności można wykorzystać go do zabezpieczenia większości usług TCP, a nawet do tworzenia sieci VPN, co zostanie przedstawione w niniejszej książce.

Protokół SSL zapewnia następujące podstawowe funkcje bezpieczeństwa:

- uwierzytelnianie stron — czyli potwierdzenie ich autentyczności na podstawie certyfikatów,
- poufność i integralność przesyłu — tzn. ochronę przed podsłuchaniem i modyfikacją.

Proces uwierzytelniania przebiega przy użyciu asymetrycznych algorytmów kryptograficznych. W związku z powyższym każda ze stron musi posiadać swój własny klucz prywatny (tajny) oraz klucz publiczny (certyfikat).

Po pozytywnym uwierzytelnieniu stron następuje wymiana danych przy użyciu któregoś z ustalonych, symetrycznych algorytmów kryptograficznych (są znacznie szybsze).

Aby możliwe było potwierdzenie autentyczności strony biorącej udział w komunikacji, np. serwera, musi ona wylegitymować się certyfikatem podpisanym przez zaufane centrum certyfikacji (ang. CA — *Certificate Authority*).

CA to zaufana instytucja, która zajmuje się wystawianiem certyfikatów. Do jej obowiązków należy sprawdzenie prawnych podstaw wnioskowania podmiotu o certyfikat. Chodzi tu głównie o prawo do nazwy firmy, nazwy domeny itd.

Na certyfikat składają się pola zawierające nazwę właściciela, nazwę podmiotu, okres ważności, a także certyfikat głównego i ewentualnie pośrednich centrów certyfikacji — całość stanowi tzw. ścieżkę certyfikacji.

3.1.1. Przebieg nawiązania połączenia SSL

Zanim protokoły warstwy aplikacji będą mogły wymieniać dane w bezpieczny sposób, musi nastąpić nawiązanie sesji SSL (ang. *SSL handshake*). Na SSL handshake składa się kilka faz negocjacji, które przedstawiono kolejno w punktach.

1. Klient łączy się z serwerem i wysyła pakiet początkowy Hello, a wraz z nim numer obsługiwanej wersji SSL, obsługiwane algorytmy szyfrujące, algorytmy kompresji oraz losowy numer związany z rozpoczętą sesją (ID).
2. Serwer w odpowiedzi wysyła klientowi numer obsługiwanej wersji SSL, obsługiwane algorytmy szyfrujące, a także swój certyfikat (klucz publiczny).
3. Na tym etapie klient sprawdza certyfikat serwera — czy jest ważny oraz czy wystawił go zaufany urząd (CA). Protokół SSL przewiduje także możliwość wysłania przez serwer żądania o uwierzytelnienie klienta. Uwierzytelnianie to jest opcjonalne i stosuje się je w określonych warunkach.
4. W przypadku pozytywnego uwierzytelnienia serwera klient generuje 48-bajtową liczbę zwaną „pre-master secret" i szyfruje ją, używając przy tym klucza publicznego serwera (zawartego w certyfikacie serwera). Liczba „pre-master" składa się z 2 bajtów identyfikujących klienta oraz 46 bajtów losowych.
5. Serwer po otrzymaniu liczby „pre-master" odszyfrowuje ją, używając do tego swojego klucza prywatnego, i porównuje pierwsze 2 bajty identyfikujące klienta z danymi, które otrzymał w inicjacyjnym pakiecie Hello.
6. Jeśli jest wymagane uwierzytelnienie klienta, jest to robione w tej chwili. Wówczas klient musi przesłać swój certyfikat.
7. Na podstawie już wymienionych danych (m.in. pre-master key, losowe dane wygenerowane w punkcie 1.) serwer i klient generują tzw. master key (znany tylko im).

8. Zarówno klient, jak i serwer na podstawie master-key generują symetryczne klucze sesyjne, które umożliwiają im szyfrowanie i sprawdzanie integralności przesyłanych danych[1].
9. Kończąc handshake, klient przesyła do serwera wiadomość zaszyfrowaną ustalonym kluczem sesyjnym. Wiadomość ta, nazywana końcowym uzgodnieniem (ang. *finished handshake*), jest jako pierwsza szyfrowana tajnym kluczem.
10. Serwer odpowiada także wiadomością zaszyfrowaną za pomocą wspólnego klucza. Od tej pory sesja SSL jest nawiązana.

