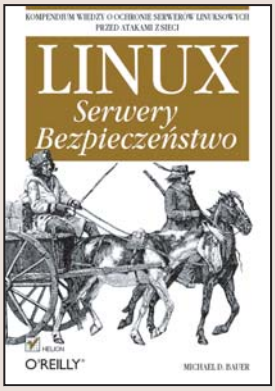
Linux. Serwery. Bezpieczeństwo



Autor: Michael D. Bauer
Tłumaczenie: Marek Pętlicki (rozdz. 6–11), Grzegorz Werner (przedmowa, rozdz. 3, 5, 12, 13, dod. A), Sławomir Woźniak (rozdz. 1, 2, 4)
ISBN: 83-7361-988-7
Tytuł oryginału: Linux Server Security
Format: B5, stron: 520

Kompendium wiedzy o ochronie serwerów linuksowych przed atakami z sieci

- Projektowanie sieci granicznej
- Korzystanie z mechanizmów szyfrowania transmisji
- Zabezpieczanie usług udostępnianych przez serwer

Pod kontrolą systemów operacyjnych z rodziny Linux działają setki serwerów internetowych. Możliwości Linuksa pozwalają na uruchomienie serwera WWW, FTP, poczty elektronicznej, DNS i baz danych. Aby jednak funkcje serwerowe działały bez zakłóceń, udostępniony w sieci serwer należy odpowiednio zabezpieczyć. Bezpieczeństwo serwerów, szczególnie w świetle rosnącej ilości włamań i kradzieży danych, jest niezwykle istotnym zagadnieniem. Linux wyposażony jest w narzędzia umożliwiające zabezpieczenie uruchomionych w nim usług i danych przechowywanych w sieci. Trzeba jednak wiedzieć, których narzędzi użyć i jak je skonfigurować.

Książka „Linux. Serwery. Bezpieczeństwo" to podręcznik dla administratorów serwerów, którzy chcą podnieść poziom bezpieczeństwa swoich sieci. Zawiera dokładne opisy narzędzi niezbędnych do zabezpieczenia serwerów oraz praktyczne rady dotyczące ich stosowania. Przedstawia ogólne środki bezpieczeństwa: wykrywanie włamań i filtrowanie pakietów, oraz rozwiązania pozwalające na ochronę konkretnych usług. Czytając ją, dowiesz się, jak projektować strefy DMZ, korzystać z narzędzia iptables i szyfrować dane przesyłane do serwera. Nauczysz się także zabezpieczać serwery DNS, WWW i bazy danych oraz analizować dzienniki systemowe.

- Motywy i cele ataków
- Tworzenie sieci granicznych
- Konfiguracja narzędzia iptables
- Administrowanie zdalne za pomocą SSH
- Zabezpieczanie usługi DNS
- Wykorzystywanie LDAP do uwierzytelniania użytkowników
- Zabezpieczanie bazy danych MySQL oraz poczty elektronicznej
- Bezpieczeństwo serwerów WWW oraz treści witryn internetowych
- Zabezpieczanie serwerów plików
- Monitorowanie dzienników systemowych
- Wykrywanie włamań

Jeśli chcesz, aby administrowany przez Ciebie serwer stał się twierdzą, przeczytaj tę książkę.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 5.

OpenSSL i Stunnel

Ten rozdział mieści się — zarówno w sensie technologicznym, jak i dosłownym — między częścią opisującą zakulisowe mechanizmy a poświęconą usługom; traktuje o pakiecie OpenSSL, który zapewnia usługi szyfrowania i uwierzytelniania wielu narzędziom omówionym w niniejszej książce. OpenSSH, Apache, OpenLDAP, BIND, Postfix i Cyrus IMAP to tylko niektóre spośród aplikacji wykorzystujących OpenSSL.

OpenSSL jest jednak niezwykle skomplikowaną technologią, a jej pełny opis wymagałby oddzielnego tomu (takiego jak *Network Security with OpenSSL* wydawnictwa O'Reilly). Dlatego w tym rozdziale pokażemy tylko, jak używać OpenSSL w konkretnej sytuacji: do osadzania niezaszyfrowanych usług TCP w zaszyfrowanych „tunelach" SSL przy użyciu popularnego narzędzia Stunnel.

Tak się składa, że konfigurowanie programu Stunnel wymaga, abyśmy użyli pakietu OpenSSL do kilku zadań wspólnych dla wszystkich aplikacji zależnych od OpenSSL, które często wykorzystuje się w serwerach bastionowych. Zatem, nawet jeśli samo narzędzie Stunnel okaże się niepotrzebne, warto przeczytać ten rozdział, aby dowiedzieć się, jak generować certyfikaty serwera, zarządzać własnym urzędem certyfikacji itd.

Stunnel i OpenSSL: pojęcia

Mówiąc najprościej, **tunelowanie** polega na osadzaniu pakietów jednego protokołu w pakietach drugiego. W kontekście bezpieczeństwa termin ten zwykle określa osadzanie pakietów niezabezpieczonego protokołu w pakietach zaszyfrowanych[1]. W tym rozdziale pokażemy, jak używać programu Stunnel — „nakładki" na protokół SSL — do przekazywania różnych transakcji sieciowych przez tunele SSL.

Wiele aplikacji sieciowych cechuje się prostotą (pod względem sposobu korzystania z zasobów sieciowych) i użytecznością, ale nie ma mechanizmów bezpieczeństwa takich jak szyfrowanie albo silne (czy choćby odpowiednio realizowane) uwierzytelnianie. Do tej kategorii należały usługi WWW, dopóki firma Netscape Communications nie wynalazła w 1994 roku protokołu Secure Sockets Layer (SSL).

[1] Znawcy sieci mogliby uznać takie użycie terminu **tunelowanie** za nieco naciągane. Zaszyfrowany strumień danych różni się od protokołu sieciowego, a niektórzy twierdzą, że tunelowanie dotyczy protokołów, a nie rozróżnienia między tekstem jawnym a zaszyfrowanym. Uważam jednak, że termin można stosować w tym znaczeniu ze względu na ostateczny rezultat, czyli to, że jeden typ transakcji zostaje osadzony w drugim.

SSL z powodzeniem dodał przezroczyste, ale dobrze zaimplementowane funkcje bezpieczeństwa, do protokołu HTTP, nie utrudniając pracy zwykłym użytkownikom. Umożliwił też uwierzytelnianie klientów i serwerów za pomocą cyfrowych certyfikatów X.509 (choć funkcję uwierzytelniania klientów wykorzystuje się obecnie w niewielkim stopniu). Firma Netscape chciała, żeby SSL stał się standardem internetowym, więc opublikowała wszystkie informacje niezbędne do napisania bezpłatnych bibliotek SSL. Jedną z najpopularniejszych była SSLeay Erica A. Younga, której bezpośredni potomek — OpenSSL — jest nadal używany i rozwijany.

Należy zauważyć, że protokół SSL — choć wciąż powszechny — jest w rzeczywistości przestarzały. Jego następcą jest protokół Transport Layer Security (TLS), który m.in. umożliwia inicjowanie zabezpieczonych (uwierzytelnionych i/lub zaszyfrowanych) połączeń w ramach istniejącej sesji. Inaczej odbywa się to w protokole SSL, w którym uwierzytelnianie i szyfrowanie muszą być inicjowane na początku sesji (właśnie dlatego usługi SSL, takie jak HTTPS, zwyczajowo używają innego portu niż ich odpowiedniki przesyłane tekstem jawnym — na przykład TCP 443 w przypadku HTTPS i TCP 80 w przypadku HTTP — a aplikacje TLS mogą używać tego samego portu do wszystkich transakcji, bez względu na to, czy ruch jest szyfrowany, czy nie).

Protokół SSL nie tylko miał oczywisty wpływ na bezpieczeństwo sieci WWW, ale doprowadził także do powstania programu Stunnel, jednego z najbardziej elastycznych i przydatnych narzędzi open source. Stunnel umożliwia szyfrowanie praktycznie wszystkich jednoportowych usług TCP przy użyciu SSL, bez żadnych modyfikacji w samej usłudze. Przez „jednoportową usługę TCP" rozumiem usługę, która oczekuje na połączenia w jednym porcie TCP i nie wykorzystuje dodatkowych do innych funkcji.

Taką usługą jest protokół HTTP, który oczekuje na połączenia i przeprowadza wszystkie transakcje z wykorzystaniem jednego portu (zwykle TCP 80). Inne przykłady to *rsync*, Syslog-ng, MySQL, a nawet Telnet — wszystkie te usługi mogą działać w szyfrowanych tunelach SSL.

Protokół FTP **nie jest** taką usługą, oczekuje on na połączenia w porcie 21, ale do transmisji danych wykorzystuje inne, losowo wybrane porty. To samo dotyczy usług, które używają wywoływania zdalnych procedur (ang. *Remote Procedure Call*, RPC), ponieważ RPC wykorzystuje program Portmapper, który dynamicznie przydziela przypadkowe porty na użytek połączeń RPC. Często używanymi usługami RPC są NFS i NIS/NIS+; żadna z nich nie działa w połączeniu z programem Stunnel.

Nowsza usługa WebNFS firmy Sun nie wymaga Portmappera; może używać pojedynczego portu TCP (2049), więc jest potencjalnym kandydatem do współpracy z programem Stunnel, ale sam nigdy tego nie robiłem. Więcej informacji o użyciu WebNFS w Linuksie można znaleźć na stronach *nfsd(8)* i *exports(5)* podręcznika systemowego.

Protokół udostępniania plików i drukarek SMB (CIFS) Microsoftu działa w podobny sposób, jeśli zostanie ograniczony do portu TCP 139, więc również może być tunelowany. Więcej informacji można znaleźć w znakomitym dokumencie *Samba HOWTO* napisanym przez Davida Lechnyra i dostępnym pod adresem *http://fluffygerbil.com/docs/samba.txt*. W podrozdziale „Encrypting Access (SSH)" tego dokumentu wyjaśniono, jak Samba działa w takiej konfiguracji — choć autor skupił się na programie SSH, a nie na Stunnel.

OpenSSL

Stunnel wykorzystuje OpenSSL do wszystkich funkcji kryptograficznych. Aby więc używać programu Stunnel, trzeba zainstalować OpenSSL we wszystkich odpowiednich hostach. Bieżące wersje niemal wszystkich dystrybucji Linuksa zawierają pakiety binarne OpenSSL w wersji 0.9.7 lub nowszej. Podstawowy pakiet OpenSSL powinien wystarczyć, ale jeśli pojawią się problemy z kompilowaniem programu Stunnel, należy zainstalować pakiet *openssl-devel* (albo jego odpowiednik używany w danej dystrybucji).

