Stunnel

Paweł Krawczyk <kravietz@aba.krakow.pl>

Gdzieś około stycznia 1998 na grupie dyskusjnej *pl.comp.security* spotkało się trzech administratorów-programistów, którzy w tym okresie natknęli się na identyczny problem – klienci dostrzegli niebezpieczeństwa ściągania poczty przez Internet otwartym połączeniem i coś z tym trzeba było zrobić. W ten sposób, w ciągu kilku tygodni, narodził się *stunnel*, który przez następne kilka lat stał się standardowym narzędziem do tego celu.

Jakie to zagrożenia? Protokół POP3 po którym najczęściej ściągamy pocztę jest najprostszy z możliwych – najpierw następuje uwierzytelnienie użytkownika jego nazwą i hasłem, a następnie kolejno ściągane są wiadomości, wszystko otwartym tekstem. Podsłuchujący na dowolnym odcinku połączenia uzyskuje zatem dostęp do loginu i hasła użytkownika, a zaraz potem – do treści jego korespondencji.
Podstawowym zastosowaniem Stunnela jest nawiązywanie szyfrowanych połączeń do popularnych usług, które nie posiadają wbudowanej obsługi protokołu SSL. Umożliwia to stosunkowo łatwe budowanie bezpiecznych serwisów w oparciu o dotychczas wykorzystywane i sprawdzone serwery, bez konieczności dokonywania jakichkolwiek przeróbek w ich kodzie.
Główne zalety Stunnela to wszechstronność (mało jest rzeczy, których nie da się przy jego pomocy osiągnąć), otwartość i darmowość (jest dostępny z kodem źródłowym) oraz przenaszalność – można go skompilować praktycznie na każdym używanym obecnie systemie operacyjnym, dla użytkowników Windows jest także dostępny w postaci gotowych do użycia binariów. W Windows NT Stunnel może także działać jako usługa systemowa.

Co potrafi Stunnel?

Stunnel można najkrócej opisać jako dwukierunkowe *proxy* dla protokołu SSL, jego funkcje to w szczególności:

- akceptowanie szyfrowanych połączeń SSL na ustalonym porcie i przekazywanie ich w postaci otwartego połączenia na inny port na tym samym serwerze
- to samo, z przekazaniem połączenia na wskazany port zdalnego serwera
- to samo, z uruchomieniem programu w obrębie tej samej maszyny i przekazanie mu komunikacji z klientem przez standardowe

wejście/wyjście
- akceptowanie otwartego połączenia na ustalonym porcie i przekazanie go wewnątrz tunelu SSL na inny port zdalnego serwera (*client proxy*)

Dodatkowe funkcje to:

- przepisywanie adresu klienta dla usługi lokalnej (*transparent proxy*)
- weryfikacja nazwy klienta za pomocą protokołu IDENT
- obsługa rozszerzeń popularnych protokołów o wsparcie dla SSL
- uwierzytelnienie na poziomie SSL na podstawie certyfikatów klienta
- kontrola dostępu do portów przy pomocy *tcp_wrappers*
- działanie z prawami nieuprzywilejowanego użytkownika

Poniżej opisujemy szczegółowo poszczególne tryby działania Stunnela, wraz z przykładami praktycznymi. Proszę zwrócić uwagę, że aby nie zaciemniać opisu we wszystkich przypadkach pominięte zostały opcje dotyczące certyfikatów X.509, którym z kolei poświęcona jest w całości osobna część tego artykułu.