3.1.2. Znaczenie zaufanego certyfikatu

Co w praktyce oznacza, że dany certyfikat jest zaufany?

Oznacza to tylko (aż) tyle, że został wystawiony przez wiarygodne (zaufane) Centrum Certyfikacji CA. Wszystkie popularne przeglądarki internetowe, programy pocztowe i inne aplikacje korzystające z protokołu SSL mają „zaszytą" w sobie na stałe listę zaufanych wystawców CA (ich certyfikaty). Dzięki temu podczas nawiązywania połączenia SSL aplikacja nie zgłasza błędu, rozpoznając, że certyfikat został wystawiony przez któryś z zaufanych CA.

Certyfikaty wystawione przez zaufane CA mają znaczenie głównie dla publicznych serwerów WWW, gdzie rozproszeni po całym świecie klienci muszą mieć pewność, że serwer, z którym się łączą, jest na pewno tym, za jaki się podaje (np. sklep internetowy).

W przypadku tworzenia połączeń VPN nic nie stoi na przeszkodzie, abyś stworzył swoje CA i sam wystawiał certyfikaty na własne potrzeby. Możesz przecież ufać certyfikatom, które sam wygenerowałeś, i instalować je na laptopach pracowników.

W przeciwieństwie do serwera WWW i przeglądarek internetowych w zastosowaniach VPN-owych często ważne jest uwierzytelnienie klienta przez serwer (bramę VPN). Nie chcemy przecież, aby do sieci korporacyjnej mógł podłączyć się ktokolwiek na świecie posiadający klienta VPN, a jedynie osoby posiadające odpowiednie certyfikaty.

Kolejny punkt tego rozdziału zawiera opis programu OpenSSL wraz z przykładami tworzenia kluczy, wniosków i certyfikatów.

[1] Na podstawie master-key generowanych jest sześć kluczy — trzy w kierunku serwer-klient i trzy w drugą stronę. Klucze te służą do szyfrowania i sprawdzania integralności danych. Zainteresowanych Czytelników odsyłam do dokumentacji dostępnej na stronie korporacji Netscape: *http://wp.netscape.com/eng/ssl3/draft302.txt*.

3.2. Generowanie certyfikatów przy użyciu programu OpenSSL

Większość programów opisanych w tej książce do uwierzytelniania stron wykorzystuje protokół SSL. Informacje zawarte w tym rozdziale, a w szczególności przykłady generowania kluczy i certyfikatów X.509, wykorzystywane będą często w dalszej części książki. Aby uniknąć wielokrotnego opisywania tych samych czynności w następnych rozdziałach, będę odsyłał Czytelnika do instrukcji zawartych w niniejszym punkcie.

Do stworzenia własnego CA, generowania kluczy, wniosków i certyfikatów będziemy używali najpopularniejszej w świecie „open source" biblioteki openssl. Biblioteka ta zawiera implementację protokołów SSL i TLS oraz większość używanych algorytmów kryptograficznych. Korzysta z niej szereg znanych aplikacji wykorzystujących algorytmy szyfrujące, jak np. OpenSSH, OpenVPN, ApacheSSL itp.

Biblioteka openssl dostarczana jest standardowo wraz z popularnymi dystrybucjami Linuksa. Najprawdopodobniej bibliotekę tę masz zainstalowaną w systemie. Aby się o tym przekonać, wpisz po prostu polecenie `openssl`. Powinien zgłosić się znak zachęty programu: `OpenSSL>`. W przeciwnym razie musisz ją zainstalować w sposób typowy dla swojej dystrybucji. Ostatecznie możesz pobrać źródła najnowszej wersji ze strony *http://openssl.org* i własnoręcznie skompilować program.

Wersja instalacyjna pod systemy Windows dostępna jest na stronie *http://www.slproweb.com/products.html*.

Zanim przejdziemy do generowania certyfikatów dla serwerów i klientów, musimy stworzyć własny Urząd Certyfikacji (CA). Na początku dwie uwagi ogólne.

Ważne jest, abyś swoje CA utworzył na jakimś bezpiecznym komputerze odłączonym od internetu albo przynajmniej za firewallem, tak aby nie był on bezpośrednio „widoczny" w internecie.

