Systemy kryptograficznej ochrony komunikacji warstwy aplikacyjnej (ssh)

1. Wprowadzenie

Kryptograficzna ochrona komunikacji coraz częściej wymagana jest przy przesyłaniu różnego rodzaju danych. W szczególności tymi danymi może być interaktywne połączenie w powłoką zdalnego systemu operacyjnego. Systemy nie wykorzystujące kryptograficznej ochrony komunikacji cały czas istnieją i są używane w sieci Internet, jednak coraz częstsze przechwytywanie ruchu, jego podsłuchiwanie wymusza zaczęcie stosowania bezpiecznych metod transmisji. Kryptografia to jedyna możliwość ochrony przed przypadkowym podsłuchaniem. Oczywiście nie można mieć pewności, że podsłuchujący nie będzie w stanie odszyfrować naszej transmisji, jednak trzeba starać mu się to utrudnić.

Protokół SSH umożliwia bezpieczny dostęp do zdalnego konta. Protokół pozwala na zastosowanie bezpiecznego uwierzytelniania użytkownika i szyfrowanie transmisji. Korzysta z portu TCP o numerze 22. Ponadto możliwe jest też szyfrowane tunelowanie ruchu pomiędzy arbitralnymi portami TCP, co umożliwia tworzenie tuneli wirtualnych. Transmitowane dane mogą podlegać automatycznej kompresji.

2. Aplikacja SSH w systemie Linux/Unix

Program ssh zastępuje w działaniu program rlogin i rsh.

```
local> ssh -l username remotehost ls
```

Umożliwia jednakże osiągnięcie podwyższonego stopnia bezpieczeństwa przy dostępie do zdalnego konta. Cała transmisja jest bowiem szyfrowana (łącznie w procedurą logowania, a więc i ewentualnym przesłaniem hasła), a do weryfikacji tożsamości użytkownika również stosowane są mechanizmy kryptograficzne.

Metody uwierzytelniania użytkownika.

Program ssh podejmuje próby uwierzytelnienia użytkownika w następującej kolejności:

- uwierzytelnianie przez mechanizm zaufania do systemów zdalnych połączony z kryptograficznym uwierzytelnieniem obu systemów metodą klucza asymetrycznego wraz z ustaleniem symetrycznego klucza sesji (metodą Dieffiego-Hellmana);
- uwierzytelnienie użytkownika kryptograficzną metodą klucza asymetrycznego, typu *challenge-response* (algorytm RSA lub DSA)
- uwierzytelnianie klasyczne poprzez hasło systemowe użytkownika zdalnego (komunikacja jest jednak szyfrowana, więc hasło nie jest transmitowane tekstem jawnym).

Mechanizm zaufania może być identyczny, jak dla poleceń r∗ (bazujący na plikach /etc/hosts.equiv oraz ~/.rhosts). Jednak, z powodu wysokiego niebezpieczeństwa jego stosowania, protokół SSH stosuje swój własny system zaufania, wykorzystujący pliki, odpowiednio, /usr/local/etc/shosts.equiv oraz ~/.shosts. Mimo tego, stosowanie mechanizmu zaufania nie jest rekomendowane i niektóre implementacje protokołu SSH nie obsługują tego mechanizmu (w szczególności w standardzie SSH2 nie przewiduje się już tego mechanizmu).

Kryptograficzne uwierzytelnianie systemu zdalnego wykorzystuje zbiór kluczy publicznych zdalnych systemów znanych systemowi lokalnemu, przechowywanych w plikach /usr/local/etc/ssh_known_hosts oraz ~/.ssh/known_hosts. Mechanizm szyfrowania asymetrycznego i tajność kluczy prywatnych wystarcza do obrony systemu przed atakami typu DNS spoofing, IP spoofing czy routing spoofing.

Kryptograficzne uwierzytelnianie zdalnego użytkownika wykorzystuje zbiór kluczy publicznych zaufanych użytkowników, przechowywanych w pliku ~/.ssh/authorized_keys. Weryfikacja tożsamości metodą *challenge-response*, przy zachowaniu tajności kluczy prywatnych użytkowników, umożliwia bezpieczne uwierzytelnianie.

Ssh dokonuje uwierzytelnienia przy pomocy algorytmu RSA.

Stosowane algorytmy szyfracji

W aktualnej wersji SSH v.2 obsługiwane są następujące symetryczne algorytmy szyfracji komunikacji:

- 3DES (wybierany domyślnie),
- Arcfour – bardzo szybki algorytm, pochodny RC4,
- Blowfish i CAST128 – algorytmy 128-bitowe

a także algorytmy podpisu elektronicznego (do zapewnienia integralności komunikacji):

- HMAC-MD5 oraz HMAC-SHA1.