Scenariusz #1

Posiadamy działający serwer internetowy z działającą usługą POP3, do którego musimy dodać obsługę SSL (system operacyjny nie ma w tym wypadku znaczenia). Serwer POP3 działa tradycyjnie na porcie 110, dla POP3 po SSL wybieramy na podstawie tabeli 1. port 995. Chcemy, by klient łączący się ze swojego programu pocztowego (Netscape, Outlook) po zaznaczeniu odpowiedniej opcji mógł ściągać pocztę jak dotychczas, ale po szyfrowanym połączeniu.
W najprostszym wariancie wystarczy uruchomienie następującego polecenia:

```
stunnel -d 995 -r 110
```

Opcje te przekładają się następująco:
- Stunnel otwiera port 995, na którym przyjmuje szyfrowane połączenia od klientów
- wszystkie komendy otrzymywane od klienta wewnątrz tunelu SSL są przekazywane bezpośrednio na port 110 lokalnego serwera
- odpowiedzi od serwera POP3 działającego na tym porcie są zwracane z powrotem do klienta przez tunel SSL

Scenariusz #2

Powyższą architekturę możemy zmodyfikować o dodatkowe wymaganie – Stunnel, uruchamiany na routerze brzegowym musi udostępniać na zewnątrz (po SSL) serwer POP3, który znajduje się jednak na jednym z wewnętrznych serwerów, dostępnych pod adresem 10.1.1.30. Zmiana linii komend w tym wypadku jest bardzo prosta – Stunnel powinien być uruchamiany na routerze w taki sposób:

```
stunnel -d 995 -r 10.1.1.30:110
```

Jak widać, do opcji `-r` określającej cel przekierowania dodano adres IP zdalnego serwera. Podanie, jak poprzednio, samej liczby określającej numer portu zostanie zinterpretowane jako przekierowanie na adres 127.0.0.1, czyli lokalny serwer.

Scenariusz #3

Oba powyższe rozwiązania wymagają już działającego serwera POP3, co może być niepotrzebnym otwieraniem portu oraz marnowaniem zasobów. W takich przypadkach bardziej opłacalne może okazać się uruchamianie programu serwera bezpośrednio przez Stunnel. W tym wypadku polecenia klienta będą przekazywane na i z standardowego wejścia/wyjścia serwera. Dotyczy to jednak tylko demonów uniksowych, które są przystosowane do takiego trybu pracy, przydatnego do uruchamiania z serwera nadrzędnego *inetd.* Na przykład:

```
stunnel -d 995 -L /usr/sbin/in.pop3d – pop3d
```

W tym przypadku Stunnel będzie oczekiwał na nadchodzące połączenia na porcie 995 – otrzymanie go spowoduje uruchomienie procesu serwera POP3, komunikującego się dalej z klientem.

Scenariusz #4

Stunnel może także działać jako *proxy* klienckie, czyli akceptować połączenia przychodzące od klientów otwartym tekstem i przekazywać je po SSL do wybranego serwera zdalnego. Tę funkcję można wykorzystać na wiele sposobów, w szczególności do dodania obsługi

SSL do oprogramowania klienckiego, które jest go natywnie pozbawione. Na przykład:

- ściągania poczty po SSL starym programem pocztowym
- łączenia się po SSL z programów, które nie posiadają SSL gdy nie mamy możliwości ich modyfikacji (np. w komercyjnych systemach tworzenia kopii zapasowych, oprogramowaniu bankowym)

Takie rozwiązanie rozważaliśmy podczas zabezpieczania aplikacji bankowej działającej pod systemem Windows, korzystającej do łączenia z serwerem zwykłego Telneta. Serwer działający pod systemem AIX posiadał natywnie obsługę SSL, której brakowało jednak po stronie stacji roboczych (rysunek 1). Jedna z propozycji przewidywała uruchomienie po stronie klienta Stunnela z następującymi parametrami:

```
stunnel -c -d 23 -r serwer:992
```

Zastosowana tutaj opcja `-c` odwraca strony tunelowania – szyfrowany jest ruch od lokalnego portu 23 do portu 992 (*Telnet/*SSL) na serwerze. W ten sposób, po uruchomieniu odpowiedniej usługi na serwerze (Stunnela w typowej konfiguracji lub systemowego serwera SSL) i ustawieniu w kliencie adresu serwera na *localhost* osiągamy tunelowanie po SSL bez konieczności jakichkolwiek modyfikacji w oprogramowaniu klienckim (rysunek 2).