Ważne jest także regularne robienie kopii bezpieczeństwa wystawionych certyfikatów oraz całego katalogu „ssl", tak aby w razie potrzeby można było unieważnić któryś z certyfikatów.

Zakładam, że masz zainstalowany pakiet OpenSSL. Jeżeli nie, to musisz go zainstalować, używając menedżera pakietów swojej dystrybucji, lub skompilować program samodzielnie.

3.2.1. Tworzenie własnego CA

W pierwszej kolejności odnajdujemy plik *openssl.cnf*. Prawdopodobne lokalizacje tego pliku to:

- */etc/ssl/openssl.cnf* — dla programu instalowanego z paczek (np. Debian),
- */usr/local/etc/openssl.cnf* — w przypadku wersji kompilowanej ręcznie,

♦ *C:\OpenSSL\bin* — dla systemów Win32.

W pliku tym odnajdujemy sekcję `[ CA_default ]`. Zmień wpisy u siebie tak, aby wyglądały jak te z listingu 3.2.1.1.

Listing 3.2.1.1. *Wycinek pliku openssl.cnf*

```
[ CA_default ]
dir = /etc/ssl                          # główny katalog, w którym zapisywane są pliki
certs = /etc/ssl/certs                  # katalog, w którym zapisywane są certyfikaty
crl_dir = $dir/crl                      # katalog z listą certyfikatów unieważnionych (CRL)
private_key = $dir/private/cakey.pem # klucz prywatny CA
database = $dir/index.txt               # baza, w której przechowywane są informacje
                                          ↪ o wystawionych certyfikatach wraz ze statusem
certificate = $dir/cacert.pem           # Certyfikat CA — do podpisu wniosków
serial = $dir/serial                    # plik pomocniczy z bieżącym numerem — inkrementowany
                                        # po każdym wystawieniu certyfikatu
crl = $dir/crl.pem                      # bieżąca lista certyfikatów unieważnionych
```

Upewniamy się, czy istnieje katalog podany w zmiennej `dir`, czyli */etc/ssl*, oraz wszystkie jego podkatalogi. Jeżeli nie, musimy je założyć. Dla katalogu *ssl/private* ustaw uprawnienia tak, aby tylko użytkownik *root* mógł do niego wejść.

Stwórz pliki */etc/ssl/index.txt* oraz */etc/ssl/serial*, używając komend podanych poniżej.

```
root@ca:~# touch /etc/ssl/index.txt          #(ma być pusty)
root@ca:~# echo 00 > /etc/ssl/serial         #(ma zawierać wpis 00)
```

Przystępujemy do generowania klucza prywatnego centrum certyfikacji CA. Jest to czynność jednorazowa, tzn. po wygenerowaniu klucza prywatnego CA, a następnie odpowiadającego mu certyfikatu będziesz używał ich do podpisywania innych certyfikatów. Pamiętaj, aby zarchiwizować pliki z katalogu */etc/ssl* w bezpiecznym miejscu.

Będąc w katalogu */etc/ssl*, wydajemy następujące polecenie:

```
root@ca:/etc/ssl# openssl genrsa -des3 -out private/cakey.pem 1024
Generating RSA private key, 1024 bit long modulus
.........++++++
...++++++
e is 65537 (0x10001)
Enter pass phrase for private/cakey.pem: <podaj hasło klucza prywatnego CA>
```

Po potwierdzeniu hasła do klucza prywatnego CA klucz zostanie zapisany w pliku *private/cakey.pem*. Nie zapomnij hasła do tego klucza, będzie Ci nieraz potrzebne.

Kolejną czynnością jest wygenerowanie certyfikatu CA. W tym celu wpisujemy następujące polecenie:

```
root@ca:/etc/ssl# openssl req -new -x509 -days 365 -key private/cakey.pem -out
↪cacert.pem
```

Zostaniemy poproszeni o podanie kilku pól zawartych w certyfikacie.

```
Country Name (2 letter code) [AU]:PL
State or Province Name (full name) [Some-State]:Slask
```

```
Locality Name (eg, city) []:Gliwice
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Moja Firma Sp. z o.o.
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []: ca.firma.pl
Email Address []:
```

Znaczenie powyższych pól jest raczej jasne, zwracam jedynie uwagę na pole `Common Name`, które powinno zawierać nazwę podmiotu — np. nazwę użytkownika lub jednostki. W tym przypadku, gdy generujesz certyfikat dla CA, wpisz tutaj nazwę swojej organizacji słownie (np. Firma SA) albo podaj np. nazwę domeny.