Z biegiem lat wykryto kilka luk w zabezpieczeniach OpenSSL, w tym przepełnienia buforów, podatność na ataki synchronizacyjne, błędy w analizie składniowej ASN.1 oraz zrozumiałe tylko dla wtajemniczonych, ale niebezpieczne usterki kryptograficzne. Podobnie jak w przypadku OpenSSH wynika to raczej z trudności, które są nieodłącznie związane z konstruowaniem bezpiecznego kryptosystemu, a nie z nieudolności zespołu programistów OpenSSL.

Niezwykle **istotne** jest instalowanie wszystkich ukazujących się poprawek zabezpieczeń OpenSSL. Każdy słaby punkt w OpenSSL bezpośrednio wpływa na wszystkie składniki systemu, które używają tego pakietu — Apache, OpenSSH itd.

Ci, którzy zamierzają używać programu Stunnel z certyfikatami klienckimi (tzn. z uwierzytelnianiem opartym na certyfikatach), powinni pobrać i zainstalować najnowszy kod źródłowy OpenSSL (dostępny pod adresem *http://www.openssl.org*), zamiast używać pakietów binarnych. Aby skompilować OpenSSL, należy zdekompresować i rozpakować archiwum *tar*, przejść do podkatalogu *root* w katalogu z kodem źródłowym i uruchomić skrypt *config*. Skryptowi warto przekazać następujące cztery argumenty:

`--prefix=`
: Określa podstawowy katalog instalacyjny (ja używam katalogu */usr/local*).

`--openssldir=`
: Określa katalog macierzysty OpenSSL (często używa się katalogu */usr/local/ssl*).

`shared`
: Nakazuje zbudowanie i zainstalowanie współdzielonych bibliotek OpenSSL, używanych przez Stunnel i OpenSSH.

`zlib-dynamic`
: Nakazuje używać zewnętrznych bibliotek kompresji *zlib*, zamiast nadmiarowo dołączać te funkcje do OpenSSL; w bibliotekach *zlib* również zdarzały się luki w zabezpieczeniach, więc lepiej oddzielić je od OpenSSL (inaczej trzeba będzie rekompilować OpenSSL za każdym razem, kiedy pojawi się nowy problem z bibliotekami *zlib*). Można też użyć opcji *no-zlib*, aby całkowicie zrezygnować z obsługi *zlib*.

Na przykład w razie użycia zalecanych ścieżek polecenie konfiguracyjne wyglądałoby tak:

```
[root openssl-0.9.7d]# ./config --prefix=/usr/local \
--openssldir=/usr/local/ssl shared zlib-dynamic
```

Dalej w tym podrozdziale przyjmujemy założenie, że katalogiem macierzystym OpenSSL jest */usr/local/ssl/*, choć można wybrać dowolny inny.

Dystrybucje binarne OpenSSL w Linuksie Red Hat i SUSE używają */usr/share/ssl/* jako katalogu macierzystego OpenSSL, natomiast Debian używa */usr/lib/ssl/*. Ponieważ korzystam ze wszystkich

trzech dystrybucji, w swoich systemach zwykle tworzę dowiązania symboliczne od */usr/local/ssl/* do rzeczywistego katalogu macierzystego OpenSSL (dlatego m.in. w przykładach zamieszczonych w niniejszym rozdziale używam tej ścieżki).

Jeśli skrypt *config* zostanie wykonany się bezbłędnie, należy wydać polecenie `make`, następnie (opcjonalnie) `make test`, a wreszcie `make install`. Teraz można utworzyć lokalny urząd certyfikacji i przystąpić do generowania certyfikatów.

Jakie funkcje pełni urząd certyfikacji i do czego może być potrzebny?

Program Stunnel używa dwóch typów certyfikatów, serwera i klienta. Kiedy Stunnel działa w trybie demona (to znaczy **bez** opcji `-c`), musi używać certyfikatu serwera. Dystrybucje binarne programu Stunnel często zawierają wstępnie wygenerowany plik *stunnel.pem*, ale nadaje się on **wyłącznie do celów testowych**!

Trzeba więc będzie wygenerować przynajmniej jeden certyfikat serwera. Ci, którzy chcą używać certyfikatów klienta, również będą musieli je utworzyć. Tak czy owak niezbędny będzie urząd certyfikacji (ang. *Cerficate Authority*, CA).

Niektórzy myślą, że urzędem CA może być tylko komercyjna firma, taka jak VeriSign albo Thawte, która wystawia i podpisuje certyfikaty dla serwerów WWW; rzeczywiście, certyfikaty X.509 sprzedawane przez takie firmy działają w OpenSSL i Stunnel. Kiedy użytkownicy (albo ich przeglądarki WWW) muszą weryfikować autentyczność certyfikatu serwera WWW, podpis „neutralnej strony trzeciej", takiej jak komercyjny urząd CA, często jest nieodzowny.

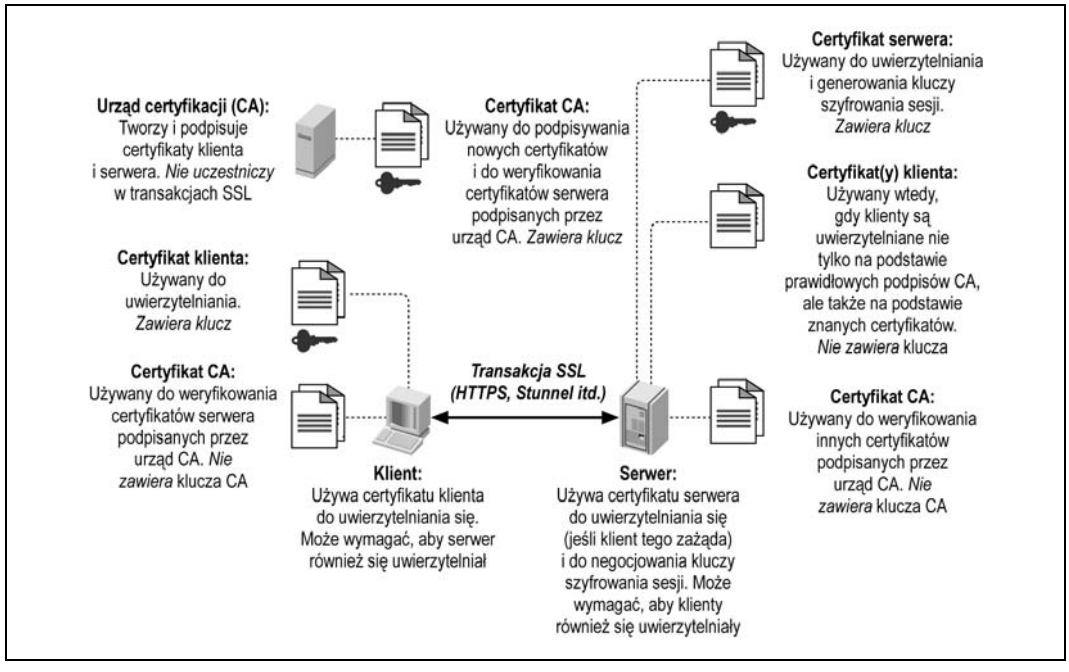
Jest jednak znacznie bardziej prawdopodobne, że weryfikacja certyfikatu dokonywana przez Stunnel będzie polegać na uwierzytelnianiu klientów przez serwer, a nie odwrotnie. W tym modelu zagrożeń nie ma właściwie miejsca na niezależny urząd CA; serwer jest bardziej zagrożony przez nieautoryzowanych użytkowników niż oni przez fałszywy serwer. Jeśli użytkownicy muszą weryfikować autentyczność serwera, prawdopodobnie wystarczy podpis lokalnego, wewnątrzorganizacyjnego „urzędu" CA. Właśnie w takich sytuacjach dobrze jest prowadzić własny urząd certyfikacji.

Jeśli wszystko to wydaje się nieco zagmatwane, na rysunku 5.1 pokazano, jak klienty, serwery i urzędy CA w relacjach SSL korzystają z certyfikatów.

Na rysunku 5.1 zilustrowano kilka ważnych aspektów SSL (i każdej innej infrastruktury kluczy publicznych). Po pierwsze, można zauważyć rozróżnienie między publicznymi **certyfikatami** a prywatnymi **kluczami**. W kryptografii z kluczem publicznym każdy uczestnik komunikacji ma dwa klucze — publiczny i prywatny. Protokół SSL opiera się na kryptografii z kluczem publicznym; w żargonie SSL klucz publiczny określa się mianem certyfikatu, a prywatny po prostu mianem klucza (Czytelnicy, którzy nigdy nie słyszeli o kryptografii z kluczem publicznym, mogą zajrzeć do punktu „Szyfrowanie z użyciem klucza publicznego" w rozdziale 4.).

Jak widać na rysunku 5.1, certyfikaty, nawet urzędów CA, są swobodnie udostępniane. Z kluczami jest inaczej: każdy jest znany tylko właścicielowi i musi być dobrze chroniony, aby związany z nim certyfikat mógł funkcjonować jako unikatowy i weryfikowalny dowód tożsamości.

Innym ważnym wnioskiem z rysunku 5.1 jest to, że urzędy CA **nie biorą bezpośredniego udziału w transakcjach SSL**. Codzienne operacje SSL urzędów CA polegają głównie na podpisywaniu nowych certyfikatów. Zaufanie do tych podpisów jest sprawą tak wielkiej wagi, że im mniej kontaktów ma urząd CA z systemami sieciowymi, tym lepiej.



Rysunek 5.1. Sposób używania certyfikatów przez klienty, serwery i urzędy CA

Jest nie tylko możliwe, ale wręcz pożądane, aby urząd CA nie był podłączony do sieci i akceptował wnioski o podpis oraz eksportował nowe podpisy metodami **ręcznymi** — na przykład na dyskietkach lub dyskach CD-ROM. Minimalizuje to ryzyko skopiowania i nadużycia klucza, którego urząd CA używa do podpisywania certyfikatów; w chwili, w której ktoś zdobędzie dostęp do klucza CA, wszystkie certyfikaty podpisane przez urząd stają się bezużyteczne. Z tej przyczyny główny intranetowy serwer plików nie nadaje się do roli urzędu CA; każdy serwer dostępny publicznie jest absolutnie nie do przyjęcia.

