Uniwersytet Mikołaja Kopernika w Toruniu
Wydział Matematyki i Informatyki

Kacper Czajkowski
Mikołaj Suchodolski

OpenSSH

Praca przygotowana w ramach konwersatorium z przedmiotu
“Administrowanie usługami sieciowymi”

Toruń, listopad 2020

Spis treści

Wprowadzenie

OpenSSH to zestaw programów pozwalających na zaszyfrowaną komunikację sieciową. Korzysta on z protokołu SSH (Secure shell), który jest następcą protokołu telnet. Opiera się na ostatnią wolną wersję programu Secure Shell wydanego przez Tatu Ylönen przed staniem się wersją komercyjną. Projekt został zapoczątkowany przez programistę oraz założyciela projektu OpenBSD, Theo de Raadt, który pochodzi z Pretorii w Afryce Południowej.

Zgodnie z definicją protokołu SSH, pakiet OpenSSH udostępnia zestaw narzędzi pozwalający na połączenie z pulpitem zdalnym. Wykorzystywany jest do zdalnego łączenia się z maszynami oraz do zarządzania nimi.

Początki OpenSSH

W roku 1999, programiści projektu OpenBSD otrzymali zadanie utworzenia darmowej wersji SSH. Mieli za zadanie uporządkować kod z nim związany oraz sprawy licencyjne. OpenSSH bazuje na ostatniej darmowej wersji ssh 1.2.12 autorstwa Tatu Ylönela. Korzystała ona z biblioteki matematycznej libgmp udostępnionej na zwykłej licencji GNU Public License.

Ssh tworzone przez Tatu od wersji 1.2.12 rozpoczęło restrykcyjne licencjonowanie mimo darmowego libgmp, które było podstawą działania oprogramowania. Wraz z czasem zabroniono tworzenia wersji pod systemy Windows lub DOS oraz nakazywano, że w przypadku wykorzystywania komercyjnego, należy wykupić wersję płatną.

W roku 1999, Björn Grönvall rozpoczął pracę nad poprawianiem ostatniej darmowej wersji ssh Tatu. Jego wersja nazywała się OSSH i obsługiwała tylko protoków SSH 1.3.

O OSSH członkowie projektu OpenBSD dowiedzieli się niedługo przed wydaniem 2.6. Doszli do wniosku, że chcą włączyć obsługę protokołu ssh do ich projektu. Dlatego zdecydowali się rozpocząć własne prace nad tym projektem. Oddzielili ostatnią wersję OSSH i rozpoczęli jej rozwój, co poskutkowało dodaniem ich wersji ssh o nazwie “OpenSSH” do pakietu OpenBSD.

Ważne daty:

- 1 grudnia 1999r - wydanie OpenSSH w wersji 1.2.2 wraz z OpenBSD
- 15 czerwca 2000r - wydanie OpenSSH 2.0 raz z OpenBSD 2.7. Ta wersja wspierała protokół SSH 2.0
- listopad 2000r - rozpoczęcie wspierania protokołu SFTP dla wersji serwerowej

Idea

OpenSSH pozwala na łączenie się z maszynami z dowolnego miejsca na świecie, co daje wielkie możliwości. Dzięki temu rozwiązaniu praca nad serwerami nie wymaga już obecności administratora bezpośrednio przy maszynie. W celu dokonania najmniejszej zmiany wystarczy, że zostanie uruchomione połączenie przy użyciu SSH.

Zestaw programów znalazł duże zastosowanie w branży administratorskiej. Na przykład, firmy posiadające serwery rozsiane na terenie całego kraju przestały potrzebować w każdym z tych miejsc całego zespołu administratorów. Połączenie przy użyciu SSH pozwala, by jeden zespół zajmował się serwerami, między którymi dzielą ich dziesiątki tysięcy kilometrów.

Dlaczego OpenSSH

OpenSSH jest odpowiednim zestawem narzędzi do użytkowania dlatego, że m.in. jest darmowe, dzięki czemu staje się ono dostępne dla każdego problemu. Powoduje to także, że łatwo znaleźć informacje dotyczące konfigurowania serwerów.

SSH rozwiązuje problemy z łącznością, unikając problemów bezpieczeństwa związanych z eksponowaniem wirtualnej maszyny w chmurze. Tunel SSH może zapewnić bezpieczną ścieżkę przez Internet, poprzez zaporę ogniową do maszyny wirtualnej. Korzystając z SSH, możemy swobodnie poruszać się po strukturze plików maszyny, z którą się łączymy.