Po podaniu hasła do klucza prywatnego certyfikat zostanie zapisany w pliku *cacert.pem*.

W powyższym przykładzie czas ważności certyfikatu wynosić będzie 1 rok. Można go oczywiście wydłużyć.

Na tym zakończyliśmy tworzenie własnego urzędu CA. Mając pliki *cakey.pem* i *cacert.pem*, czyli klucz prywatny i publiczny CA, możemy już wystawiać certyfikaty innym podmiotom.

3.2.2. Tworzenie klucza prywatnego dla serwera

Celem stworzenia klucza prywatnego dla serwera wpisujemy następujące polecenie:

```
root@ca:/etc/ssl# openssl genrsa -des3 -out private/serverkey.pem 1024
```

Program OpenSSL zapyta o hasło — będzie to hasło klucza prywatnego serwera. Klucz prywatny zapisany zostanie w pliku *private/serverkey.pem*.

Następną czynnością jest wygenerowanie wniosku o wystawienie certyfikatu.

3.2.3. Generowanie wniosku o wystawienie certyfikatu

```
root@ca:/etc/ssl# openssl req -new -key private/serverkey.pem -out serverreq.pem
```

Będziesz musiał podać hasło klucza prywatnego serwera (to, które podawałeś przed chwilą). Jeżeli hasło będzie poprawne, zostaniesz zapytany o dane do wniosku, zgodnie z listingiem poniżej.

```
Country Name (2 letter code) [AU]:PL
State or Province Name (full name) [Some-State]:Slask
Locality Name (eg, city) []:Gliwice
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Moja Firma Sp. z o.o.
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []: server.firma.pl
Email Address []:
```

Zwracam tutaj uwagę na pole `Common Name`. W przypadku certyfikatów dla serwerów pole to powinno zawierać pełną nazwę domenową, pod jaką serwer działa w internecie

(tzw. FQDN). Jest to ważne, ponieważ jeśli pole `Common Name` nie pokrywa się z nawą domenową, niektóre programy zgłaszają niezgodność certyfikatu z adresem serwera (zwłaszcza przeglądarki i klienty poczty).

Wniosek zostanie zapisany w pliku *serverreq.pem*. Kolejnym krokiem będzie podpisanie go przez nasze CA, czyli wystawienie mu certyfikatu.

3.2.4. Wystawianie certyfikatu dla serwera

Celem wystawienia certyfikatu dla podmiotu (serwera) musisz podpisać jego wniosek. Aby to uczynić, wpisz poniższe polecenie.

```
root@ca:/etc/ssl# openssl ca -notext -in serverreq.pem -out servercert.pem
```

Zostaniesz zapytany o hasło do klucza prywatnego CA *cakey.pem* (nie myl z hasłem do klucza prywatnego serwera!).

Następnie OpenSSL pokaże szczegóły certyfikatu i zapyta, czy chcesz go podpisać. Operacja podpisu wygląda tak, jak pokazano na listingu 3.2.4.1.