Kiedy host „weryfikuje certyfikat", robi to przy użyciu lokalnej kopii certyfikatu CA, który — jak każdy — sam w sobie nie jest poufny. Jest jednak istotne, aby certyfikaty były kopiowane z jednego hosta do drugiego za pośrednictwem zabezpieczonych kanałów, gdyż uniemożliwia to ich zmodyfikowanie. Choć poufność certyfikatu nie jest istotna, to jego autentyczność ma ogromne znaczenie. Dotyczy to zwłaszcza certyfikatów CA (ponieważ są one używane do sprawdzania autentyczności i ważności innych certyfikatów).

Jak założyć niewielki urząd CA?

Każdy może utworzyć własny urząd CA, używając pakietu OpenSSL na wybranej platformie; kompiluje się on i działa nie tylko w Linuksie i pozostałych Uniksach, ale także w Windows, VMS i innych systemach. Oczywiście wszystkie przykłady w tym rozdziale będą pokazywać działanie OpenSSL w Linuksie. Zważywszy na wagę i poufność operacji wykonywanych przez urząd CA, przed przystąpieniem do pracy należy zalogować się jako *root*, a wszystkie pliki i katalogi CA powinny należeć do użytkownika *root* i mieć ustawiony tryb dostępu 0600 lub 0700.

Najpierw należy zainstalować OpenSSL w sposób opisany w podrozdziale „OpenSSL". W katalogu macierzystym OpenSSL (na przykład */usr/local/ssl*) znajduje się podkatalog o nazwie *misc/*, który zawiera kilka skryptów. Jednego z nich, *CA*, można użyć do automatycznego skonfigurowania hierarchii katalogów CA wraz z plikami indeksowymi i certyfikatem CA (oraz kluczem). W zależności od wersji OpenSSL skrypt *CA* może być napisany w języku powłoki (*CA.sh*), w Perlu (*CA.pl*) albo mieć wersje w nich obu.

Przed uruchomieniem skryptu należy jednak zmodyfikować zarówno jego, jak i plik *openssl.cnf* (umieszczony w katalogu macierzystym OpenSSL) zgodnie z potrzebami lokalnego środowiska. Najpierw w pliku *CA.sh* należy odpowiednio zmodyfikować zmienne umieszczone na początku skryptu. Jedną z godnych uwagi zmiennych jest `DAYS`, która określa czas ważności nowych certyfikatów. Ja zwykle pozostawiam domyślną wartość `-days 365`, ale można wybrać inny okres ważności.

Natomiast zawsze modyfikuję zmienną `CA_TOP`, która określa nazwę drzewa katalogów urzędu CA. Domyślnie jest ona ustawiona na `./demoCA`, ale ja wolę nazwę `./localCA` albo po prostu `./CA`. Początkowe znaki `./` są przydatne: sprawiają one, że skrypt tworzy nowy urząd CA w katalogu roboczym. Można również użyć ścieżki bezwzględnej, ale wówczas trzeba będzie przerabiać skrypt przed utworzeniem nowego urzędu CA, w przeciwnym razie nadpisze on dane starego (w jednym hoście można utworzyć wiele urzędów CA, każdy w oddzielnym drzewie katalogów).

W niektórych systemach (na przykład w Fedorze) skrypt *CA* ignoruje wartość `CA_TOP` w pliku *openssl.cnf* (więc wszystkie nowe katalogi urzędów CA otrzymują nazwę *demoCA*). Aby to zmienić, trzeba będzie ręcznie zmodyfikować skrypt *CA* (albo *CA.sh* lub *CA.pl*).

W pliku *openssl.cnf* można ustawić kilka innych zmiennych, które określają domyślne wartości używane w certyfikatach (zobacz listing 5.1). Są one mniej istotne, ponieważ większość z nich można zmienić podczas tworzenia certyfikatu, ale jedną, `default_bits`, najwygodniej jest zmodyfikować właśnie w pliku *openssl.cnf*. Ustawienie to określa siłę klucza zawartego w certyfikacie i używanego do podpisywania innych certyfikatów, a w przypadku klientów i serwerów SSL (ale nie urzędów CA) do negocjowania kluczy sesji SSL oraz uwierzytelniania jej.

Domyślnie zmienna `default_bits` jest ustawiona na 1024. Postępy w dziedzinie rozkładu dużych liczb na czynniki pierwsze sprawiły, że wartość 2048 jest bezpieczniejsza, choć kosztowna obliczeniowo (ale tylko podczas operacji takich jak generowanie, podpisywanie i weryfikowanie podpisów, a także na początku sesji SSL; nie ma ona wpływu na szybkość transmisji danych). Skrypt *CA* odczytuje plik *openssl.cnf*, więc jeśli certyfikat CA ma zawierać klucz krótszy lub dłuższy niż 1024 bity, należy zmienić ten plik przed uruchomieniem skryptu *CA.pl* lub *CA.sh* (zobacz listing 5.1).

Listing 5.1. Zmienione wiersze przykładowego pliku openssl.cnf

```
# To są jedyne ważne wiersze w tym przykładzie...
dir           = ./CA
default_bits  = 2048

# Zmiana poniższych wartości oszczędza czas podczas generowania nowych certyfikatów.
countryName_default            = ES
stateOrProvinceName_default    = Andalucia
localityName_default           = Sevilla
O.organizationName_default     = Mesòn Milwaukee
organizationalUnitName_default =
```

```
commonName_default              =
emailAddress_default            =

# Nie używam wartości unstructuredName, więc oznaczam ją komentarzem:
# unstructuredName              = Opcjonalna nazwa firmy
```

Teraz należy przejść do katalogu, w którym ma znajdować się hierarchia katalogów urzędu CA. Wiele osób używa */root* albo katalogu macierzystego OpenSSL, którym często bywa */usr/local/ssl*. Należy w nim wydać jedno z poniższych poleceń:

```
[root ssl]# /usr/local/ssl/misc/CA.pl -newca
```

lub:

```
[root ssl]# /usr/local/ssl/misc/CA.sh -newca
```

W obu przypadkach należy zastąpić ścieżkę */usr/local/ssl* katalogiem macierzystym OpenSSL (jeśli jest inny).

Skrypt zapyta o nazwę pliku z istniejącym certyfikatem CA (zobacz listing 5.2); wystarczy nacisnąć klawisz *Enter*, aby stworzyć nowy certyfikat. Następnie pojawi się pytanie o hasło do nowego klucza CA. Hasło to jest niezwykle istotne: każdy, kto je zna i ma dostęp do klucza CA, może podpisywać certyfikaty, których autentyczność będzie dało się pozytywnie zweryfikować. Należy wybrać możliwie długie i skomplikowane hasło. Można w nim stosować odstępy i znaki interpunkcyjne.

Listing 5.2. Sesja CA.pl

```
[root@tamarin ssl]# /usr/local/ssl/misc/CA.pl -newca
CA certificate filename (or enter to create)

Making CA certificate ...
Using configuration from /usr/local/ssl/openssl.cnf
Generating a 2048 bit RSA private key
........++++++
....++++++
Writing new private key to './CA/private/cakey.pem'
Enter PEM pass phrase: *************
Verifying password - Enter PEM pass phrase: *************
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [ES]:
State or Province Name (full name) [Andalucia]:
Locality Name (eg, city) [Sevilla]:
Organization Name (eg, company): [Mesòn Milwaukee]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:ca.mesonmilwaukee.com
Email Address []:certmaestro@mesonmilwaukee.com
```

Domyślnie skrypty *CA.pl* i *CA.sh* tworzą certyfikat CA o nazwie *cacert.pem* w głównym katalogu drzewa urzędu CA (na przykład */usr/local/ssl/CA/cacert.pem*) oraz klucz CA o nazwie *cakey.pem* w katalogu *private/* drzewa urzędu CA (na przykład */usr/local/ssl/CA/private/cakey.pem*). Certyfikat CA trzeba skopiować do każdego hosta, który będzie weryfikował certyfikaty podpisane przez ten urząd CA, ale **klucz** CA musi pozostać w katalogu *private/*. Jego właścicielem musi być użytkownik *root* i tylko on może mieć prawo do odczytu pliku z kluczem.

Teraz można przystąpić do tworzenia i podpisywania własnych certyfikatów. Z technicznego punktu widzenia certyfikaty może generować każdy host z zainstalowanym pakietem OpenSSL, bez względu na to, czy pełni funkcję urzędu CA, czy nie. Jednakże w praktyce lepiej jest robić to właśnie w hoście CA, ponieważ nie trzeba się wówczas martwić o integralność certyfikatów utworzonych gdzie indziej i przesłanych przez niezabezpieczone łącza. Innymi słowy, można spać o wiele spokojniej, podpisując certyfikat wygenerowany lokalnie, a nie przesłany do hosta CA przez internet.

Na użytek programu Stunnel potrzebne będą certyfikaty dla każdego hosta, który będzie pełnił funkcję serwera. Ci, którzy planują uwierzytelniać klienty SSL, będą również potrzebowali certyfikatu dla każdego z nich. Program Stunnel obsługuje dwa typy uwierzytelniania certyfikatów klienta — można zezwolić na połączenia tylko tym klientom, które mają certyfikat podpisany przez zaufany urząd CA, albo dopuszczać tylko te certyfikaty, których kopie są przechowywane w serwerze. W obu typach uwierzytelniania używa się tego samego typu certyfikatu klienta.

Zwykle nie ma różnicy między certyfikatem klienta a serwera, z tym wyjątkiem, że certyfikaty serwera czasem muszą zawierać niezaszyfrowane (tzn. niechronione hasłem) klucze, ponieważ są używane przez zautomatyzowane procesy, natomiast szyfrowanie (chronienie hasłem) certyfikatów klienta jest zazwyczaj pożądane. Jeśli klucz w certyfikacie klienta jest chroniony dobrym hasłem, ryzyko skopiowania go lub wykradzenia jest w znacznej mierze ograniczone.

Z drugiej strony, jeśli aplikacja tunelowana przez Stunnel ma odpowiednie mechanizmy uwierzytelniania albo jeśli proces kliencki Stunnel będzie używany w sposób zautomatyzowany, niezaszyfrowane klucze klienta mogą być uzasadnione. Należy tylko pamiętać, że certyfikaty klienta pozbawione hasła są praktycznie bezużyteczne, jeśli chodzi o uwierzytelnianie użytkowników. Więcej uwag na ten temat można znaleźć w ramce „Certyfikaty bez haseł — zagrożenia".

