SSH (Secure Shell)

- Protocole sécurisé (rfc 4250-54 publiées en 2006) pour accès distant *(secure remote access)*
- Architecture client – serveur

  Labo : client = putty (Windows 7) – serveur = Linux Fedora 16 CLI
- Authentification du serveur via clé publique
- Confidentialité & intégrité des messages échangés
- Authentification du client : *username-password*, …, clé publique
- Tunnel *(port forwarding)*
- Analogie avec protocole SSL
- Version 1 (1995) obsolète

Méthode de Diffie – Hellman (DH)

- En 1976, Whitfield Diffie & Martin Hellman publie une méthode permettant à Alice et Bob de partager un secret K à partir de l'échange des 2 clés publiques

	Alice	Bob
DH group (1024 bit)	p, g	p, g
Private (random)	x	y
Public (sended)	$e=g^x \bmod p$ →	← $f=g^y \bmod p$
Shared Secret Key K =	**$f^x \bmod p$**	**= $e^y \bmod p$**

Méthode DH (suite)

	Alice	Bob
• DH group (1024 bit)	p, g	p, g

A et B se mettent d'accord sur un nombre premier p et une racine primitive g

A choisit un nombre secret $0 < x < p - 1$

B choisit un nombre secret $0 < y < p - 1$

• Private (random)	x	y

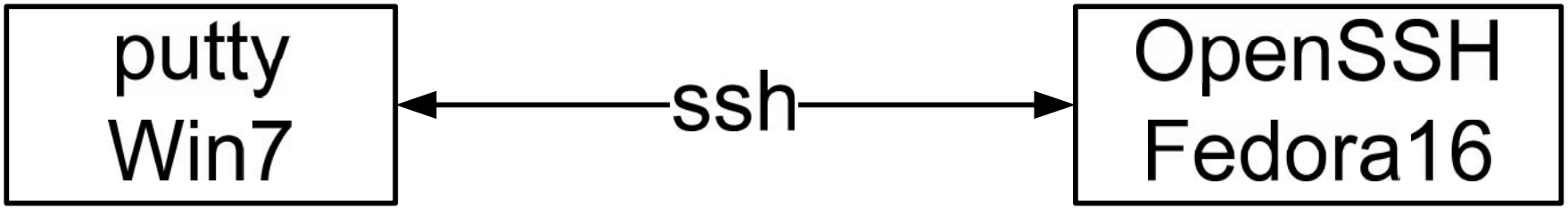
A envoie la valeur e

B envoie la valeur f

• Public (sended)	$e=g^x \bmod p$ →		← $f=g^y \bmod p$
Shared Secret Key K =	**$f^x \bmod p$**	**=**	**$e^y \bmod p$**

Voir EPFL_Beuret_Grandperrin.pdf (labo → 10.2.1.1/doclabo)

Labo §1 : Utilisation du client SSH (10 min)



§1.1 Show fingerprinting of server pub key

```
ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub
2048 45:7b:6c:18:1d:58:f9:a8:07:7c:b1:16:44:12:5c:d6
```

§1.2 Connexion depuis client putty (next slide)

Contrôle manuel de la clé publique (next slide)