Rozszerzenia protokołów

Wszystkie opisane powyżej sytuacje opisują tunelowanie protokołów aplikacyjnych w SSL bez informowania samych aplikacji o fakcie tunelowania (jest ono transparentne). Część protokołów jednak posiada wprowadzone w ostatnich latach rozszerzenia, które pozwalają na włączenie szyfrowania SSL podczas negocjacji parametrów połączenia na standardowym porcie. Dotyczy to w szczególności protokołów:

- POP3 posiada komendę STLS, które umożliwia włączenie szyfrowania SSL przed uwierzytelnieniem użytkownika komendami USER i PASS oraz przed ściąganiem listów poleceniem RETR
- SMTP posiada komendę STARTTLS, która umożliwia włączenie szyfrowania przed całą dalszą wymianą poczty (MAIL FROM, RCPT TO, DATA)

- NNTP również posiada komendę STARTTLS

Zaletą tych rozszerzeń jest to, że nie trzeba dodatkowego portu na serwer SSL, bo negocjacja SSL następuje w ramach normalnej wymiany komend. Wadą jest przede wszystkim fakt, że jest to rozszerzenie, co powoduje kłopoty ze wsparciem po stronie oprogramowania klienckiego, serwerowego oraz z kompatybilnością.
Stunnel z opcją `-n` umożliwia obsłużenie każdego z tych rozszerzeń (z argumentem odpowiednio `pop3`, `smtp`, `nntp`). Na przykład, jeśli zdalny serwer pocztowy (SMTP) obsługuje rozszerzenie STARTTLS i chcemy wysłać do niego pocztę po SSL to możemy to zrobić następująco:

```
stunnel -c -d 25 -r serwer:25 -n smtp
```

Połączenie na lokalny port 25 spowoduje nawiązanie przez Stunnel połączenia ze zdalnym serwerem, przełączenie na SSL przy pomocy rozszerzenia STARTTLS i udostępnienie nam oryginalnej winietki, ale już w sesji szyfrowanej (Czytelnikom, którzy chcieliby to przetestować w praktyce polecam serwer *mx.onet.pl*, który działa w oparciu o ZMailer i obsługuje STARTTLS).

Rozszerzenia uniksowe

Stunnel działaj pod systemami uniksowymi posiada kilka opcji, które usprawniają jego działanie. Są to:

- opcja `-T`, włączająca tryb transparentnego *proxy* – powoduje to że program będzie przepisywał adres klienta podczas wykonywania połączeń na zdalne serwisy (w przeciwnym wypadku widzą one połączenie przychodzące z adresu, pod którym działa Stunnel); opcja ta jest obecnie dostępna tylko dla Linuksa
- opcje `-u` i `-g`, wymuszające przejście Stunnela w pracę z prawami użytkownika i grupy pozbawionych przywilejów administratora (podawanych jako argumenty tych opcji)

Certyfikaty

Najmniej przyjemnym aspektem administrowania serwerami SSL jest zarządzanie certyfikatami – głównie dlatego, że do stosunkowo prostej konfiguracji samych serwerów dochodzi konieczność korzystania ze skomplikowanej infrastruktury klucza publicznego X.509. W zamian

jednak dostajemy wszystkie profity, które się z tym wiążą – na przykład zabezpieczenie przed atakami typu *man-in-the-middle*, czego nie daje nam na przykład protokół SSH.
Każdy serwer udostępniany za pośrednictwem Stunnela musi posiadać swój certyfikat X.509 (będący faktycznie jego kluczem publicznym) oraz klucz prywatny. Do wygenerowania i zarządzania certyfikatami można wykorzystać dostępne na rynku systemy (np. MS Certificate Server), można to również zrobić ręcznie, przy pomocy narzędzi z projektu OpenSSL (www.openssl.org). Cała procedura składa się z następujących kroków:

1. wygenerowanie klucza prywatnego CA (*Certifying Authority*, czyli nasz lokalny mini-urząd certyfikujący)
2. wygenerowanie CSR (*Certificate Signing Request*) dla CA
3. podpisanie CSR kluczem CA, co da w wyniku certyfikat CA (tzw. *self-signed*)
4. wygenerowanie klucza prywatnego serwera
5. wygenerowanie CSR serwera
6. podpisanie CSR serwera kluczem CA – wynikiem jest gotowy do użycia certyfikat serwera