Oprogramowanie OpenSSH w wielu przypadkach jest wystarczające, lecz w przypadku, gdy wymagana jest niezawodność programu, lepszym pomysłem byłoby zastosowanie wersji płatnych. Powodem takiej decyzji może być oferowane wsparcie, które zawarte jest w cenie oprogramowania, w przeciwieństwie do wersji darmowych. Do programów realizujących podaną usługę należą m.in: Apache MINA, Bitvise oraz CopSSH.

Skład pakietu OpenSSH

- *ssh* - klient SSH (Secure Shell Protocol) - służy umożliwieniu zalogowania się na maszynie zdalnej i wykonywania na niej poleceń. Wykorzystuje zaszyfrowane połączenie by zapewnić bezpieczeństwo przy połączeniu przez niezaufaną sieć (np.: internet).

- *scp* - program do bezpiecznego kopiowania plików pomiędzy hostami w sieci. Bezpieczny oznacza że transfer danych jest szyfrowany i wymaga przejścia przez proces autentykacji bazując na programie *ssh*.

- *sftp* - podobnie do scp służy do bezpiecznego transferu plików za pomocą *ssh*. Ponieważ jednak bazuje na programie *ftp* (który jest interfejsem dla standardu FTP - File Transfer Protocol) posiada on większą funkcjonalność np.: tryb interaktywny, przeglądanie plików czy tworzenie nowych katalogów na maszynie zdalnej, możliwość automatyzacji (opcja *-b*).

- *ssh-add* - pozwala na dodawanie nowych kluczy do użycia z *ssh-agent*.

- ssh-keygen - generuje, zarządza i zamienia klucze do użycia z *ssh*. Domyślnie generuje prywatny klucz dla użytkownika.

- *ssh-keyscan* - służy do zbierania publicznych kluczy z podanej listy hostów.

- *ssh-keysign* - służy do “odebrania” klucza lokalnego użytkownika i wygenerowania podpisu podczas autentykacji. W założeniu nie ma być wywoływany przez użytkownika.

- *sshd* - serwer SSH - daemon oczekujący na połączenia od klientów. Umożliwia zalogowanie się za pomocą ssh na maszynie na której jest uruchomiony.

- *sftp-server* - podsystem odpowiedzialny za obsługę protokołu SFTP. Jest uruchamiany automatycznie razem z *sshd*.

- *ssh-agent* - program trzymający w swojej pamięci (dla bezpieczeństwa) prywatne klucze użytkownika. Pozwala przez to na logowanie bez użycia haseł przy połączeniu ze zdalną maszyną.

Pliki konfiguracyjne

known_hosts i *authorized_keys*

known_hosts - zawiera listę publicznych kluczy zdalnych hostów. Dzięki temu administrator systemu może przygotować listę znanych i zaufanych hostów przez co można zapobiec próbom podszycia się lub pomyłek ponieważ przykładowo wyświetlane jest ostrzeżenie przy połączeniu z hostem którego klucz nie jest na liście.

authorized_keys - zawiera klucze publiczne klientów którzy mogą połączyć się z serwerem. Umożliwia to administratorowi kontrolowanie dostępu do maszyny zdalnej, a użytkownikom również logowanie bez haseł.

W uproszczeniu można powiedzieć że *known_hosts* służy do weryfikacji połączeń wychodzących a *authorized_keys* przychodzących.

Pliki te znajdują się w katalogach *~/.ssh* użytkowników.

Serwer

Innych zmian w konfiguracji dokonujemy przez modyfikację pliku z uprawnieniami administratora *sshd_config* w katalogu */usr/local/etc* (domyślnie). Możliwości konfiguracji opisane są w podręczniku wywoływanym poleceniem:

man sshd_config

Aby zaszły zmiany w konfiguracji należy zrestartować serwer poleceniem:

systemctl restart sshd

Klient

W przypadku klienta konfiguracja może być dokonana w 3 miejscach:

- Argumenty dla opcji *-o* przy wywoływaniu klienta (np.: *ssh localhost -o "Port 2222"*)
- Plik konfiguracji użytkownika *config* umieszczony w katalogu *~/.ssh* - musi posiadać prawa zapisu i odczytu tylko dla użytkownika
- Systemowy plik konfiguracji *ssh_config* w naszym wypadku w katalogu */usr/local/etc*

Przy czym są wylistowane w kolejności precedensu tzn. argumenty linii poleceń biorą precedens nad plikiem konfiguracji użytkownika itd..

Analogicznie do serwera, pełne możliwości konfiguracji klienta opisane są w podręczniku *ssh_config*.