§1.3 Afficher la clé stockée dans la base de registres du PC-Win

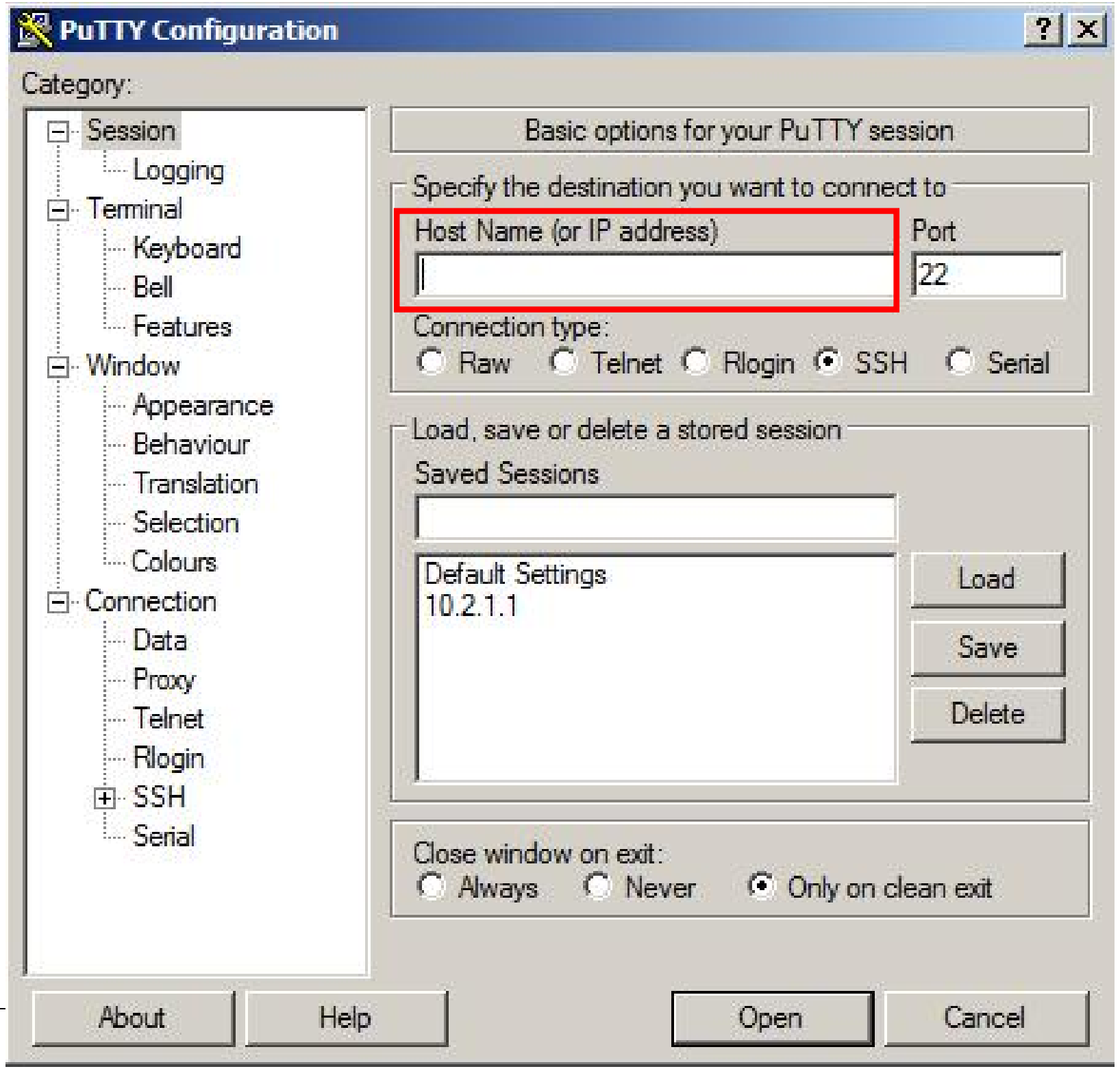
```
rsa2@22:10.2.2.6        REG_SZ      0x10001,0xb2fd95df5e698c2ec6e0452306e1b72ae89d4699d2b9af79dd27f8b957f1
```