Przed przystąpieniem do generowania certyfikatów hosta należy skopiować plik *openssl.cnf* z katalogu macierzystego OpenSSL do katalogu urzędu CA i ewentualnie zmodyfikować, aby uwzględnić różnice między certyfikatem CA a następnymi (na przykład można ustawić zmienną `default_bits` na 2048 w przypadku certyfikatu CA, ale używać 1024-bitowych kluczy w certyfikatach serwera i klienta). Na pewno w tej kopii pliku *openssl.cnf* warto ustawić zmienną `dir` na bezwzględną ścieżkę do urzędu CA (na przykład */usr/local/ssl/CA*).

Tworzenie certyfikatów podpisanych przez CA

Teraz można utworzyć certyfikat podpisany przez CA. Zacznijmy od certyfikatu dla serwera Stunnel o nazwie *elfiero*; aby go utworzyć, należy:

1. Przejść do utworzonego wcześniej katalogu urzędu CA (na przykład */usr/local/ssl/CA*).
2. Utworzyć nowy wniosek o podpis (który w rzeczywistości jest certyfikatem) oraz klucz za pomocą polecenia:

   ```
   bash-# openssl req -nodes -new -keyout elfiero_key.pem \
   -out elfiero_req.pem -days 365 -config ./openssl.cnf
   ```

   Można dołączyć opcję `-nodes`, jeśli klucz nowego certyfikatu ma być pozbawiony hasła (niezaszyfrowany). Dzięki temu nie trzeba będzie wpisywać hasła przy każdym uruchomieniu programu, który korzysta z certyfikatu; jednakże przed użyciem opcji `-nodes` należy przeczytać ramkę „Certyfikaty bez haseł — zagrożenia".

Opcja `-keyout` określa nazwę pliku z kluczem, a `-out` nazwę pliku z wnioskiem o podpis (certyfikatem). Plikom tym można nadać dowolne nazwy. Opcja `-days` określa, przez ile dni certyfikat pozostanie ważny. Można ją pominąć, ponieważ jest zdefiniowana również w pliku *openssl.cnf*.

Można również użyć opcji `-newkey rsa:[bity]`, gdzie `[bity]` to rozmiar klucza RSA w nowym certyfikacie, na przykład 1024 albo 2048. Podobnie jak inne opcje ma ona pierwszeństwo przed odpowiednim ustawieniem w pliku *openssl.cnf*.

Następnie użytkownik zostanie poproszony o wprowadzenie nowych (albo zaakceptowanie domyślnych) wartości parametrów „Distinguished Name" (`Country Name`, `Locality Name`, `Common Name` itd.), jak pokazano na listingu 5.2. Każdy certyfikat musi mieć unikatową kombinację parametrów DN; próba utworzenia certyfikatu z takimi samymi parametrami DN jak w poprzednim utworzonym przez dany urząd CA zakończy się błędem. Wystarczy jednak, żeby certyfikaty różniły się jednym polem; ja zwykle zmieniam pola `Email Address` i `Common Name`.

3. Teraz należy podpisać certyfikat za pomocą polecenia:

```
bash-# openssl ca -config ./openssl.cnf -policy policy_anything \
-out elfiero_pubcert.pem -infiles elfiero_req.pem
```

I w tym przypadku można nadać plikowi wyjściowemu dowolną nazwę (określoną za pomocą opcji `-out`). Po wydaniu tego polecenia pojawi się pytanie o hasło do klucza CA, a gdy je wpiszemy — szczegółowe informacje o nowym certyfikacie z prośbą o potwierdzenie operacji.

Czytelnicy, którzy przeszli do tej procedury z rozdziału 9. (tzn. tworzą certyfikat dla serwera SMTP, a nie Stunnel), nie muszą robić nic więcej: wystarczy, że skopiują nowy certyfikat CA, klucz serwera oraz podpisany certyfikat serwera do SMTP i wrócą do procedury w rozdziale 9. Pozostali powinni przejść do 4. etapu operacji.

4. Otworzyć nowy klucz (w tym przykładzie *elfiero_key.pem*) w edytorze tekstu, dodać pusty wiersz na końcu pliku i zapisać go.

Operacja ta nie jest niezbędna w nowych wersjach programu Stunnel, które nie są tak wybredne, jeśli chodzi o format pliku certyfikatu, ale ja wolę dodawać pusty wiersz, bo dzięki temu likwiduje się jeden potencjalny problem (na przykład w przypadku, gdyby lokalna wersja programu Stunnel była starsza niż sądziłem).

5. Otworzyć nowy podpisany certyfikat (w tym przykładzie plik *elfiero_pubcert.pem*) i **usunąć** wszystko, co znajduje się ponad wierszem `-----BEGIN CERTIFICATE-----`. Dodać pusty wiersz na końcu pliku i zapisać go. Również w tym przypadku pusty wiersz może być niepotrzebny, ale na pewno nie zaszkodzi.

6. Połączyć klucz i podpisany certyfikat w jeden plik, w taki sposób:

```
bash-# cat ./elfiero_key.pem ./elfiero_pubcert.pem > ./elfiero_cert.pem
```

To wszystko! Mamy teraz podpisany certyfikat publiczny (*elfiero_pubcert.pem*) i możemy go swobodnie udostępniać, oraz plik z certyfikatem i kluczem (*elfiero_cert.pem*), którego można użyć jako certyfikatu serwera Stunnel w hoście *elfiero*.

Certyfikaty bez haseł — zagrożenia

Wielu ekspertów od bezpieczeństwa uważa, że używanie klucza pozbawionego hasła trąci herezją, niezależnie od zastosowania. Twierdzą oni także, że jeśli jakiś proces jest na tyle poufny, że wymaga szyfrowania z kluczem publicznym, to uzasadnione jest uruchamianie go metodą ręczną (w celu wprowadzenia hasła do certyfikatu serwera tego procesu).

Jeśli na przykład użytkownik skonfiguruje serwer WWW Apache tak, aby używał chronionego hasłem certyfikatu serwera, to będzie pytany o hasło przy każdym uruchamianiu, ale nie będzie musiał wprowadzać go ponownie, dopóki serwer nie zostanie zrestartowany. Program Stunnel może w taki sam sposób używać chronionych hasłem certyfikatów serwera.

Zgodnie z przyjętą praktyką zdecydowałem się opisać tu opcję `-nodes`. Każdy jednak musi zdecydować sam, czy jej użycie w danych okolicznościach jest warte ryzyka, że ktoś przejmie kontrolę nad systemem i wykorzysta klucz do niecnych celów.

Jedna wskazówka: im więcej zastosowań ma dany certyfikat, tym ważniejsze jest to, aby był zaszyfrowany (chroniony hasłem). Jeśli certyfikat będzie używany tylko przez jedną aplikację, dużo łatwiej jest ograniczyć ryzyko związane z brakiem hasła niż w przypadku, gdy ewentualne przejęcie go mogłoby wpłynąć na inne wykorzystujące go procesy.

W powyższej procedurze założyłem, że administrator urzędu CA i administrator serwera to jedna osoba (między innymi dlatego posługuję się terminem „niewielki urząd CA"). Jeśli jednak jedna osoba jest odpowiedzialna za lokalny urząd CA, a druga za serwery, które wymagają podpisanych certyfikatów serwera, ta druga osoba powinna wykonać poniższą procedurę:

1. Utworzyć nowe żądanie podpisu oraz klucz (w opisany wyżej sposób), ale nie w hoście CA tylko w serwerze, w którym certyfikat będzie używany.
2. Przekazać administratorowi urzędu CA kopię wniosku o podpis, ale **nie** klucz, i poprosić go o podpisanie.
3. Sformatować klucz i podpisany certyfikat na użytek programu Stunnel i połączyć je w jeden plik (jak opisano w poprzedniej procedurze).

Tworzenie samodzielnie podpisanych certyfikatów

Użytkownicy, którzy nie zamierzają używać certyfikatów klienta, mogą całkowicie zrezygnować z wątpliwej przyjemności generowania urzędu CA i utworzyć certyfikat **podpisany samodzielnie** (nie przez urząd CA) bezpośrednio w systemie z serwerem, korzystając z lokalnego polecenia `openssl`. Jest to bardzo proste, należy:

1. Przejść do katalogu, w którym będzie zainstalowany certyfikat, na przykład */etc/stunnel*.
2. Utworzyć pojedynczy plik z kluczem i certyfikatem za pomocą polecenia:

```
openssl req -x509 -newkey rsa:1024 -days 365 -keyout stunnel.pem -out stunnel.pem
```

3. Jedyną nową opcją w tym poleceniu jest `-x509`, która nakazuje utworzyć certyfikat w formacie X.509 (program Stunnel wymaga tego w przypadku samodzielnie podpisanych certyfikatów, ale w przypadku podpisanych przez urząd CA już nie). Teraz wystarczy sprawdzić, czy nowy certyfikat ma odpowiednie prawa dostępu (0600, czyli `-rw-------`), i gotowe!

Certyfikaty klienta

Tworzenie certyfikatów dla klientów Stunnel — potrzebne tylko wtedy, gdy serwer Stunnel ma uwierzytelniać klienty za pomocą certyfikatów — nie różni się niczym od tworzenia certyfikatów serwera. Jeśli użytkownik nie poda opcji `-nodes` podczas tworzenia certyfikatu klienta, trzeba będzie wpisywać prawidłowe hasło, aby uruchomić demon klienta Stunnel. Kiedy jednak demon zacznie działać, każdy lokalny użytkownik w komputerze klienckim będzie mógł korzystać z tunelu[2] (choć **nadal będzie** obowiązywać uwierzytelnianie wymagane przez tunelowaną aplikację).

Z perspektywy serwera Stunnel certyfikat klienta uwierzytelnia system kliencki Stunnel, a nie użytkowników tunelowanej aplikacji. Spostrzeżenie to dotyczy wszystkich serwerów, które przyjmują połączenia, opierając się na certyfikatach z niezabezpieczonymi kluczami, albo korzystają ze współużytkowanych demonów klienta.

Używanie programu Stunnel

Po utworzeniu przynajmniej jednego certyfikatu serwera można przystąpić do konfigurowania klienta (lub klientów) oraz serwera Stunnel. Większość dystrybucji Linuksa zawiera binarny pakiet Stunnel; nowe wersje systemów SUSE, Fedora, Debian oraz Red Hat Enterprise zawierają Stunnel w wersji 4. Stunnel 3.26 to stabilna wersja, która jest dobrze udokumentowana i rozumiana. Z drugiej strony Stunnel 4 to wersja w dużej mierze przepisana, która m.in. ułatwia zarządzanie wieloma tunelami; właśnie ją tutaj omówimy. Użytkowników starszych wersji Debiana zachęcam do pobrania najnowszych źródeł programu Stunnel z witryny *http://www.stunnel.org* i samodzielnej ich kompilacji, choć ta dostępna standardowo w wersji 3.0 („Woody") tej dystrybucji powinna być bezpieczna.