Listing 3.2.4.1. *Komunikaty pojawiające się podczas wystawiania certyfikatu*

```
Signature ok
Certificate Details:
Serial Number: 5 (0x5)
Validity
Not Before: Sep 17 12:59:06 2007 GMT
Not After : Sep 16 12:59:06 2008 GMT
Subject:
countryName = PL
stateOrProvinceName = Slask
organizationName = Moja Firma Sp. z o.o.
organizationalUnitName =
commonName = server.firma.pl
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
0E:CE:3E:06:C4:46:53:78:B0:05:AB:18:9B:BA:90:79:9B:A1:A5:C8
X509v3 Authority Key Identifier:
keyid:FC:B8:73:29:C6:E4:50:B2:3E:CE:0A:78:8C:62:90:A5:62:3C:87:1B
DirName:/C=PL/ST=Slask/L=Gliwice/O=Moja Firma Sp. z o.o./
CN=ca.firma.pl/emailAddress=admin@firma.pl
serial:97:1B:4E:CE:0B:5F:CE:E2
Certificate is to be certified until Sep 16 12:59:06 2008 GMT (365 days)
Sign the certificate? [y/n]: y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Możesz teraz podejrzeć zawartość plików *index.txt* oraz *serial*, które zostały zaktualizowane po podpisaniu wniosku. Uważaj na plik *index.txt*, jest on potrzebny przy odwoływaniu certyfikatu (generowaniu CRL).

Mamy już parę kluczy dla serwera. Przy zestawianiu sieci VPN będą potrzebne też klucze i certyfikaty dla użytkowników. Generujemy je w sposób analogiczny jak dla serwera, zgodnie z punktami 3.2.2 – 3.2.4. Zwróć uwagę, aby w polu `Common Name` podawać dane jednoznacznie identyfikujące użytkownika — np. jego login korporacyjny. Niektóre programy po `Common Name` identyfikują użytkowników.

Jeszcze jeden szczegół dotyczący hasła zabezpieczającego klucz prywatny. W momencie uruchamiania aplikacji korzystającej z tego klucza będziemy zmuszeni wpisywać hasło z klawiatury. W praktyce na serwerach aplikacja często działa jako usługa uruchamiana w trakcie startu systemu i oczekiwanie na wpisanie hasła jest niezadowalającym rozwiązaniem. W takich sytuacjach możemy ściągnąć hasło z klucza prywatnego. Należy jednak pamiętać, aby dostęp do klucza miał jedynie administrator serwera (*root*). Komenda przedstawiona w punkcie 3.2.5 umożliwia ściągnięcie hasła z klucza prywatnego.

3.2.5. Ściąganie hasła z klucza prywatnego serwera

```
# openssl rsa -in private/serverkey.pem -out private/serverkey.pem_bezhasla
```

Nie zalecam natomiast ściągania haseł z kluczy prywatnych użytkowników zdalnych. Jest to dodatkowe zabezpieczenie. W razie kradzieży laptopa dostęp do VPN nie będzie możliwy bez znajomości hasła.

3.2.6. Unieważnianie certyfikatów

Prędzej czy później zajdzie potrzeba unieważnienia któregoś z certyfikatów. Klasycznym przykładem jest odejście pracownika lub kradzież laptopa. Do unieważnienia certyfikatu służy parametr `revoke` programu OpenSSL. Przykładowo aby przystąpić do unieważnienia certyfikatu osobie Jan Kowalski (*jkowalskicert.pem*), wpisz polecenie:

```
root@srv:/etc/ssl# openssl ca -revoke jkowalskicert.pem
```

OpenSSL zapyta o hasło klucza CA i po podaniu prawidłowego unieważni certyfikat:

```
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for /etc/ssl/private/cakey.pem:
DEBUG[load_index]: unique_subject = "yes"
Revoking Certificate 04.
Data Base Updated
```

Musimy jeszcze wygenerować listę CRL, w której zapisane są unieważnione certyfikaty. Robimy to zgodnie z punktem 3.2.7.

3.2.7. Generowanie listy CRL (unieważnionych certyfikatów)

```
root@ca:/etc/ssl# openssl ca -gencrl -out crl.pem
```

Plik *crl.pem* należy przegrać później na właściwy serwer i wskazać w aplikacji korzystającej z certyfikatów (szczegóły w dalszej części książki).

3.2.8. Różne formaty certyfikatów

Należy wspomnieć jeszcze o różnych formatach plików, w których zapisywane są klucze i certyfikaty. Niestety, nie ma tu jednego standardu i różni producenci preferują różne formaty. Niemniej za pomocą programu OpenSSL możesz je konwertować z jednego formatu na inny. Klucze prywatne najczęściej zapisywane są w formie PEM lub DER (binarny). Dla certyfikatów używane są formaty PEM, DER oraz PKCS12. Aplikacje bazujące na bibliotece openssl, czyli wszystkie uniksowe, używają na ogół formatu PEM.

Formaty DER i PKCS12 rozpowszechnione są w systemach Microsoftu. Format PKCS12 przechowuje w jednym pliku klucz prywatny zabezpieczony hasłem i odpowiadający mu certyfikat. Poprzednikiem PKCS12 był przestarzały obecnie format PFX; jakkolwiek firma Microsoft używa rozszerzeń plików PFX dla formatu PKCS12.

Aby przekonwertować certyfikat z jednej postaci na drugą, musisz przekazać programowi OpenSSL odpowiednie parametry. W tabeli 3.2.8.1 przedstawiono składnię programu dla kilku popularnych przekształceń.

Tabela 3.2.8.1. *Składnia programu OpenSSL potrzebna do konwersji certyfikatów*

Format wejściowy	Format wyjściowy	Składnia OpenSSL
PEM	DER	`openssl x509 -in cert.pem -out cert.der ↪-outform DER`
DER	PEM	`openssl x509 -in cert.der inform DER ↪-out cert.pem -outform PEM`
DER key	PEM key	`openssl rsa in input.key inform DER out ↪output.key outform PEM`
PEM	PKCS#12	`openssl pkcs12 -export -out cert.p12 -inkey ↪userkey.pem -in usercert.pem`
PKCS#12	PEM CERT	`openssl pkcs12 -clcerts -nokeys -in cert.p12 ↪-out usercert.pem`
PKCS#12	PEM KEY	`openssl pkcs12 -nocerts -in cert.p12 ↪-out userkey.pem`

Aby wyświetlić informację o certyfikacie, np. informacje podane podczas tworzenia wniosku, należy uruchomić program OpenSSL z następującymi parametrami:

```
ca:/etc/ssl# openssl x509 -in servercert.pem -subject -noout
subject= /C=PL/ST=Slask/O=Helion/CN=server1
```

Jeśli dodasz parametr `-issuer`, OpenSSL zwróci także informację o wystawcy (CA):

```
ca:/etc/ssl# openssl x509 -in servercert.pem -issuer -subject -noout
subject= /C=PL/ST=Slask/O=Helion/CN=server1
issuer= /C=PL/ST=Slask/L=Gliwice/O=Helion/CN=CA
```

Jeżeli certyfikat jest w formie binarnej (DER), do powyższej składni należy dodać parametr `-inform DER`.

3.3. Kompilacja biblioteki openssl ze źródeł

Kończąc rozdział dotyczący biblioteki openssl, zamieszczam instrukcję opisującą, jak skompilować i zainstalować bibliotekę „ręcznie". Powodów, dla których miałbyś samodzielnie kompilować program OpenSSL, może być kilka. Najbardziej prawdopodobny z nich to chęć podania przy kompilacji jakiejś opcji, której nie przewidzieli twórcy dystrybucji Twojego systemu. Innym powodem może być chęć zaktualizowania istniejącej wersji w dystrybucjach, które nie mają dobrze zorganizowanego systemu aktualizacji (np. Slackware ;-)).

Jeśli decydujesz się na własną kompilację, musisz sukcesywnie aktualizować bibliotekę w razie pojawienia się wykrytych błędów.

W dystrybucji Debian, której używam, paczka SSL dostarczana wraz z systemem nie była skompilowana z obsługą biblioteki zlib, w związku z czym nie mogłem użyć tego algorytmu kompresji w programie Stunnel.

Jeśli z jakichś powodów chcesz skompilować bibliotekę openssl, poniżej w punktach zamieszczam opis, jak to zrobić.

1. Pobierz ze strony *http://www.openssl.org/source/* źródła najnowszej wersji pakietu i zapisz w katalogu */usr/src/*.
2. Porównaj wartość MD5 pliku pobranego z sieci (polecenie `md5sum`) z wartością udostępnioną na stronie *openssl.org*.
3. Rozpakuj zawartość archiwum poleceniem `tar zxf openssl-<nr_wersji>.tar.gz`.
4. Przejdź do katalogu *openssl-<nr_wersji>*.
5. Przed przystąpieniem do kompilacji musisz ustalić, w którym katalogu program ma zostać zainstalowany oraz z jakimi dodatkowymi opcjami, podając je jako parametry skryptu *./config*. W poniższym przykładzie skompilujemy program z obsługą biblioteki zlib, a wynikowy program zostanie zainstalowany w katalogu */usr/local/openssl*. Wpisz polecenie:

   ```
   srv:~# ./config --prefix=/usr/local  zlib
   ```

6. Jeśli skrypt *./config* nie zgłosi błędu, możemy przejść do właściwej kompilacji programu. W tym celu wpisz polecenie `make`.

7. Proces kompilacji może potrwać kilka minut, po jego zakończeniu możesz przejść do ostatniego kroku — instalacji skompilowanych plików we właściwych katalogach.
8. Aby zakończyć instalację, wpisz polecenie `make install`.