§1.4 Connexion en mode CLI

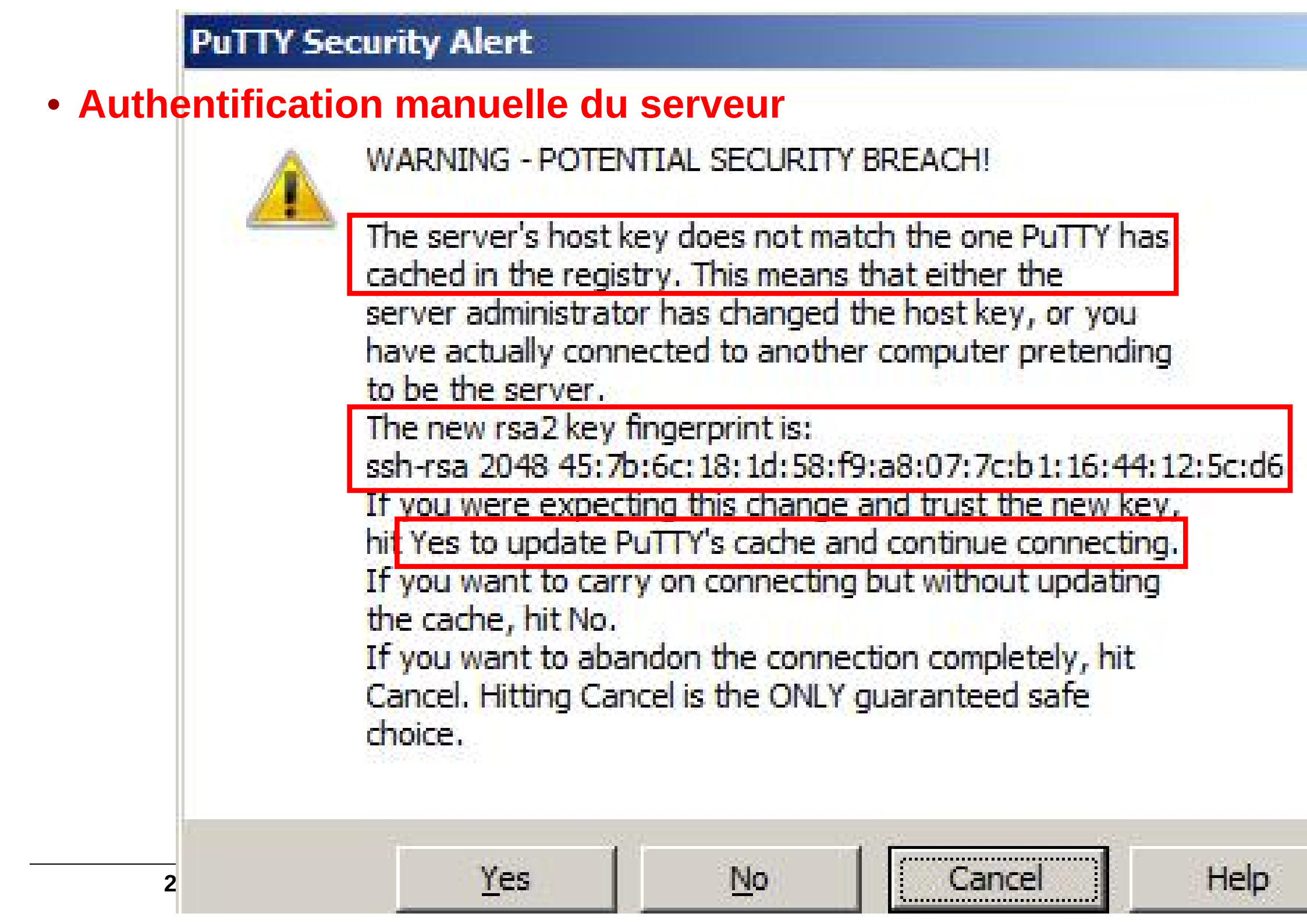
```
putty -ssh labotd@IP_Server
```

PuTTY

- Client SSH
- Connexion au serveur



- **Authentification manuelle du serveur**



Labo §2 : Clés stockées sur le serveur (20 min)