Kompilowanie programu Stunnel jest szybkie i łatwe w każdej dystrybucji Linuksa. Najpierw trzeba zainstalować dołączony do dystrybucji pakiet OpenSSL (prawdopodobnie o nazwie *openssl*), biblioteki programistyczne OpenSSL (*openssl-devel*, *openssl-dev* lub *libssl097-dev*) oraz TCP-wrapper (pakiet *libwrap0-dev* w Debianie; biblioteki te wchodzą w skład podstawowej instalacji SUSE i Fedory).

Następnie należy rozpakować archiwum z kodem źródłowym programu Stunnel i wydać polecenie `./configure && make && make install`. Jeśli z jakieś przyczyny to nie zadziała, za pomocą polecenia `./configure --help` można wyświetlić opcje, które można przekazać skryptowi `configure` — na przykład `--without-tcp-wrappers`.

Po zainstalowaniu programu Stunnel można utworzyć certyfikaty i rozpocząć tunelowanie!

[2] Program iptables zawiera nowy moduł dopasowywania o nazwie *owner*, który może ograniczać dostęp lokalnych użytkowników do demonów sieciowych. Jeśli jądro w komputerze z klientem Stunnel obsługuje iptables, można dodać do łańcuchów INPUT i OUTPUT reguły, które zezwalają na dostęp do lokalnego portu Stunnel (na przykład *localhost:ssync*) tylko grupom albo użytkownikom o określonym identyfikatorze; służą do tego opcje `--gid-owner` oraz `--uid-owner` programu iptables. Jednakże moduł *owner*, który oferuje te opcje, ma nadal charakter eksperymentalny i trzeba go włączać podczas kompilacji niestandardowego jądra. Moduł ma nazwę *ipt_owner.o*, „Owner Match Support (EXPERIMENTAL)" w skrypcie konfiguracji jądra. Książka *Linux in a Nutshell* (O'Reilly) zawiera dokumentację programu iptables, w tym modułu *owner*.

Aby zobaczyć listę opcji konfiguracyjnych, z którymi zbudowano plik binarny Stunnel, należy wydać polecenie `stunnel -version`. Jest to przydatne szczególnie wtedy, gdy program Stunnel zainstalowano z pakietu binarnego i nie wiadomo, w jaki sposób został skompilowany. Rozwiązywanie ewentualnych problemów jest łatwiejsze, gdy wiadomo, gdzie Stunnel szuka różnych plików i opcji.

Przykład użycia programu Stunnel

Po tym długim wstępie wreszcie doszliśmy do sedna sprawy — uruchamianie programu Stunnel i tunelowania ruchu. W wersjach starszych niż 4 całą konfigurację programu Stunnel określało się w wierszu polecenia, na przykład `stunnel -c -d rsync -r ssyncd -N ssync`.

W bieżących wersjach (4.0 i nowszych) Stunnel używa pliku konfiguracyjnego *stunnel.conf*. W rzeczywistości położenie tego pliku jest obecnie jedynym parametrem, jaki można określić za pomocą opcji polecenia `stunnel`. Jeśli program Stunnel zbudowano z kodu źródłowego przy użyciu domyślnych opcji, ścieżką do tego pliku jest */usr/local/etc/stunnel/stunnel.conf*. Jeśli jednak zainstalowano go z pakietu binarnego, bardziej prawdopodobna jest ścieżka */etc/stunnel/stunnel.conf*.

Zanim podamy szczegółowy opis parametrów zawartych w pliku *stunnel.conf*, przedstawimy przykładowy scenariusz, który pokazuje, jak utworzyć prosty tunel.

Przypuśćmy, że mamy dwa serwery, *skillet* i *elfiero*. *elfiero* to serwer *rsync*, a my chcemy tunelować sesje *rsync* między hostami *skillet* i *elfiero*. Najprostszym sposobem użycia serwera *rsync*, jak wyjaśniono w rozdziale 11., jest wydanie polecenia `rsync` *`nazwahosta`*`::`, które nakazuje hostowi *`nazwahosta`* przesłać listę jego anonimowych modułów (udziałów). W tym przykładzie naszym celem będzie wydanie tego polecenia w ramach sesji Stunnel.

Najpierw trzeba zainstalować *rsync* w hoście *elfiero*, skonfigurować go i uruchomić w trybie demona (przypuśćmy, że zrobiliśmy to zgodnie ze wskazówkami podanymi w rozdziale 11., dzięki czemu demon *rsync* w hoście *elfiero* stał się tak stabilny i bezpieczny, że jest przedmiotem zawiści całej lokalnej grupy użytkowników *rsync*).

Teraz trzeba zmienić to i owo w hoście *elfiero*, aby Stunnel mógł działać w trybie demona. Najważniejszy jest certyfikat serwera sformatowany tak, jak to opisano w podpunktach „Tworzenie certyfikatów podpisanych przez CA" oraz „Tworzenie samodzielnie podpisanych certyfikatów". W tym przykładzie certyfikat nosi nazwę *elfiero_cert.pem*, został skopiowany do katalogu */etc/stunnel* i ma prawa dostępu 600 (`-rw-------`).

Trzeba również dokonać kilku niewielkich zmian w istniejących plikach na serwerze: w pliku */etc/services* warto dodać wpis dla portu, w którym Stunnel będzie czekał na połączenia, ponieważ dzięki temu wpisy dziennika i polecenia będą czytelniejsze. W naszym przykładzie należy dodać do pliku */etc/services* następujący wiersz:

```
ssyncd      273/tcp         # Zabezpieczony demon rsync
```

(„Prawdziwy" demon *rsync* nasłuchuje na porcie TCP 873, więc lubię używać portu o podobnym numerze).

Ponadto na potrzeby tego przykładu załóżmy, że program Stunnel skompilowano z obsługą *libwrap*; należy więc do pliku */etc/hosts.allow* dodać poniższy wiersz:

```
ssync: ALL
```

W systemie Red Hat wpis w pliku */etc/hosts.allow* powinien mieć postać:

```
ssync: ALL: ALLOW
```

Następnie należy zmodyfikować plik */etc/stunnel/stunnel.conf* w hoście *elfiero* (lub */usr/local/etc/stunnel/stunnel.conf*, jeśli program został zainstalowany z kodu źródłowego). Na listingu 5.3 pokazano ustawienia, które nakazują programowi Stunnel używać certyfikatu serwera */etc/stunnel/elfiero_cert.pem*, działać w trybie serwera, używać *ssync* jako nazwy usługi TCPwrappers, oczekiwać na zaszyfrowane pakiety w porcie *ssyncd* (TCP 273) i przekazywać odszyfrowane pakiety do lokalnego portu *rsync*.

Listing 5.3. Plik stunnel.conf w serwerze Stunnel

```
cert = /etc/stunnel/elfiero_cert.pem
client = no
[ssync]
    accept = ssyncd
    connect = rsync
```

Teraz wystarczy uruchomić Stunnel w hoście *elfiero* za pomocą polecenia `stunnel`. Nie ma znaczenia, czy program najpierw zostanie uruchomiony w serwerze, czy też w kliencie; klient nie zainicjuje tunelu, dopóki ktoś go nie użyje. Jeśli certyfikat serwera *elfiero* jest chroniony hasłem, pojawi się prośba o jego wprowadzenie (trzeba o tym pamiętać podczas pisania skryptu uruchamiającego serwer). Po wpisaniu właściwego hasła serwer powinien być gotowy do przyjmowania połączeń!

Teraz można sprawdzić, czy uruchomienie serwera się powiodło. W tym celu należy wydać polecenie `ps auxw` i poszukać procesu *stunnel*; program nie wyświetla niczego na konsoli bez względu na to, czy uruchomi się pomyślnie, czy nie. Wysyła jednak komunikaty do systemowego mechanizmu syslog (domyślnie do kanału *daemon*), w tym komunikaty startowe.

Czas skonfigurować system kliencki *skillet*. Na razie nie planujemy używać certyfikatów klienta ani weryfikować certyfikatu serwera, więc mamy nieco mniej do zrobienia. Wystarczy dodać jeden wiersz do pliku */etc/services* i jeden wpis do */etc/hosts.allow* (to ostatnie jest potrzebne tylko wtedy, gdy program Stunnel w komputerze *skillet* został skompilowany z obsługą *libwrap*).

Aby zachować spójność, wiersz dodawany do pliku */etc/services* powinien być identyczny jak ten dodany w hoście *elfiero*:

```
ssyncd      273/tcp         # Zabezpieczony demon rsync
```

Byłoby najlepiej, gdyby program Stunnel w hoście *skillet* działał w porcie TCP 873, czyli *rsync*, aby lokalne klienty *rsync* mogły używać domyślnego portu podczas łączenia się przez tunel. Jeśli jednak w systemie klienckim działa już samodzielny demon *rsync* w porcie TCP 873, można dodać do pliku */etc/services* kolejny wiersz, aby zdefiniować port przekaźnika *rsync*:

```
zsync       272/tcp         # Zabezpieczony przekaźnik rsync
```

Podczas określania portów dla nowych usług, takich jak Stunnel, należy uważać, aby nie wybrać zajętego przez inną aktywną usługę (dzięki temu później nie trzeba będzie się zastanawiać, dlaczego nowa usługa nie chce się uruchomić!).

Aby wyświetlić wszystkie aktywne, nasłuchujące gniazda TCP/IP, należy wydać polecenie `netstat --inet -aln` (numery aktywnych portów lokalnych są wyświetlane za dwukropkiem w kolumnie „Local Address"). Polecenie to ma taką samą postać we wszystkich odmianach Linuksa.

Co to jest „kontrola dostępu w stylu TCPwrappers" i jak jej używać?

Nie opisałem jeszcze programu TCPwrappers, popularnego narzędzia rejestrującego i kontrolującego dostęp do usług uruchamianych za pośrednictwem demona *inetd*, głównie dlatego, że przydatność *inetd* w hoście bastionowym jest ograniczona (aby dowiedzieć się, czemu tak uważam, należy przeczytać podpunkt „Tryb inetd (xinetd) a tryb samodzielny" w rozdziale 11.).