Wybrane możliwości

Tworzenie

Parę kluczy (prywatny i publiczny) generujemy przez użycie programu *ssh-keygen*. Rodzaj klucza do wygenerowania podajemy jako argument dla opcji *-t* - z wyjątkiem klucza typu RSA który bez podania żadnych argumentów jest generowany domyślnie. Dla najważniejszych typów - RSA, DSA, ECDSA, ED25519 - wystarczy podać ich nazwy (wielkość liter nie ma znaczenia).

Żeby znaleźć obsługiwane rodzaje kluczy można użyć polecenia *ssh -Q HostKeyAlgorithms* lub przejrzeć podręcznik *ssh_config*.

Żeby określić długość (w bitach) wygenerowanego klucza używamy opcji *-b*.

Podczas procesu generacji użytkownik jest pytany o lokalizację i nazwę pliku dla pary kluczy - domyślnie będą one w postaci “id_<rodzaj klucza>” i przechowywane w katalogu *~/.ssh*, ponadto dla czterech podanych typów będą one domyślną “tożsamością” użytkownika używaną do autentykacji co jest określone przez domyślne argumenty dla *IdentityFile* w pliku konfiguracji *ssh_config*.

Dodatkowo pytany jest o podanie hasło - jest używane do zaszyfrowania klucza prywatnego - którego można nie podać - w takim wypadku hasło będzie przechowywane w “czystej” postaci. Hasło można też później zmienić używając polecenia *ssh-keygen -p*.

Po przejściu przez proces generacji w podanej lokalizacji pojawią się dwa pliki zawierające klucz prywatny i klucz publiczny który do podanej nazwy będzie miał dodany przyrostek *.pub* .

Krótkie porównanie rodzajów kluczy

<table>
<tr><td>Typ klucza</td><td>DSA*</td><td>RSA</td><td colspan="2">ECDSA</td><td>ED25519</td></tr>
<tr><td>Zalecana długość na 2019 - 2030 i dalej</td><td>3072</td><td>3072</td><td colspan="2">256</td><td>256</td></tr>
<tr><td>Wyzwanie dla złamania</td><td>Problem logarytmu dyskretnego</td><td>Rozkład dużych liczb złożonych na czynniki</td><td colspan="3">Arytmetyka krzywych eliptycznych</td></tr>
<tr><td rowspan="2">Uwagi</td><td rowspan="2">Potencjalna słabość w użyciu wartości losowych w przypadku słabego generatora liczb losowych</td><td rowspan="2">Najpopularniejszy i przez to najbardziej kompatybilny z różnymi klientami i serwerami ssh, zamiast używać większych długości zaleca się użycia kluczy bazujących na ECC**</td><td>Obawy przed potencjalnie “wbudowanymi” słabościami przez NSA, trudność w implementacji,</td><td>Często polecany zamiast ECDSA ze względu na podane uwagi</td></tr>
<tr><td colspan="2">Wykorzystanie ECC** pozwala na znacznie krótsze klucze przy tym samym poziomie bezpieczeństwa co inne rodzaje kluczy</td></tr>
</table>

* Używanie kluczy DSA nie jest zalecane przez OpenSSH. Dalej można generować klucze tego typu, ale tylko o długości 1024 - co jest podyktowane przestarzałą specyfikacją.

** Elliptic Curve Cryptography - kryptografia krzywych eliptycznych.

Podmiana i generacja kluczy dla serwera

Klucze serwera znajdują się w katalogu */usr/local/etc* i można je poznać po przyrostku *ssh_host* w nazwie. Żeby wymienić wybrany klucz możemy go usunąć i wygenerować w jego miejsce nowy lub go nadpisać. Należy pamiętać o uprawnieniach administratora. Przykładowa wymiana klucza *ssh_host_rsa_key*:

sudo ssh-keygen -f ssh_host_rsa_key -N ‘’

Opcja *-f* pozwala od razu podać nazwę klucza zamiast czekać na pytanie od programu, opcja *-N* działa analogicznie dla hasła. Należy potwierdzić nadpisanie klucza.

Teraz jeśli wymieniany klucz znajdował się w pliku *known_hosts* klienta będzie wyświetlony komunikat o zmienionym kluczu podczas próby połączenia z serwerem.

Tunele SSH

Dzięki tunelom SSH możemy przykładowo zabezpieczyć transmisję danych z aplikacji które normalnie można po prostu odczytać (nie są zaszyfrowane) albo poprzez odpowiednie dobranie portów przy tworzeniu tuneli obejść restrykcje sieci (w szczególności firewall). Poniżej podane są podstawowe przykładowe polecenia tworzące różne rodzaje tuneli i opis ich działania.