- **`/etc/ssh`** Dossier
- **`sshd.config`** Configuration

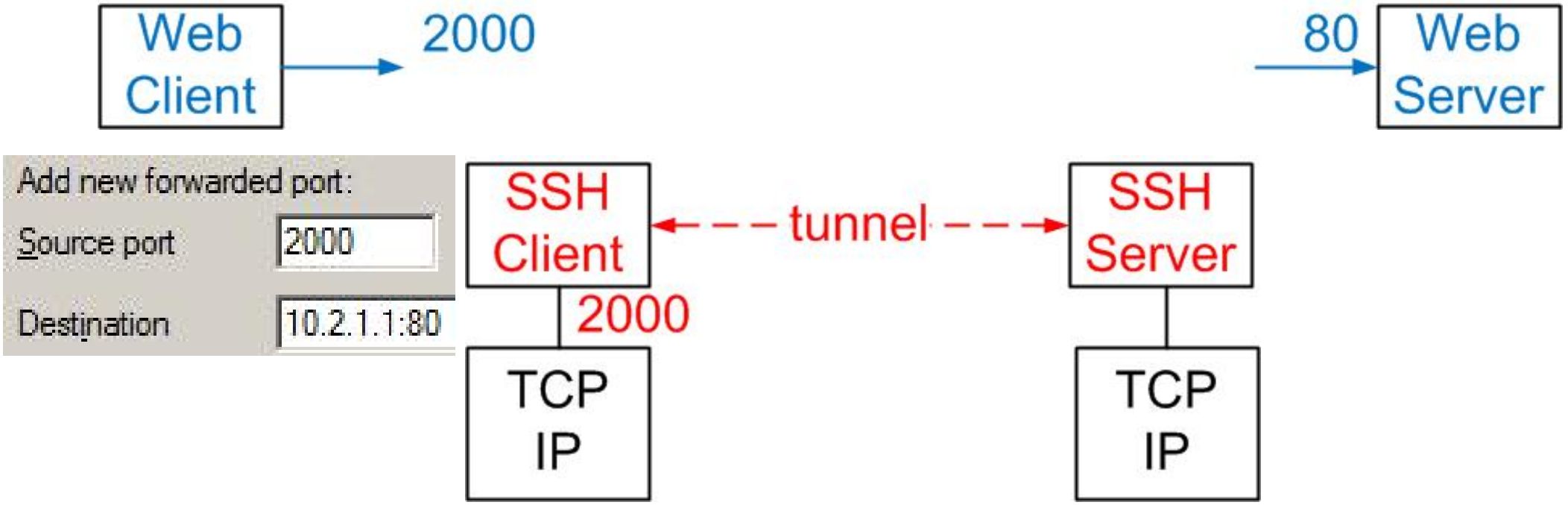
  Paramètres Port, ListenAdress, Protocol, PermitRootLogin, Banner

  voir slides 20-24
- **`ssh_host_rsa_key`** Private key for SSHv2
- **`ssh_host_rsa_key.pub`** Public key for SSHv2
- **`$HOME/.ssh/authorized_keys`** Clé(s) publique(s) des clients autorisés
- https://www.centos.org/docs/5/html/Deployment_Guide-en-US/s1-ssh-configfiles.html

Labo §3 : Tunnel SSH - port forwarding (30 min)

- Problématique (alternative à SSL)

- Service sécurisé (confidentialité + intégrité) à travers internet



Labo §3 : Tunnel SSH (suite)

§3.1 Etablir une connexion http sans tunnel

§3.2 Créer un tunnel

Client SSH mis en écoute du port TCP 2000

Le socket **localhost:2000** est redirigé sur **10.2.1.1:80**

Client web utilise **`http://localhost:2000`**

§3.3 Analyse Wireshark

Configurer les filtres → méthodologie de mesure

L'échange est-il sécurisé (chiffré) ?

Quels sont les ports utilisés ?

SSH protocol (rfc 4253)

- Server normally listens on TCP port 22 (§4.1)

- Both sides MUST send an identification string (§4.2)

  ← **`SSH-2.0-OpenSSH_5.8`** Wireshark packet 1

  → **`SSH-2.0-PuTTY_Release_0.63`** packet 2

- Vulnerable to man-in-the-middle attacks (rfc 4251 §9.3.4)

- Key exchange (KEX) begins by each side sending name-lists of supported algorithms (§7)

