

# SSH - Secure Shell

*Omówienie protokołu na przykładzie OpenSSH*



Paweł Pokrywka

# Co to jest SSH?

Secure Shell to protokół umożliwiający przede wszystkim zdalne wykonywanie komend. Powstał jako bezpieczna alternatywa telnetu.

Wykorzystuje szyfrowanie transmisji oraz kilka metod bezpiecznej autoryzacji.

# Co to jest OpenSSH?

- OpenSSH to wolna (otwarta) implementacja protokołu SSH udostępniana na licencji BSD.
- Rozwijana przez zespół doświadczonych programistów <sup>a</sup>
- W chwili obecnej (rok 2004) najpopularniejsza implementacja dla systemów uniksowych.
- Wykorzystywana w prawie każdej dystrybucji Linuksa, systemach BSD, a także w Solarisie, Cisco IOS i innych.

---

<sup>a</sup>...tworzących system OpenBSD

# Możliwości

- Silne szyfrowanie (AES, Blowfish, 3DES)
- Silne metody uwierzytelniania (PKI, OTP, Kerberos)
- Transmisja sesji X11
- Przekierowywanie portów
- SSO
- SFTP - transmisja plików
- Kompresja transmisji
- VPN - PPPoSSH

# Wersje

SSH1 to pierwsza wersja protokołu. Odkryto w niej luki, które zaowocowały powstaniem SSH2 <sup>a</sup>

- Łamanie haseł (RC4 + blokowanie pustych haseł)
- Powtarzanie połączeń (RC4)
- Modyfikacja transmitowanych danych (RC4 + CRC)
- Przenoszenie autoryzacji na inny serwer (gdy szyfrowanie wyłączone)

SSH2 nie jest kompatybilne z SSH1. OpenSSH implementuje obie wersje protokołu.

---

<sup>a</sup>Luki nie są krytyczne i mogły zostać stosunkowo łatwo poprawione. Marketing.

# Specyfikacja

SSH2 jest zdefiniowane w następujących dokumentach:

- draft-ietf-secsh-architecture-12
- draft-ietf-secsh-transport-14
- draft-ietf-secsh-userauth-15
- draft-ietf-secsh-connect-15

# Architektura SSH2

Protokół SSH składa się z 3 warstw:

- Transportowej (uzgadnianie algorytmów i kluczy). Zapewnia:
  - Integralność
  - Poufność
  - Kompresję danych (opcjonalnie)
  - *Perfect Forward Secrecy*
- Autentykacji (hasła, PKI itd.)
- Połączenia (z wielokrotnianie)

# Architektura SSH2 cd.

- SSH działa w sieciach IP.
- Architektura klient – serwer.
- Serwer nasłuchuje na porcie 22/TCP.

# Serwer

- Podczas instalowania serwera SSH administrator generuje *klucz hosta* dla tej maszyny.
- Kluczy może być tyle, ile algorytmów obsługuje serwer.
- Obsługiwane algorytmy asymetryczne:
  - RSA
  - DH
  - DSA
- Modele certyfikacji kluczy:
  - Baza kluczy na kliencie
  - CA

# Centrum Certyfikacji

Schemat działania systemu opartego o centrum certyfikacji:

- CA generuje parę kluczy: publiczny i prywatny
- Klucze publiczne serwerów są podpisywane przez CA <sup>a</sup>
- Każdy klient instaluje klucz CA

Gdy klient posiada klucz CA może z łatwością zweryfikować poprawność certyfikatu serwera — wystarczy sprawdzić czy podpis jest prawidłowy.

---

<sup>a</sup> Podpisany klucz publiczny to *certyfikat*

# Baza kluczy na kliencie

Jest to rozwiązanie najpopularniejsze, ponieważ nie wymaga Infrastruktury Klucza Publicznego (*PKI*).

- Przy pierwszym połączeniu klient ściąga z serwera jego klucz publiczny.
- Klient powinien zaprezentować użytkownikowi *odcisk palca* klucza serwera.
- Użytkownik powinien zweryfikować za pomocą innego kanału czy odcisk palca zgadza się.
- Klucz zapisywany jest w lokalnej bazie.

```
joe@localhost:~$ ssh joe@serwer
```

```
The authenticity of host 'serwer (a.b.c.d)' can't be established.
```

```
RSA key fingerprint is 4f:f6:35:c6:c3:22:05:ad:c3:80:c8:5f:b7:06:6c:b4
```

```
Are you sure you want to continue connecting (yes/no)?
```

# Negocjacja parametrów

Połączenie SSH może korzystać z wielu dostępnych algorytmów i protokołów kryptograficznych. Decyzja o użyciu konkretnego zbioru algorytmów zapada na etapie negocjacji parametrów połączenia.

Ustalania dotyczą m.in.:

- Algorytmu symetrycznego
- Algorytmu asymetrycznego
- Metody autoryzacji
- Operacji, które użytkownik może wykonać korzystając z tego połączenia (przenoszenie połączeń X11 etc.)

Preferencje dot. ustaleń określa polityka (ang. *Policy*).