- *ssh -L* <port lokalny>:<adres maszyny zdalnej>:<port zdalny> <użytkownik>@<adres maszyny z serwerem ssh>

Tworzone jest gniazdo na maszynie lokalnej nasłuchujące na podanym porcie lokalnym i przychodząca na to gniazdo transmisja jest przekierowywana do podanego serwera ssh który następnie przekazuje transmisję na podany port zdalny na maszynie zdalnej.

- *ssh -R* <adres A>:<port A>:<adres B>:<port B> <użytkownik>@<adres maszyny z serwerem ssh>

Tym razem gniazdo zostanie utworzone na maszynie z serwerem SSH. Będzie ono nasłuchiwać na podanym adresie i porcie A (bez podania adresu A będzie użyty adres “loopback” - 127.0.0.1 / localhost) a jakakolwiek transmisja która przyjdzie na te gniazdo będzie przekierowana na adres i port B z punktu widzenia maszyny na której zostało wykonane polecenie. W szczególności to znaczy że gdy za adres B poda się localhost to będzie możliwe połączenie na porcie B na tej maszynie przez adres i port A - może to służyć jako punkt dostępu do maszyny wewnątrz prywatnej, chronionej sieci ale też jako potencjalne zagrożenie w przypadku gdy maszyna A jest słabo chroniona ponieważ teraz maszyna wewnątrz jest “odsłonięta”.

- *ssh -D* <port lokalny> <użytkownik>@<adres maszyny z serwerem ssh>

Ponownie tworzone jest gniazdo nasłuchujące na podanym porcie maszyny lokalnej i po otrzymaniu transmisji przekierowywuje ją używając protokołu aplikacji która wysyła dane na ten port. To znaczy że aplikacja musi wspierać ten typ przekierowywania, przykładowo ustawienia proxy w przeglądarce.

Idea Keyrings

“Pęk kluczy” jest usługą, która pozwala na bezpieczne przechowywanie loginów, haseł oraz kluczy na dysku komputera. Główna koncepcja polega na tym, że hasła są zaszyfrowane i zapisane na dysku, a cały “pęk” zabezpieczony hasłem. Powoduje to, że gdy główne hasło jest bezpieczne, pozostałe hasła także są.

Na przykład, gdy korzystamy z przeglądarki internetowej z odpowiednim jej rozszerzeniemi logujemy się do jakiegoś serwisu, przeglądarka może zaproponować zapisanie hasła. Będzie ona wtedy próbować skontaktować się z usługą znajdującą się na naszym komputerze, która to oferuje.

Keyrings chronią przed pasywnymi atakami. Oznacza to, że w sytuacji gdy nasz komputer zostanie skradziony, to hasła będące na dysku są niemożliwe do zdobycia, ponieważ będą one zabezpieczone. Niestety, nie bronią one przed atakami aktywnymi, czyli pochodzącymi z pulpitu użytkownika (z jego perspektywy).

Rozwiązanie zadania

Kompilacja i instalacja na systemie Fedora 31 Server

Domyślnie pakiet OpenSSH jest już zainstalowany więc jeśli nie korzystamy z ssh by pracować na maszynie to można go od razu odinstalować poleceniem:

Mozna, ale nie trzeba.MG

sudo dnf remove openssh

W przeciwnym wypadku należy najpierw zainstalować nową wersję i skonfigurować ją tak żeby serwer SSH uruchamiał się wraz ze startem maszyny.

Wstępne wymagania:

- gcc - kompilator języka C.

  sudo dnf install gcc

- make - narzędzie do kompilacji i budowania programów.

  sudo dnf install make

- zlib - biblioteka używana do obsługi skompresowanych plików (np.: tar -z).

  sudo dnf install zlib-devel

- OpenSSL - zestaw narzędzi dla protokołów TLS (Transport Layer Security) i SSL (Secure Sockets
  Layer), a także biblioteka dla zastosowań w kryptografii.

  sudo dnf install openssl-devel

Pobieramy OpenSSH na przykład przez polecenie:

wget https://cloudflare.cdn.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-8.4p1.tar.gz

Wypakowywanie:

tar -xzf openssh-8.4p1.tar.gz

Po przejściu do wypakowanego katalogu należy wykonać polecenie:

./configure

Skrypt ten sprawdza czy wszystkie wymagania potrzebne do instalacji i zbudowania programu są spełnione między innymi wykryje jeśli wcześniej nie zainstalowaliśmy pakietu *openssl-devel* czy kompilatora.

Można użyć opcji *--prefix=/usr/local* chociaż domyślnie bez tego instalacja też powinna odbyć się w tym katalogu.