  → **`KEXINIT (Key Exchange Init)`** packet 3 → next slide

  ← **`KEXINIT (Key Exchange Init)`** packet 4

```
SSH Version 2 (encryption:aes256-ctr mac:hmac-sha2-256 compression:none)
  Packet Length: 668
  Padding Length: 10
⊟ Key Exchange
    Message Code: Key Exchange Init (20)
  ⊟ Algorithms
      Cookie: 27ad96bd0b6b0df72f86751e1b4b5e27
      kex_algorithms length: 154
      kex_algorithms string: diffie-hellman-group-exchange-sha256,diffie-
      server_host_key_algorithms length: 15
      server_host_key_algorithms string: ssh-rsa,ssh-dss
      encryption_algorithms_client_to_server length: 159
      encryption_algorithms_client_to_server string: aes256-ctr,aes256-cb
      encryption_algorithms_server_to_client length: 159
      encryption_algorithms_server_to_client string: aes256-ctr,aes256-cb
      mac_algorithms_client_to_server length: 45
      mac_algorithms_client_to_server string: hmac-sha2-256,hmac-sha1,hma
      mac_algorithms_server_to_client length: 45
      mac_algorithms_server_to_client string: hmac-sha2-256,hmac-sha1,hma
      compression_algorithms_client_to_server length: 9
      compression_algorithms_client_to_server string: none,zlib
      compression_algorithms_server_to_client length: 9
      compression_algorithms_server_to_client string: none,zlib
      languages_client_to_server length: 0
      languages_server_to_client length: 0
      KEX First Packet Follows: 0
      Reserved: 00000000
    Padding String: 0fa033b73af22daaccce
```

SSH protocol (rfc 4253)

- Kex_algorithms → (§6.5)

  Méthode diffie-hellman-group-exchange-sha256 supportée (1er choix) par client & serveur packet 3 & 4

- A DH group is defined by a prime modulus and a generator
- The prime modulus is a bit expensive to generate, so OpenSSH will not do that on a general basis
- Instead, sshd will, upon receiving a connection, use one of the groups in the /etc/ssh/moduli files
- That file contains pre-generated moduli of various sizes.
- Source = http://security.stackexchange.com/questions/39603/whats-the-modp-length-of-diffie-hellman-group-exchange-sha256

SSH protocol (rfc 4253)

- Server_host_key_algorithms → §7.1

  Format ssh-rsa = format de la clé publique → §6.6

  raw RSA key → no certificate – no signature !

- Encryption_algorithms → §7.1 (analogie avec SSL)

  Client préfère aes256-ctr (packet 3)

  → http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

  Server préfère aes128-ctr (packet 4)

- Mac_algorithms → §7.1 (analogie avec SSL)

  Client préfère hmac-sha2-256 (packet 3)

  Server ne supporte que hmac-md5 et hmac-sha1 (packet 4)

SSH protocol (rfc 4253)

→ **KEXINIT (Key Exchange Init)** packet 5

- After the KEXINIT message exchange, the key exchange algorithm is run

- Envoi de p & g

  ← **KEXINIT (Key Exchange Reply)** packet 6

  ← **KEX_DH_GEX_GROUP** selon rfc 4419 §5

- C generates x (random number), where $1 < x < (p-1)/2$

  C computes **$e = g^x \bmod p$**

  → **DH GEX Init (e)** packet 7

SSH protocol (rfc 4253 - §8)

- S generates y (random number), where 0 < y < (p-1)/2

  S computes **$f = g^y \bmod p$**

  S receives **e** computes **$K = e^y \bmod p$**

  ← DH GEX Reply (S_pub_key || f || Signature) packet 8

- **Logs de putty**

```
Host key fingerprint is:
ssh-rsa 2048
0b:34:cf:a1:5a:10:0d:53:23:b3:13:79:40:ec:c8:cc
```

SSH protocol (rfc 4253)

→ **`New Keys`** packet 9

- Key exchange ends with NEWKEYS message
- All messages sent after this message MUST use the new keys and algorithms

packets 10-25 sont chiffrés

- **Utiliser les logs de putty pour la suite de l'analyse**

Initialised AES-256 SDCTR client->server encryption

Initialised HMAC-SHA1 client->server MAC algorithm

← **`New Keys`** packet 10

Initialised AES-256 SDCTR server->client encryption

Initialised HMAC-SHA1 server->client MAC algorithm

SSH protocol (rfc 4253)

→ **SSH2_MSG_SERVICE_REQUEST (ssh-userauth)** p11

← **SSH2_MSG_SERVICE_ACCEPT (ssh-userauth)** p12

→ **SSH2_MSG_USERAUTH_REQUEST (labotd)** p13

← **SSH2_MSG_USERAUTH_FAILURE** p14

- Attempting GSSAPI authentication

→ **SSH2_MSG_SERVICE_REQUEST (ssh-userauth,...)** p15

... p16-18

→ **SSH2_MSG_USERAUTH_SUCCESS** p19

SSH protocol (rfc 4253)