Jednakże program TCPwrappers ma mechanizm kontroli dostępu, który ogranicza połączenia przychodzące na podstawie adresów IP zdalnych klientów, co jest przydatnym sposobem zwiększania bezpieczeństwa aplikacji. Mechanizm ten, który będę nazywał „kontrolą dostępu w stylu TCPwrappers", jest obsługiwany przez Stunnel i wiele innych samodzielnych usług poprzez bibliotekę *libwrap.a* programu TCPwrappers.

Mechanizm ten wykorzystuje dwa pliki, */etc/hosts.allow* i */etc/hosts.deny*. Kiedy host kliencki próbuje połączyć się z jakąś chronioną usługą, jego adres jest najpierw porównywany z zawartością pliku */etc/hosts.allow*. Jeśli adres zostanie dopasowany do któregokolwiek wiersza w tym pliku, połączenie jest akceptowane. Jeśli natomiast nie pasuje do żadnego, sprawdzany jest plik */etc/hosts.deny*. Jeśli adres zostanie dopasowany do któregokolwiek wiersza w tym pliku, połączenie jest odrzucane. Jeśli adres IP klienta nie zostanie znaleziony w **żadnym** z tych dwóch plików, połączenie jest akceptowane.

Ponieważ ta metoda **domyślnego zezwalania** na połączenie nie jest zbyt bezpieczna, większość osób stosuje politykę **domyślnego blokowania** połączeń, umieszczając w pliku */etc/hosts.deny* tylko jeden wiersz:

```
ALL: ALL
```

Dzięki temu dostęp jest kontrolowany jedynie przez plik */etc/hosts.allow*: każda kombinacja usługi i adresu IP, która nie jest wymieniona w tym pliku, zostanie odrzucona.

Najprostszy wpis w pliku *hosts.allow* (i *hosts.deny*) składa się z dwóch pól:

```
demon1 [demon2 itd.] : host1 [host2 itd.]
```

Pierwsze pole jest tu listą nazw demonów (rozdzieloną spacjami lub przecinkami), a drugie (poprzedzone dwukropkiem) listą adresów IP (również rozdzieloną spacjami lub przecinkami).

Nazwa demona jest zwykle ustalana na podstawie wartości `argv[0]` przekazywanej przez niego do powłoki, w której został uruchomiony. W przypadku programu Stunnel nazwa ta albo wywodzi się z opcji `-N` przekazanej mu podczas uruchamiania, albo jest kombinacją nazwy tunelowanej usługi oraz nazwy hosta, z którym Stunnel się łączy. Można również użyć symbolu wieloznacznego `ALL`.

Adresy IP hostów można podać w całości lub częściowo, na przykład `10.200` pasuje do wszystkich adresów IP z zakresu od `10.200.0.1` do `10.200.254.254`. Można również użyć symbolu wieloznacznego `ALL`.

W systemie Red Hat (i każdym innym, w którym demon *tcpd* został skompilowany z opcją *PROCESS_OPTIONS*) używa się trzeciego pola, poprzedzonego kolejnym dwukropkiem i najczęściej ustawianego na `ALLOW` lub `DENY`. Eliminuje ono potrzebę korzystania z pliku */etc/hosts.deny*: zarówno reguły `ALLOW`, jak i `DENY` mogą być określone w pojedynczym pliku */etc/hosts.allow*.

Więcej informacji można znaleźć na stronach *hosts_access(5)* i *hosts_options(5)* podręcznika systemowego.

Jeśli pakiet Stunnel w hoście *skillet* został skompilowany z obsługą *libwrap*, trzeba dodać poniższy wiersz do pliku */etc/hosts.allow*:

```
ssync: ALL
```

A w wersji *libwrap* używanej w Red Hacie (*PROCESS_OPTIONS*):

```
ssync: ALL: ALLOW
```

Plik *stunnel.conf* w hoście *skillet* będzie bardzo podobny do tego w hoście *elfiero*, z tym że parametr `client` należy ustawić na `yes`, a `accept` i `connect` powinny być odwrócone. Na listingu 5.4 pokazano wszystkie niezbędne parametry *stunnel.conf*, które nakazują programowi Stunnel uruchomić się w trybie klienta, używać nazwy usługi TCPwrappers *ssync*, nasłuchiwać lokalnych połączeń na porcie *rsync* (TCP 873) i przekazywać je do portu *ssyncd* (TCP 273) w hoście *elfiero*:

Listing 5.4. Plik stunnel.conf w kliencie Stunnel

```
client = yes
[ssync]
    accept = rsync
    connect = elfiero.mesonmilwaukee.com:ssyncd
```

(Jeśli widok niewyjaśnionych parametrów pliku *stunnel.conf* na listingach 5.3 i 5.4 kogoś denerwuje, zapewniam, że opiszę je w następnym podrozdziale z charakterystyczną dla mnie rozwlekłością).

W hoście *skillet* pozostaje tylko uruchomić program Stunnel, tak jak poprzednio, za pomocą polecenia `stunnel`.

Wreszcie dotarliśmy do mety: czas wywołać program *rsync*. Polecenie, które pobiera listę modułów z hosta *elfiero*, zwykle ma następującą postać:

```
[schmoe@skillet ~]$ rsync elfiero::
```

W rzeczywistości nic co zrobiliśmy do tej pory, nie zapobiegłoby działaniu tego polecenia (blokowanie nietunelowanego dostępu do serwera wykracza poza ramy tego krótkiego przykładu).

Zrobimy jednak coś ciekawszego: połączymy się z **lokalnym** procesem, który w niezauważalny sposób przekaże polecenie w ramach zaszyfrowanej sesji z hostem *elfiero*, a odpowiedź z *elfiero* przyjdzie tym samym zaszyfrowanym kanałem. Na listingu 5.5 przedstawiono przykład takiej wymiany (warto podkreślić, że nie trzeba być administratorem, aby uruchomić aplikację klienta).

Listing 5.5. Uruchamianie rsync poprzez Stunnel

```
[schmoe@skillet ~]$ rsync localhost::
narzedzia         Bezpłatne oprogramowanie do organizowania przepisów
przepisy          Pączki, kremówki, tempura, hot dogi, żeberka wieprzowe itd.
fotografie        Zdjęcia sławnych kucharzy w niecodziennych pozach
medyczne          Adresy gabinetów angioplastycznych
```

Zadziałało! Teraz osoby, które mają konto w hoście *skillet*, mogą pobierać przepisy na niezdrowe dania z hosta *elfiero* zabezpieczeni przed czujnym wzrokiem urzędników z Ministerstwa Zdrowia.

Przy okazji, gdybyśmy użyli niestandardowego portu *rsync* dla procesu Stunnel w kliencie (na przykład ustawiając parametr `connect` z listingu 5.5 na *zsync* zamiast *rsync*), przykład wyglądałby tak jak na listingu 5.6.

Listing 5.6. Uruchamianie rsync poprzez Stunnel (niestandardowy port rsync)

```
[schmoe@skillet ~]$ rsync --port=272 localhost::
narzedzia         Bezpłatne oprogramowanie do organizowania przepisów
przepisy          Pączki, kremówki, tempura, hot dogi, żeberka wieprzowe itd.
fotografie        Zdjęcia sławnych kucharzy w niecodziennych pozach
```

Innymi słowy, polecenie `rsync` może łączyć się z dowolnym portem, ale jeśli nie jest to port 873, trzeba określić go za pomocą opcji `--port=`. Polecenie `rsync` nie odczytuje pliku */etc/services*, więc trzeba podać numer zamiast nazwy.

To był szybki start. Teraz podwińmy rękawy, przeanalizujmy to, co zrobiliśmy do tej pory i omówmy dodatkowe zastosowania programu Stunnel.

Wyjaśnienie ustawień w przykładowym pliku stunnel.conf

Jak pokazano przed chwilą, Stunnel używa pojedynczego pliku binarnego, który może działać w dwóch różnych trybach — klienta i serwera. Oba tryby są podobne, ale jest jeden ważny wyjątek: w trybie klienta Stunnel oczekuje na niezaszyfrowane połączenia (z lokalnego komputera) i przekazuje je przez zaszyfrowane połączenie SSL do programu Stunnel, który działa w zdalnym komputerze; w trybie serwera Stunnel oczekuje na zaszyfrowane połączenia SSL (od zdalnych procesów Stunnel), odszyfrowuje je i przekazuje do lokalnych procesów. Parametry w plikach *stunnel.conf* pokazanych na listingach 5.3 i 5.4 są więc bardzo podobne; różny jest głównie **sposób** ich użycia.

Oto opis parametrów użytych w plikach *stunnel.conf* z listingów 5.3 i 5.4:

`client = yes | no`
: Opcja `-c` nakazuje programowi *stunnel* działać w trybie klienta i odpowiednio interpretować inne znaczniki i opcje (na przykład `-d` i `-r`). Bez tej opcji program uruchamia się w trybie demona.

`cert = /ścieżka/do/certyfikatu.pem`
: Ta opcja określa pełną ścieżkę do certyfikatu hosta. W trybie klienta jest potrzebna tylko wtedy, gdy komputer musi przedstawiać swój certyfikat serwerom, z którymi się łączy. Certyfikat jest zawsze niezbędny w trybie serwera.

`[nazwa_usługi]`
: Ta etykieta, umieszczona w nawiasie kwadratowym, wskazuje początek definicji usługi, a ponadto określa jej nazwę, którą *stunnel* przekazuje w wywołaniach *libwrap* (w celu porównania z wpisami w pliku */etc/hosts.allow*). Wszystkie parametry **powyżej** pierwszej definicji usługi mają zastosowanie globalne. Definicja kończy się albo nazwą następnej usługi, albo końcem pliku (w zależności od tego, co występuje najpierw).

`accept [IP_hosta:]port_demona`
: Opcja `accept` określa, pod jakim adresem IP i w którym porcie program `stunnel` powinien oczekiwać na połączenia. Parametr *`IP_hosta`* — lokalny adres IP albo nazwa hosta, którą da się przetłumaczyć na adres IP — określa, pod jakim lokalnym adresem (albo nazwą hosta) program Stunnel ma oczekiwać na połączenia. Parametr *`port_demona`* może być numerem portu TCP albo nazwą usługi wymienioną w pliku */etc/services*. W trybie serwera opcja ta zwykle służy do określania portu, w którym program ma oczekiwać na zaszyfrowane (tunelowane) pakiety. W trybie klienta opcja ta określa port, w którym program ma oczekiwać na niezaszyfrowane pakiety (przed tunelowaniem).