Zarządzanie kluczami kryptograficznymi

Do tworzenia pary kluczy kryptograficznych służy program ssh-keygen. Zapisuje on klucz prywatny użytkownika wygenerowany dla algorytmu DSA (lub RSA) w pliku ~/.ssh/id_dsa (odpowiednio ~/.ssh/id_rsa), a klucz publiczny w pliku ~/.ssh/id_dsa.pub (~/.ssh/id_rsa.pub). Dla dodatkowego zabezpieczenia, klucz prywatny może być zapisany w pliku w postaci zaszyfrowanej, domyślnie algorytmem 3DES. Posługiwanie się wówczas tym kluczem wymaga każdorazowo podania sekwencji traktowanej jako rodzaj hasła.

```
local> ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (~/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_dsa.
Your public key has been saved in ~/.ssh/id_dsa.pub.
The key fingerprint is:
5c:13:fc:48:75:19:9b:27:ec:5c:4f:d3:b1:a2:6c:da
user@host.domain
```

Program ten jest również wykorzystywany do generacji kluczy systemu, prywatnego oraz publicznego, odpowiednio w plikach: /usr/local/etc/ssh_hosts_key i /usr/local/etc/ssh_hosts_key.pub.

Tunele wirtualne warstwy aplikacji (TCP port forwarding)

Istnieje możliwość zbudowania tunelu wirtualnego pomiędzy dwoma komputerami (np. bramami typu BastionHost), przechodzącego przez sieć niezabezpieczoną (np. publiczną), Tunel ten może być następnie wykorzystywany do bezpiecznej komunikacji w warstwie aplikacyjnej pomiędzy innymi komputerami (np. klientem znajdującym się przed pierwszą bramą i serwerem za drugą bramą).

```
gate1> ssh -L localport:server:serviceport gate2
```

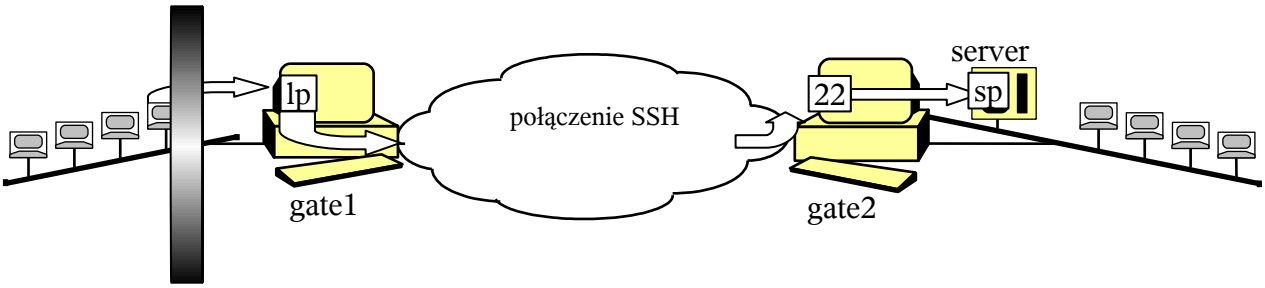
Takie polecenie utworzy tunel pomiędzy bramami `gate1` i `gate2`, który umożliwi komunikację poprzez port `localport` z usługą `serviceport` na komputerze `server` (`-L` = Local port forwarding). Komunikacja jest zaszyfrowana na odcinku `gate1-gate2`.

`localport` – port TCP lokalnego systemu (bramy `gate1`)

`server` – nazwa (domenowa) lub adres (IP) komputera, którego port udostępniamy poprzez tunel wirtualny (za bramą `gate1`)

`serviceport` – udostępniany zdalnie port TCP na serwerze `server`

Propagowanie połączeń w tunelu przedstawia schematycznie poniższy rysunek.



Identyczny efekt do poprzedniego ma kolejne polecenie:

```
gate2> ssh -R remoteport:server:serviceport gate1
```

(`-R` = Remote port forwarding).

`remoteport` – port TCP zdalnego systemu (bramy `gate1`)

`server` – nazwa (domenowa) lub adres (IP) komputera, którego port udostępniamy poprzez tunel wirtualny (przed bramą `gate2`)

`serviceport` – udostępniany zdalnie port TCP na serwerze `server`

3. Zadania

- Za pomocą narzędzi z pakietu OpenSSH nawiąż połączenie ze wskazanym serwerem (np. gumis).
- Wykonaj zdalnie w systemie tego serwera polecenie cat /etc/HOSTNAME.
- Dokonaj skopiowania lokalnego pliku do wybranego podkatalogu na koncie w systemie zdalnym.

- Wygeneruj parę kluczy kryptograficznych szyfrowania asymetrycznego dla swojego lokalnego konta (bez zabezpieczania hasłem pliku z kluczem prywatnym). Wykorzystaj klucze do uwierzytelniania użytkownika w systemie zdalnym.
- Zabezpiecz hasłem plik ze swoim kluczem prywatnym.
- Uruchom lokalne propagowanie połączeń (port forwarding) tylko dla lokalnych połączeń (*loopback*) z portu 8080 na port 80 serwera www (www.cs.put.poznan.pl) w tunelu wirtualnym pomiędzy swoim lokalnym systemem a serwerem unixlab.
- Uruchom lokalne propagowanie połączeń lokalnych i zdalnych (z sieci lokalnej) w konfiguracji jak powyżej.

4. Problemy do dyskusji

- Wady, zalety aplikacji SSH.
- Braki możliwości w aplikacji SSH.
- Czy SSH jest wszechstronną aplikacją?
- Czy dzięki SSH mogę być pewien, że łączę się z konkretnym serwerem zdalnym?
- Czy SSH wspiera możliwość uwierzytelniania przy użyciu certyfikatów X.509?

5. Bibliografia

[SSH] http://www.openssh.com/