Mając gotowy certyfikat CA możemy powtarzać kroki 4-6 dowolną ilość razy dla różnych serwerów, co jest dość istotne bo każdy certyfikat jest ściśle związany z nazwą domenową danego serwera. O ile więc możemy zastosować ten sam certyfikat dla różnych usług dostępnych po SSL na maszynie *secure1.test.com*, o tyle dla serwera *secure2.test.com* powinniśmy wygenerować osobny certyfikat.

Lokalny urząd certyfikujący

1. generujemy klucz RSA o długości 2048 bitów, który zapisujemy do pliku `ca.key` (rysunek 3)
2. na podstawie klucza prywatnego generujemy CSR, czyli klucz publiczny plus dane posiadacza certyfikatu (rys. 4)
3. podpisujemy CSR kluczem CA (opcja `signkey` oznacza podpisanie typu *self-sign*), wystawiając certyfikat `ca.cert` na 9999 dni (rys. 5)

W ten sposób uzyskujemy klucz prywatny oraz certyfikat urzędu certyfikującego, które możemy wykorzystywać do podpisywania certyfikatów dla poszczególnych usług. Klucz CA powinien być zabezpieczony hasłem.

Certyfikat serwera

1. generujemy klucz prywatny serwera o długości RSA, nie zabezpieczony hasłem, zapisując go do pliku `server.key` (rys. 6)
2. analogicznie jak poprzednio, generujemy CSR serwera – istotne jest podanie jego nazwy domenowej w polu *Common Name*(rys. 7)
3. podpisujemy nowy CSR kluczem oraz certyfikatem urzędu certyfikującego, co stanowi poświadczenie jego autentyczności (rys. 8)

Jako rezultat istotne są pliki `server.key` oraz `server.cert`, czyli klucz prywatny oraz certyfikat serwera. Zauważmy, że klucz prywatny serwera nie może być (w najprostszym wypadku) zabezpieczony hasłem, ponieważ będzie on wczytywany automatycznie przez Stunnel, który ma być w założeniu uruchamiany bez interwencji użytkownika.
Plik `ca.srl` zawiera po prostu kolejny numer certyfikatu wydanego przez lokalny „urząd”, niemniej jest konieczny do wystawiania kolejnych certyfikatów.

Korzystanie z certyfikatu

Wszystkie wyżej wymieniony przykłady uruchamiania Stunnela pomijały opcję, wskazującą plik zawierający certyfikat oraz klucz prywatny danej instancji serwera. Plik ten powinien stanowić połączenie obu plików wynikowych, które można wygenerować na przykład tak:

```
cat server.key server.cert >server.pem
```

Aby wskazać Stunnelowi ten plik należy do jego linii poleceń dodać opcję `-p` w następujący sposób:

```
stunnel -p server.pem ...
```

Uwagi końcowe

Oficjalna dokumentacja programu podaje przykład stworzenia przy jego pomocy sieci VPN, tunelując przez SSL protokół PPP (a w nim IP i wyższe). Innym nasuwającym się pomysłem jest wykorzystanie Stunnela postawionego na maszynie przed właściwym serwerem do odciążenia tego ostatniego z czasochłonnych operacji

kryptograficznych.
Jak widać, obszar zastosowań narzędzia tak prostego, ale tak funkcjonalnego jak Stunnel jest ograniczony tylko potrzebami jego użytkowników, którzy zresztą aktywnie wspierają jego rozwój w ramach listy dyskusyjnej *stunnel-users.* W chwili obecnej projekt osiągnął stadium funkcjonalności oraz stabilności w pełni kwalifikujące go do zastosowań produkcyjnych, do czego go z czystym sumieniem polecam, również jako wieloletni użytkownik.

Oficjalna strona Stunnela: stunnel.mirt.net