`connect [zdalny_IP:]zdalny_port`
: Opcja `connect` określa, do jakiego portu program Stunnel powinien przekazywać pakiety. W trybie serwera jest to lokalny port, do którego należy przekazywać pakiety odebrane z portu `accept` (po rozszyfrowaniu). W trybie klienta jest to port, w którym zdalny system (określony przez parametr *zdalny_IP*, który może być adresem IP albo nazwą hosta) oczekuje na tunelowane połączenia. Ponieważ parametr *zdalny_IP* ma domyślną wartość `localhost`, można pomijać go w serwerach Stunnel.

Warto zauważyć, że za pomocą parametru `accept` można wskazać interfejs, przez który Stunnel przyjmuje połączenia. A co z samą usługą „docelową"? Jeśli niektóre połączenia *rsync* mają być szyfrowane, to prawdopodobnie chcemy, aby były to **wszystkie**. Różne aplikacje sieciowe różnie sobie z tym radzą, ale aby poinformować program *rsync*, że ma przyjmować połączenia tylko od lokalnych procesów (tzn. od programu *stunnel*), należy wywołać go w następujący sposób:

```
rsync --daemon --address=127.0.0.1.
```

Oczywiście nie wszystkie usługi pozwalają określić lokalne adresy IP, pod którymi czekają na połączenia. W takich przypadkach można użyć pewnej kombinacji pliku *hosts.allow*, programu iptables oraz uwierzytelniania opartego na certyfikatach (zobacz dalszy punkt „Uwierzytelnianie przy użyciu certyfikatów").

Ustawienia globalne zwiększające poziom bezpieczeństwa

Powyższy krótki przykład pokazywał, jak szybko utworzyć prosty tunel. Program Stunnel w wersji 4 obsługuje jednak dodatkowe parametry pliku *stunnel.conf*, które znacznie zwiększają jego poziom bezpieczeństwa, pozwalając uruchomić Stunnel w środowisku `chroot` i wykonywać go z identyfikatorami nieuprzywilejowanego użytkownika i grupy. Parametry te mają charakter globalny, więc powinny poprzedzać wszystkie definicje usług.

`chroot = /ścieżka/do/klatki/chroot`
: Parametr ten nakazuje programowi Stunnel traktować wskazany katalog jako katalog główny po odczytaniu pliku konfiguracyjnego i certyfikatu hosta (jeśli jest używany), ale przed zapisaniem identyfikatora PID, przeanalizowaniem plików *hosts.allow* i *hosts.deny* albo uwzględnieniem parametrów `exec` (zobacz listing 5.7). W „więzieniu" `chroot` trzeba utworzyć (lub skopiować do niego) pliki *etc/hosts.allow*, *etc/hosts.deny* oraz pliki procesów, które mają być wykonywane przez Stunnel.

`setuid = nazwa` lub `identyfikator_użytkownika`
: Określa nazwę albo liczbowy identyfikator nieuprzywilejowanego użytkownika, z którego uprawnieniami ma działać program Stunnel. Może to wpłynąć na niektóre operacje wykonywane przez Stunnel — takie jak zapisywanie pliku PID albo uruchamianie demonów określonych za pomocą parametru `exec`.

`setgid = nazwa` lub `identyfikator_grupy`
: Określa nazwę albo liczbowy identyfikator nieuprzywilejowanej grupy, z której uprawnieniami ma działać program Stunnel.

Inne globalne i specyficzne dla usług parametry *stunnel.conf* są opisane na stronie *stunnel(8)* podręcznika systemowego.

Inna metoda używania programu Stunnel w serwerze

W przykładzie z hostami *skillet* i *elfiero* program Stunnel działał w trybie serwera. Oprócz trybu serwera i klienta Stunnel może działać w trybie Inetd. W tym trybie proces *inetd* w serwerze uruchamia demon Stunnel (oraz usługę, która jest tunelowana przez Stunnel) za każdym razem, kiedy odbiera połączenie przez określony port. Opis tej konfiguracji można znaleźć w dokumencie Stunnel FAQ (*http://www.stunnel.org/faq/*) oraz na stronie *stunnel(8)* podręcznika systemowego.

Nie będę tu dokładniej opisywał uruchamiania programu Stunnel w trybie Inetd: powiedziałem już, że jestem przeciwny używaniu Inetd w hostach bastionowych. Na dowód, że nie są to tylko moje uprzedzenia, przytaczam cytat z dokumentu Stunnel FAQ:

> Tryb demona (serwera) jest znacznie lepszy od trybu Inetd. Dlaczego?
>
> - każde połączenie wymaga inicjalizacji SSL;
> - nie jest możliwe buforowanie sesji;
> - praca w trybie Inetd wymaga rozwidlania procesów, co wiąże się z dodatkowymi kosztami; proces działający w trybie serwera nie będzie się rozwidlał, jeśli program Stunnel skompilowano z obsługą wątków.

Zamiast uruchamiać Stunnel z pliku *inetd.conf*, dużo lepiej jest obsługiwać demony typu Inetd — takie jak *in.telnetd* oraz *in.talkd* — uruchamiając je za pomocą demona Stunnel przy użyciu parametru `exec`, a nie `connect` w definicji usługi.

Aby na przykład utworzyć zabezpieczoną usługę Telnet w hoście *elfiero*, można użyć metody opisanej w poprzednim punkcie. Jednakże linuksowy demon *in.telnetd* nie jest właściwie przeznaczony do pracy w trybie samodzielnym, z wyjątkiem celów diagnostycznych. Lepiej będzie zatem użyć w serwerze Stunnel definicji usługi takiej jak pokazana na listingu 5.7 (na potrzeby tego przykładu załóżmy, że w każdym hoście dodano już do pliku */etc/hosts.allow* wpis dla usługi *telnets*).

Listing 5.7. Definicja usługi telnets w serwerze

```
[telnets]
accept = telnets
exec = /usr/sbin/in.telnetd
execargs = /usr/sbin/in.telnetd
```

Parametr `exec` informuje, jaki lokalny proces należy uruchomić, aby przekazywać do niego odszyfrowane pakiety. Należy zauważyć, że jeśli użyto opcji `chroot`, aby uruchomić Stunnel w środowisku `chroot`, wszystkie ścieżki podane w instrukcjach `exec` będą interpretowane względem ścieżki `chroot`. Parametr `execargs` to rozdzielona spacjami lista argumentów przekazywanych procesowi `exec`, poczynając od `$0` (nazwy procesu). Jeśli nawet proces nie potrzebuje żadnych innych argumentów, trzeba użyć instrukcji `execargs`, aby poinformować program Stunnel, jaką nazwę procesu ma przekazać w argumencie `$0`; instrukcje `exec` i `execargs` zawsze idą w parze.

W systemie klienckim można po prostu uruchomić klient Telnetu, który obsługuje *telnets* (istnieją takie). Można też uruchomić program Stunnel w trybie klienta, używając takiej definicji usługi jak pokazana na listingu 5.8.

Mogłoby się wydawać, że pominąłem pewien etap, mianowicie dodawanie wiersza usługi *telnets* do pliku */etc/services*. Tak się jednak składa, że organizacja Internet Assigned Numbers Authority (IANA) wyznaczyła już numery portów dla usług chronionych przez SSL, przy czym port TCP 992 przypisano usłudze *Telnets* (ang. *Telnet secure*). Zatem w większości systemów linuksowych ta kombinacja nazwy usługi i numeru portu znajduje się już w pliku */etc/services*.

Szybkim i łatwym sposobem wyświetlenia listy portów przypisanych przez IANA usługom SSL jest wydanie poniższego polecenia:

```
bash-# grep SSL /etc/services
```

Pełną bieżącą listę numerów portów IANA można znaleźć w internecie pod adresem *http://www.iana.org/assignments/port-numbers*.

Listing 5.8. Definicja usługi telnets w kliencie

```
client = yes
[telnets]
accept = 127.0.0.1:telnets
connect = elfiero:telnets
```

Następnie można użyć standardowego linuksowego polecenia `telnet`, aby połączyć się z lokalnym przekaźnikiem Stunnel:

```
[schmoe@skillet ~]$ telnet localhost telnets
```

Pomijając standardową sesję Telnetu, która rozpoczyna się po wydaniu tego polecenia, w przykładzie tym zachodzi, co następuje:

1. Proces *telnet* łączy się z lokalnym procesem Stunnel, który działa w trybie klienta i oczekuje na połączenia w porcie TCP 992.
2. Proces Stunnel działający w trybie klienta otwiera zaszyfrowany tunel SSL do procesu Stunnel, który działa w trybie serwera i oczekuje na połączenia na porcie TCP 992 zdalnego systemu.
3. Po utworzeniu tunelu zdalny (działający w trybie serwera) proces Stunnel uruchamia swój lokalny demon *in.telnetd*.
4. Proces Stunnel działający w trybie klienta przekazuje sesję Telnetu przez tunel, a zdalny demon Stunnel przekazuje pakiety Telnetu do uruchomionej przez siebie usługi *in.telnetd*.

Przy okazji powiem, jeśli jeszcze tego nie wyjaśniłem, że procesy klienta i serwera Stunnel **mogą oczekiwać na połączenia na portach o różnych numerach**. Wystarczy tylko upewnić się, że:

- w każdym hoście wybrano port, na którym nie nasłuchuje żaden inny proces;
- demon klienta **wysyła** dane do tego samego portu, na którym demon serwera **nasłuchuje** (to znaczy port określony w kliencie za pomocą opcji `connect` pasuje do portu określonego w serwerze za pomocą opcji `accept`).

Dwie ważne uwagi dotyczące usługi *telnets*: po pierwsze demon *in.telnetd* używa kilku różnych plików systemowych i specjalnych, więc uruchomienie go przy użyciu procesu *stunnel*, który działa w środowisku `chroot`, jest trudne; prawdopodobnie nie uda się użyć parametru `chroot` w konfiguracji tunelowanego Telnetu. Ponieważ zaś *in.telnetd* musi być uruchomiony przez administratora (albo proces działający z jego uprawnieniami), nie będzie można również użyć parametrów `setuid` ani `setgid`.

Uwierzytelnianie przy użyciu certyfikatów

Przekazywanie niezabezpieczonych aplikacji przez zaszyfrowane tunele SSL jest godne polecenia. Jeszcze lepsze jest korzystanie z programu Stunnel w połączeniu z uwierzytelnianiem przy użyciu cyfrowych certyfikatów X.509.