- Opening session as main channel

  → **SSH2_MSG_CHANNEL_OPEN (session)** p20

  ← **SSH2_MSG_CHANNEL_OPEN_CONFIRMATION** p21

  → **SSH2_MSG_CHANNEL_REQUEST (xterm)** p22

  → **SSH2_MSG_CHANNEL_REQUEST (shell)** p23

  ← **SSH2_MSG_CHANNEL_SUCCESS** p24

- Started a shell

  ← **SSH2_MSG_CHANNEL_DATA (labotd@localhost:~.** p25

Labo §4 : Analyse Wireshark et logs (30 min)

- Quelle est l'adresse IP du serveur ?
- Quelle est la chaîne d'identification du serveur ?
- Quelle est la chaîne d'identification du client ?
- Quel est l'algorithme utilisé pour le chiffrement des données ?
- Quel est l'algorithme utilisé pour le contrôle d'intégrité ?
- Dans quel paquet se trouve la valeur e ?
- Dans quel paquet se trouve la valeur f ?
- Dans quel paquet se trouve la clé publique du serveur ?
- Quelle est l'utilité de la signature dans le paquet 8 ?
- Quelle est l'invite de commande (prompt) obtenue par le client ?

Audit & Hardening d'un serveur (1/5)

- La config. par défaut de OpenSSH peut être améliorée → renforcée

  Modifier **`sshd.config`** puis **`service sshd restart`**

- Illustration avec la version OpenSSH_5.3p1 (mars 2010)
- Chaque renforcement proposé doit être testé → test unitaire

  Utiliser le client SSH → **`ssh ...`**

- Les renforcements proposés présentent des dépendances

  Bloquer l'accès distant (SSH) à root

- Ils sont énumérés dans l'ordre de leur utilisation par le protocole

1) Version du logiciel ? **`ssh -V`** Mise à jour nécessaire ?
2) Modifier le port TCP d'écoute (= 22 par défaut)

   `Port 37589` **`ssh IP:37589`**
3) Limiter l'accès à la version 2 du protocole

   `Protocol 2` **`ssh –1`**

Audit & Hardening d'un serveur (2/5)

4) Quand la paire de clés RSA a-t-elle été créé ?

```
Date de ssh_host_rsa_key        Private key for SSHv2
Date de ssh_host_rsa_key.pub    Public key for SSHv2
```

5) Si nécessaire générer une nouvelle paire de clés RSA

```
ServerKeyBits 2048                  Définir la taille
ssh-keygen –t rsa –f /etc/ssh/ssh_host_rsa_key
```

6) Authentification de l'utilisateur par mot de passe imposée

```
PasswordAuthentication yes
ChallengeResponseAuthentication no
UsePAM yes
HostbasedAuthentication no
IgnoreUserKnownHosts yes
IgnoreRhosts yes
```

Audit & Hardening d'un serveur (3/5)

7) Créer une liste des utilisateurs autorisés

```
AllowUsers pierre paul
AllowGroups secretaire
```

8) Empêcher une authentification sans mot de passe

```
PermitEmptyPassword no
```

9) Limiter nombre de tentatives d'authentification (brute force attack)

```
PMaxAuthTries 3
```

10) Diminuer l'intervalle de temps pendant lequel l'authentification est possible (120 secondes par défaut)

```
LoginGraceTime 30
```

11) Check user's permissions in home directory before accepting login

```
StrictModes yes
```

Audit & Hardening d'un serveur (4/5)

12) Exiger une longueur minimale de clés symétriques

Ciphers aes128-ctr,aes192-ctr,aes256ctr,
arcfour256,arcfour128,aes128-cbc,3des-cbcxx

Tester en modifiant ssh.config

ssh –c pour tenter une connexion sans chiffrement

13) Interdire des fonctions de hachage faibles

MACs hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160

Tester en modifiant ssh.config

14 Déconnecter un client inactif

ClientAliveInterval 120

ClientAliveCountMax 0 **???**

TCPKeepAlive no

Audit & Hardening d’un serveur (5/5)