# Transmisja

Format pakietu SSH (warstwa transportowa):

- Długość pakietu (4 bajty)
- Długość dopełnienia (1 bajt)
- Dane
- Losowe dopełnienie
- *MAC* - tylko ta część pakietu nie jest szyfrowana!

Dopełnienie jest dobierane tak, aby wielkość całego pakietu z wyjątkiem *MACa* była wielokrotnością długości bloku <sup>a</sup> algorytmu szyfrującego.

---

<sup>a</sup> Dla algorytmów strumieniowych - wielokrotnością 8

# Zabezpieczenia

Poufność: Szyfrowanie, najczęściej blokowe w trybie CBC.

Integralność:

$$mac = MAC(K, S||P)$$

$K$  – klucz,

$S$  – numer pakietu,

$P$  – niezaszyfrowany pakiet

Transmisja jest rozbita na dwa kanały: wysyłanie i odbieranie. Szyfrowanie i zapewnienie integralności jest realizowane oddzielnie dla obu kanałów.

# Uzgadnianie kluczy

Gdy zakończy się faza *KEX*, klient i serwer dysponują parą tajnych danych: *K* i *H*

Za pomocą:

$$x = \text{HASH}(K \| H \| \textit{litera} \| \textit{id\_sesji})$$

generowane są:

- Wektory inicjalizujące
- Klucze szyfrujące
- Klucze używane do zapewnienia integralności (*MAC*)

# Przebieg połączenia

Warstwa transportowa.

1. Wymiana wersji oprogramowania.
2. Wymiana listy obsługiwanych algorytmów i protokołów.
3. Uzgodnienie kluczy  $K$  i  $H$ .
4. Wygenerowanie kluczy sesji.
5. Przekazanie  $H$  do warstwy autentykacji.

Renegocjacja kluczy jest powtarzana co godzinę lub co 1GB przesłanych danych, w zależności od tego co nastąpi wcześniej.

# Przebieg połączenia cd.

W warstwie autentykacji także dochodzi do negocjacji protokołu uwierzytelniania.

- *password*
- *publickey*
- *hostbased*

Wybrana metoda uwierzytelniania zależy od polityki zdefiniowanej na serwerze i na kliencie.

# password

Autoryzacja oparta na haśle.

- Użytkownik wpisuje hasło, które jest wysyłane do serwera.
- Poufność jest zapewniana przez warstwę transportową.

Wada tej metody jest to, że hasło może zostać przechwycone przez:

- nieuczciwego administratora
- osobę wykonującą atak człowiek w środku

# publickey

1. Użytkownik za pomocą bezpiecznego kanału umieszcza swój klucz publiczny na serwerze.
2. Klucz prywatny użytkownika znajduje się tylko na jego komputerze; jest chroniony hasłem.
3. Podczas logowania użytkownik podpisuje pakiet danych, które zawierają między innymi  $H$  i wysyła do serwera.
4. Serwer sprawdza, czy podpis jest prawidłowy używając klucza publicznego użytkownika.
5. Jeśli podpis jest prawidłowy serwer zezwala użytkownikowi na dostęp.

Hasło nie opuszcza komputera użytkownika!

# Perfect Forward Secrecy

Serwer oprócz kluczy *host key*, które są stałe, co godzinę generuje parę kluczy *temp key*.

- Po nawiązaniu połączenia klient ściąga z serwera *temp key*
- Do uzgodnienia kluczy szyfrujących wykorzystywane są obie pary kluczy.

Jeśli atakujący:

- Podslucha całą zaszyfrowaną sesję i...
- przechwyci prywatny klucz serwera *host key*

to ma co najwyżej godzinę na przechwycenie prywatnego klucza tymczasowego *temp key*. Po tym czasie transmisji nie da się odszyfrować!

# Atak człowiek w środku

1. A łączy się z M, myśląc, że komunikuje się z B.
2. M przedstawia A swój klucz publiczny.
3. A naiwnie go akceptuje <sup>a</sup>.
4. A wysyła hasło M, cały czas myśląc, że komunikuje się z B.
5. M wysyła hasło B, i dalej działa jako pośrednik podsłuchując transmisję.

Atakujący poznaje hasło A.

Można się przed tym zabezpieczyć stosując autentykację opartą na kluczu publicznym i/lub weryfikując klucz publiczny serwera.

---

<sup>a</sup>Fuzzy Fingerprint

# Przykład MitM

```
@@@@@@@@@@@@server@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

```
Someone could be eavesdropping on you right now (man-in-the-middle  
attack)!
```

```
It is also possible that the RSA host key has just been changed.
```

```
The fingerprint for the RSA key sent by the remote host is
```

```
4f:f6:35:c6:c3:22:05:ad:c3:80:c8:5f:b7:06:6c:b4.
```

```
Please contact your system administrator.
```

```
Add correct host key in /home/joe/.ssh/known_hosts to get rid of  
this message.
```

```
Offending key in /home/joe/.ssh/known_hosts:30
```

```
RSA host key for server has changed and you have requested  
strict checking.
```

```
Host key verification failed.
```