Zła wiadomość jest taka, że trudno jest znaleźć jasną i konsekwentną dokumentację poświęconą temu zagadnieniu, a dobra, że samo **używanie** cyfrowych certyfikatów nie jest trudne, a zamieszczone niżej wskazówki i procedury (wraz z podanymi wcześniej informacjami dotyczącymi OpenSSL) powinny pozwolić na bezproblemową konfigurację uwierzytelniania.

Program Stunnel może używać certyfikatów X.509 na kilka różnych sposobów, co określa się za pomocą globalnego parametru `verify` w pliku *stunnel.conf*. Parametr ten może przybrać jedną z trzech wartości:

1. Jeśli zdalny host przedstawia certyfikat, sprawdzany jest jego podpis.
2. Przyjmowane są tylko połączenia od hostów, które przedstawiają certyfikaty podpisane przez zaufany urząd CA.
3. Przyjmowane są tylko połączenia od hostów, które przedstawiają certyfikaty zarówno **buforowane lokalnie** (to znaczy znane), jak i podpisane przez zaufany urząd CA.

W rzeczywistości istnieje jeszcze czwarty poziom weryfikacji: brak weryfikacji, który jest ustawieniem domyślnym. Aby zrezygnować z weryfikowania certyfikatów, należy „wykomentować" wiersz `verify` w pliku *stunnel.conf* albo całkowicie go usunąć.

Ponieważ w SSL obowiązuje równorzędny (ang. *peer-to-peer*) model uwierzytelniania (to znaczy z perspektywy SSL nie ma „certyfikatów klienta" i „certyfikatów serwera"; są po prostu „certyfikaty"), proces Stunnel może wymagać weryfikacji certyfikatu bez względu na to, **czy** działa w trybie serwera, **czy** klienta. Innymi słowy, nie tylko serwery Stunnel mogą wymagać, aby klienty przedstawiały ważne certyfikaty; klienty również mogą sprawdzać certyfikaty serwerów!

W praktyce jest to prawdopodobnie najbardziej użyteczne w scenariuszach HTTPS (na przykład w handlu elektronicznym; jeśli ktoś ma przesłać numer swojej karty kredytowej do sklepu internetowego, to chce się upewnić, że nie ma do czynienia z oszustem). Jeśli chodzi o użycie programu Stunnel, trudno znaleźć równie istotny powód do uwierzytelniania serwerów przez klienty. Tak czy owak, przetestowałem to i odkryłem, że nie różni się niczym od odwrotnej sytuacji, więc oba następne przykłady będą dotyczyć uwierzytelniania klientów przez serwery.

Przykład uwierzytelniania X.509

Wróćmy do przykładu przekazywania usługi *rsync* między hostami *skillet* i *elfiero*. Przypominam, że *skillet* jest klientem i w pliku */etc/services* ma wpis odwzorowujący nazwę usługi *ssyncd* na port TCP 273. To samo dotyczy serwera *elfeiero*. Oba hosty mają również w pliku */etc/hosts.allow* wiersz, który daje wszystkim komputerom dostęp do usługi *ssync*. Wreszcie w hoście *elfiero* działa demon *rsync* uruchomiony poleceniem `rsync --daemon --address=127.0.0.1`.

W tym przykładzie skonfigurujemy host *elfiero* tak, aby przyjmował tylko połączenia od klientów, które mają certyfikat podpisany przez nasz urząd CA. Host *skillet* będzie zatem potrzebował własnego certyfikatu; należy go utworzyć zgodnie z procedurą opisaną w podpunkcie „Tworzenie certyfikatów podpisanych przez CA". Uzyskanym w ten sposób plikom nadamy nazwy *skillet_cert.pem* (połączony certyfikat i klucz hosta *skillet*) oraz *skillet_pubcert.pem* (podpisany certyfikat hosta *skillet*). Będziemy też potrzebować kopii certyfikatu CA, *cacert.pem*.

W hoście *elfiero* musi znajdować się kopia certyfikatu CA (*cacert.pem*). W hoście *skillet* musi być plik *skillet_cert.pem*, ale certyfikat CA nie jest potrzebny, chyba że później nakażemy *skillet* weryfikować certyfikat serwera *elfiero*.

Certyfikaty można przechowywać w dowolnym miejscu, z tym zastrzeżeniem, że muszą mieć prawa dostępu 400 i należeć do użytkownika oraz grupy `root` (lub `wheel`). Zatem, dla uproszczenia, w obu systemach umieścimy je w katalogu */etc/stunnel*. Można połączyć wszystkie certyfikaty urzędu CA i klientów w jeden duży plik za pomocą polecenia `cat`, a następnie wskazać go przy użyciu parametru `CAfile` w pliku *stunnel.conf* (taką metodą posłużymy się w tym przykładzie). Możemy też przechowywać certyfikaty jako oddzielne pliki w katalogu określonym przez parametr `CApath`.

Ci, którzy zdecydują się na drugie rozwiązanie (użycie parametru `CApath`), muszą wiedzieć, że wartość `CApath` jest interpretowana względem ścieżki `chroot` programu Stunnel (chyba że w pliku *stunnel.conf* nie podano opcji `chroot`), a więc inaczej niż w przypadku parametru `CAfile`, który określa ścieżkę bezwzględną. Ponadto Stunnel oczekuje, że wszystkie pliki certyfikatów w katalogu `CApath` będą miały nazwy w postaci wartości skrótu (hash). Ponieważ nikt nie lubi nadawać plikom takich nazw, zwykle oblicza się skrót pliku, a następnie tworzy dowiązanie symboliczne od wartości skrótu do rzeczywistej nazwy pliku.

Pakiet OpenSSL zawiera bardzo przydatne polecenie, `c_rehash`, które robi to automatycznie. Przyjmuje ono argument w postaci nazwy katalogu i tworzy odpowiednie dowiązania symboliczne do wszystkich znajdujących się w nim certyfikatów — na przykład `c_rehash /etc/stunnel`.

Po umieszczeniu certyfikatów CA (i certyfikatów klienta, jeśli używany jest 3. poziom weryfikacji) w serwerze oraz certyfikatu klienta w kliencie można ponownie skonfigurować i uruchomić demony Stunnel.

Na listingu 5.9 przedstawiono opcje globalne i definicje usług z pliku *stunnel.conf* w hoście *elfiero*, które nakazują programowi Stunnel nasłuchiwać na porcie *ssyncd* (TCP 273), przekazywać pakiety do lokalnego portu *rsync* (TCP 973), domagać się certyfikatów z zaufanymi podpisami i używać pliku */etc/stunnel/cacert.pem* do weryfikowania certyfikatów klientów.

Listing 5.9. Plik stunnel.conf w serwerze sprawdzającym certyfikaty klientów

```
cert = /etc/stunnel/elfiero_cert
client = no
verify = 2
CAfile = /etc/stunnel/cacert.pem
```

Jeśli używany jest dowolny poziom weryfikacji certyfikatów, należy **koniecznie określić miejsce ich przechowywania** za pomocą parametru `CApath` (w celu określenia katalogu) albo `CAfile` (w celu określenia pliku zawierającego wiele certyfikatów urzędu CA i klientów). Znakomita większość problemów z uwierzytelnianiem w programie Stunnel, z jakimi miałem do czynienia, była spowodowana tym, że program nie mógł odszukać certyfikatów hosta albo urzędu CA.

W systemie klienckim *skillet* wystarczy dodać jedną globalną opcję: `cert` (zobacz listing 5.10).

Listing 5.10. Uruchamianie programu Stunnel w trybie klienta, z certyfikatem klienta

```
cert = /etc/stunnel/skillet_cert
```

Polecenie, za pomocą którego należy uruchomić program *rsync* w hoście *skillet*, jest dokładnie takie samo jak na listingu 5.5. W tym przypadku transakcja jest jednak lepiej chroniona, a dodatkowe zabezpieczenia są **całkowicie niewidoczne dla użytkownika**.

Aby zwiększyć poziom weryfikacji certyfikatów z 2. na 3. (to znaczy sprawdzać nie tylko ważność podpisów, ale także lokalne kopie certyfikatów), należy wykonać dwie dodatkowe czynności:

1. Dołączyć kopię podpisanego certyfikatu hosta *skillet* (*skillet_pubcert.pem*, wersję bez klucza hosta *skillet*) na końcu pliku */etc/stunnel/cacert.pem*, w hoście *elfiero*.
2. W pliku *stunnel.conf*, w hoście *elfiero* zmienić wartość parametru `verify` z `2` na `3`.

Jeśli ktoś ma ochotę skopiować do hosta *elfiero* również plik *skillet_cert.pem* (z certyfikatem i kluczem), powinien oprzeć się tej pokusie; niepotrzebne kopiowanie kluczy prywatnych jest bardzo złym nawykiem.

Używanie programu Stunnel w serwerze i innych aplikacji SSL w klientach

Program Stunnel nie jest jedyną aplikacją SSL, która może nawiązywać połączenia z demonem Stunnel. Można na przykład uruchomić Stunnel w serwerze POP3, który nasłuchuje na standardowym porcie *pop3s* (TCP 995) i przekazuje pakiety do lokalnego demona poczty POP3. Pozwala to łączyć się z serwerem za pomocą popularnych klientów POP3 obsługujących SSL, takich jak Outlook Express i Eudora, w systemach klienckich, w których nie da się uruchomić programu Stunnel.

W rzeczywistości jest to **prostsze** niż przykłady podane w niniejszym rozdziale; konfiguracja serwera jest taka sama, a po stronie klienta wystarczy włączyć obsługę SSL. Więcej wskazówek można znaleźć w dokumencie Stunnel FAQ (*http://www.stunnel.org/faq/*).

Inne narzędzia do tunelowania

Zaszyfrowane tunele można tworzyć nie tylko za pomocą programu Stunnel. Wśród innych narzędzi warto wymienić napisany przez Ricka Kasegumę, przypominający Stunnel program SSLwrap (nie był on aktualizowany od 2000 roku) oraz program SSH, który był tematem poprzedniego rozdziału. Strona główna programu SSLwrap znajduje się pod adresem *http://www.quiltaholic.com/rickk/sslwrap*, a więcej informacji o tunelowaniu podano w rozdziale 4.

Zasoby

http://www.openssl.org
: Oficjalna strona główna projektu OpenSSL.

http://ospkibook.sourceforge.net/
: Dokument Open Source PKI Book.

http://www.openca.org/openca/
: Strona główna projektu OpenCA.

Viega, Joh, Matt Messier i Pravir Chandra *Network Security With OpenSSL*, Sebastopol, CA: O'Reilly, 2002.
: Kompletny przewodnik poświęcony korzystaniu z OpenSSL.