15) Pas de mode graphique sur le serveur du labo

```
X11Forwarding no
```

16) Set a warning banner

```
Xxx
```

17) Bloquer l’accès distant (SSH) à root

```
PermitRootLogin no
```

A la fin pour faciliter les test unitaires

Port Knocking

- Le port ouvert (TCP 22 ou autre) peut constituer un risque
- La solution port knocking consiste à ouvrir ce port après une sequence plus ou moins complexe → exemple TCP 2005 – 1905 – 3005
- Un service (daemon) comme knockd doit être actif côté serveur ou firewall
- /etc/knockd.conf :

```
sequence = 2005,1905,3005
seq_timeout = 20 tcpflags = syn
start_command = /sbin/iptables ... --dport 22 -j ACCEPT
stop_command = /sbin/iptables  ... --dport 22 -j ACCEPT
```

- https://www.it-connect.fr/chapitres/configuration-du-port-knocking-ssh/

  → **security by obscurity !**

Liens utiles

- OpenSSH hardening by Jean-Philippe Aumasson
  http://cybermashup.com/2013/11/28/openssh-hardening-for-cloud-machine-part-1/
- Top 20 OpenSSH Server Best Security Practices
  http://www.cyberciti.biz/tips/linux-unix-bsd-openssh-server-best-practices.html
- rfc 4250-54 (2006) → http://tools.ietf.org/html/rfc4253
- rfc 4419 (2006) → http://tools.ietf.org/html/rfc4419
- putty user manual → http://the.earth.li/~sgtatham/putty/0.58/htmldoc/
- Man pages → http://www.openssh.com/manual.html
- Server statistics → http://sshdb.tmptech.net/
- Wikipedia → http://fr.wikipedia.org/wiki/Secure_Shell
- SSH by William Stallings → http://www.ipjforum.org/?p=180

Mode verbose : ssh –v (1/3)

- *OpenSSH_5.3p1 Debian-3ubuntu7, OpenSSL 0.9.8k 25 Mar 2009*
- *Reading configuration data /etc/ssh/ssh_config*
- *Applying options for **
- *Connecting to 10.1.40.124 [10.1.40.124] port 22.*
- *Connection established.*
- *identity file /home/labotd/.ssh/identity type -1*
- *identity file /home/labotd/.ssh/id_rsa type -1*
- *identity file /home/labotd/.ssh/id_dsa type -1*
- *Remote protocol version 2.0, remote software version OpenSSH_5.5*
- *match: OpenSSH_5.5 pat OpenSSH**
- *Enabling compatibility mode for protocol 2.0*
- *Local version string SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7*

Mode verbose : ssh –v (2/3)

- *SSH2_MSG_KEXINIT sent* *Algorithmes*
- *SSH2_MSG_KEXINIT received*
- *kex: server->client aes128-ctr hmac-md5 none*
- *kex: client->server aes128-ctr hmac-md5 none*
- *SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent*
- *expecting SSH2_MSG_KEX_DH_GEX_GROUP*
- *SSH2_MSG_KEX_DH_GEX_INIT sent*
- *expecting SSH2_MSG_KEX_DH_GEX_REPLY*
- *Server host key: RSA ff:9e:8e:fb:d7:2c:d5:f8:2f:a3:71:56:35:24:ff:6a*
- *Host ’10.1.40.124’ is known and matches the RSA host key.*
- *Found key in /home/labotd/.ssh/known_hosts:1*
- *ssh_rsa_verify: signature correct*

Mode verbose : ssh –v (3/3)

- *SSH2_MSG_NEWKEYS sent*
- *expecting SSH2_MSG_NEWKEYS*
- *SSH2_MSG_NEWKEYS received*
- *Roaming not allowed by server*
- *SSH2_MSG_SERVICE_REQUEST sent*
- *SSH2_MSG_SERVICE_ACCEPT received*
- *Authentications that can continue: publickey, gssapi-keyex, gssapi-with-mic, password*
- *…*
- *Next authentication method: password*
- *labotd@10.1.40.124’s password:*

SSH Protocol Architecture (rfc 4251)

- Vue d'ensemble, définitions

- §4.4 *Security Properties*

  All encryption, integrity, and public key algorithms used are well-known, well-established algorithms.
  *All algorithms are used with cryptographically sound **key sizes** that are believed to provide protection against even the strongest cryptanalytic attacks for decades.*

- §9 *Security Considerations*

  The transport protocol [SSH-TRANS] provides a confidential channel over an insecure network. It performs server host authentication, key exchange, encryption, and integrity protection. It also derives a unique session id that may be used by higher-level protocols.