Zakończenie procesu instalacji:

make

sudo make install

Można sprawdzić czy instalacja się powiodła przykładowo sprawdzając wersję poleceniem:

ssh -V

Skrypt startowy

Aby móc zarządzać serwerem SSH za pomocą *systemctl* i sprawić by uruchamiał się przy starcie systemu należy napisać “unit file” - plik reprezentujący zasoby lub usługi w systemie - umożliwiający zarządzanie właśnie przez *systemctl*. Należy przenieść się do katalogu */etc/systemd/system* a następnie z uprawnieniami administratora utworzyć plik *sshd.service*.

Zaczynamy od sekcji *[Unit]* w której zamieszczamy podstawowe informacje o naszej usłudze. Można na przykład tu podać parametr *Requires* dzięki któremu możemy określić że na przykład do działania tej usługi potrzebujemy połączenia z siecią.

[Unit]

Description=Serwer SSH

W sekcji *[Service]* podajemy parametry mówiące o tym jak kontrolować usługę. Przede wszystkim wskazujemy na lokalizację pliku wykonywalnego.

[Service]

ExecStart=/usr/local/sbin/sshd

Tutaj parametr *WantedBy* jest w pewnym sensie analogiczny do *Requires* z sekcji *[Unit]*. Używając go mówimy że przy starcie podanej usługi uruchamiamy też naszą. Podanie jako argumentu *multi-user.target* efektywnie umożliwia uruchamianie jej ze startem systemu, gdyż jest to specjalny “stan” który system osiąga przy starcie i pozwala na działanie i wykonywanie pewnych operacji.

[Install]

WantedBy=multi-user.target

Teraz można zarządzać serwerem SSH przy pomocy polecenia *systemctl* i żeby uruchamiać go przy starcie systemu wykonujemy polecenie: *systemctl enable sshd*

Uruchamianie serwera SSH: *systemctl start sshd*

Zatrzymanie serwera SSH: *systemctl stop sshd*

Uniemożliwienie logowania jako root

By uniemożliwić logowanie się użytkownikowi root w pliku konfiguracyjnym serwera *sshd_config* należy zmodyfikować istniejącą lub dodać linijkę:

PermitRootLogin no

Zmiana portu

Żeby serwer zaczął pracował na porcie 2222 należy zmodyfikować istniejącą lub dodać linijkę:

Port 2222

Teraz trzeba zmodyfikować zasady firewall-a i tutaj domyślnie powinna być dopuszczona usługa "ssh" (można to sprawdzić poleceniem *firewall-cmd --list-services*). Jeśli tak nie jest to można ją dodać poleceniem:

firewall-cmd --permanent --add-service=ssh

Teraz można zmodyfikować jej parametry - chcemy zmienić port 22 na 2222 i robimy to za pomocą poleceń:

firewall-cmd --permanent --service=ssh --remove-port=22/tcp

firewall-cmd --permanent --service=ssh --add-port=2222/tcp

Po czym należy wykonać polecenie *firewall-cmd --reload* by zmiany zaszły miejsce.

Teraz do zalogowania się za pomocą *ssh* na maszynie zdalnej należy użyć opcji *-p* podając odpowiedni numer portu.

Logowanie bez hasła z poziomu serwera ultra60

Należy wygenerować klucz nie podając hasła do zaszyfrowania klucza - gdy poda się hasło przy każdej próbie połączenia przez *ssh* dalej trzeba będzie podawać hasło tyle że do odszyfrowania klucza prywatnego a nie użytkownika na maszynie zdalnej. Następnie można wykorzystać program *ssh-copy-id* jeśli maszyna takowy posiada (tak jak ultra60) lub przekopiować swój publiczny klucz do pliku *authorized_keys* na maszynie zdalnej (np. używając *scp* a potem *cat*) - w razie problemów należy się upewnić że plik ten ma uprawnienia "644". Przykładowe polecenia:

ssh-keygen

ssh-copy-id -i .ssh/id_rsa.pub -p 2222 username@hostname

Gdzie za *username* podajemy swoją nazwę użytkownika na maszynie zdalnej a za *hostname* adres maszyny (np. ultra60).

Opcja *-i* pozwala na wybranie klucza, *-p* portu. Teraz można zalogować się bez hasła na maszynie zdalnej (o ile nie podaliśmy hasła do klucza prywatnego).

Bibliografia

zlib.net

openSSL.org

ssh.com

openssh.com

Linux man pages

docs.fedoraproject.org

firewalld.org

NIST Special Publication 800-57 Part 1 Revision 5 Recommendation for Key Management: Part 1 – General, https://doi.org/10.6028/NIST.SP.800-57pt1r5

wiki.archlinux.org

pl.wikipedia.org

en.wikipedia.org

medium.com

inmotionhosting.com

wiki.gnome.org