SSH Protocol Architecture (rfc 4251)

- §9 *Security Considerations*

  The authentication protocol [SSH-USERAUTH] provides a suite of mechanisms that can be used to authenticate the client user to the server.

- §9.1 *Pseudo-Random Number Generation*

  Special care should be taken to ensure that all of the random numbers are of good quality.

- §9.2 *Control Character Filtering*

  When displaying text to a user, such as error or debug messages, the client software SHOULD replace any control characters (except tab, carriage return, and newline) with safe sequences to avoid attacks by sending terminal control characters.

SSH Protocol Architecture (rfc 4251)

- §9.3.1 *Confidentiality*
  Beyond the scope of this document.

- §9.3.2 *Data Integrity*

  Because MACs use a 32-bit sequence number, they might start to leak information after 2^{32} packets have been sent.

  Therefore, rekeying SHOULD happen after 2^{28} packets

- §9.3.3 *Replay*

  … the transport protocol provides a unique session identifier (bound in part to pseudo-random data that is part of the algorithm and key exchange process) that can be used by higher level protocols to bind data to a given session and prevent replay of data from prior sessions.

SSH Protocol Architecture (rfc 4251)

- §9.3.4 *Man-in-the-middle*
  This section describes this and encourages administrators and users to understand the importance of verifying this association before any session is initiated.
  → même problématique que pour la connexion SSL
  Secure if endpoints = Alice & Bob → authentication !!!

- §9.3.5. *Denial of Service*
  this protocol is vulnerable to denial of service attacks because an attacker can force the server to go through the CPU and memory intensive tasks of connection setup and key exchange without authenticating.

SSH Protocol Architecture (rfc 4251)

- §9.3.6 *Covert Channels*

  the padding, SSH_MSG_IGNORE messages, and several other places in the protocol can be used to pass covert information

- §9.3.7 *Forward Secrecy*

  PFS is essentially defined as the cryptographic property of a key-establishment protocol in which the compromise of a session key or long-term private key after a given session does not cause the compromise of any earlier session

  La méthode DH utilisée exige de nouvelles valeurs (aléatoires) des grandeurs x et y à chaque *rekeying*

  SSL autorise la réutilisation des clés (voir session key cache : timeout, clear)

SSH Protocol Architecture (rfc 4251)

- §9.3.8 *Ordering of Key Exchange Methods*
  each device will send a list of preferred methods for key exchange. The most-preferred method is the first in the list. It is RECOMMENDED that the algorithms be sorted by cryptographic strength, strongest first.

- §9.3.9 Traffic Analysis
  Passive monitoring of any protocol may give an attacker some information about the session, the user, or protocol specific information that they would otherwise not be able to garner. For example, it has been shown that traffic analysis of an SSH session can yield information about the length of the password. Implementers should use the SSH_MSG_IGNORE packet, along with the inclusion of random lengths of padding, to thwart attempts at traffic analysis.

SSH Protocol Architecture (rfc 4251)

- §9.4. *Authentication Protocol*
  The purpose of this protocol is to perform client user authentication. It assumes that this runs over a secure transport layer protocol, which has already authenticated the server machine, established an encrypted communications channel, and computed a unique session identifier for this session.

  Several authentication methods with different security characteristics are allowed.

  The server may go into a sleep period after repeated unsuccessful authentication attempts to make key search more difficult for attackers.

SSH Protocol Architecture (rfc 4251)

- §9.4.4 *Public Key Authentication*
  The use of public key authentication assumes that the client host has not been compromised. It also assumes that the private key of the server host has not been compromised.

  This risk can be mitigated by the use of passphrases on private keys. The use of smartcards, or other technology to make passphrases an enforceable policy is suggested.