Secure SHell (SSH)

Chapter 7

Network & Security

Gildas Avoine

INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
RENNES

IUF

SUMMARY OF CHAPTER 7

- Primer
- Security Mechanism
- File Transfer
- Port Forwarding
- Conclusion

PRIMER

- Primer
- Security Mechanism
- File Transfer
- Port Forwarding
- Conclusion

Security Over the Transport Layer

- To offer a single secure service (web, mail, login), it is better to secure the data at the transport layer than at the network layer.

- Two most commonly used protocols are:

  - SSH (Secure SHell, 1995, Port 22) : allows secure logins, file transfer, etc.

  - SSL/TLS (Secure Socket Layer, 1995): allows securing any TCP based service (https, pop3s, telnets, ftps, esmtp,...).

SSH Objectives

- SSH (RFC 4251) implements secure communication channels over insecure networks in a client-server session.
  - Confidentiality, authenticity, integrity.
- Original philosophy.
  - User-friendly (SSH designed to replace telnet, rlogin, rsh, ftp).
  - Ready to use without any complicated installation.
- The security level is not so high, but much higher than telnet-like tools: data is encrypted, passwords are no longer exchanged in the clear.

Products

- SSH created and commercialized by Tatu Ylönen in Finland (www.ssh.com).
- OpenSSH is a public domain implementation.
  - OpenSSH is a library that allows creating SSH servers and clients.
- Examples of clients: Putty, SecureCRT,...

SSH Versions

- SSH1 (1995).
  - RSA (patented till 2000).
  - 3DES, Blowfish, and possibly IDEA (not free for commercial use).
  - CRCs to verify the data's integrity.
  - Attacks possible, even though limited.
- SSH2 (2006).
  - DSA (copyrights free) to authenticate the server.
  - MAC instead of CRC.
  - Standardized by the IETF, available in open source.

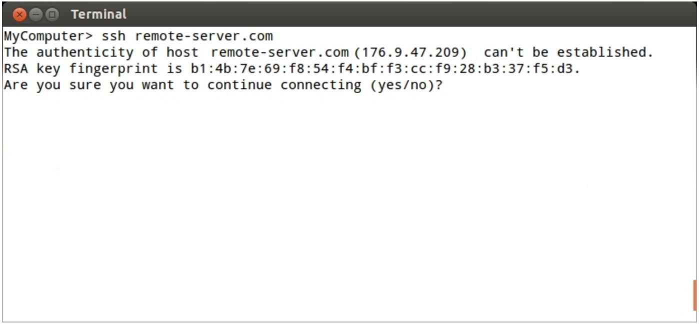
SECURITY MECHANISM

- Primer
- Security Mechanism
- File Transfer
- Port Forwarding
- Conclusion

Architecture

- Transport layer (RFC-4253)
  - Handles server authentication, cryptographic protocols agreement, initial key exchange, ...
- Client/User authentication layer (RFC-4252)
  - Handles client authentication.
- Connection layer (RFC-4254)
  - Manages the services: Shell, SFTP, SCP, port-forwarding,...

Transport Layer: Authenticating the Server

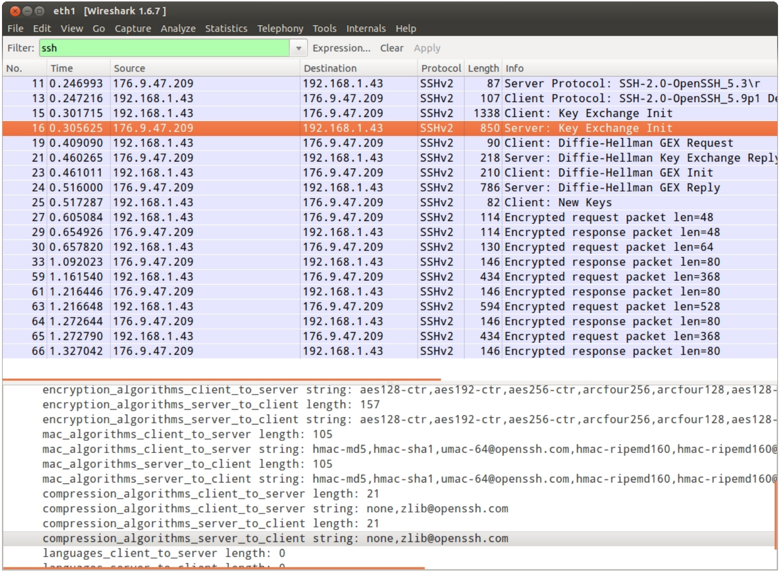


```
MyComputer> ssh remote-server.com
The authenticity of host  remote-server.com (176.9.47.209)  can't be established.
RSA key fingerprint is b1:4b:7e:69:f8:54:f4:bf:f3:cc:f9:28:b3:37:f5:d3.
Are you sure you want to continue connecting (yes/no)?
```

Server Authentication Procedure

- Server and Client negotiate the cryptographic protocols.
- Server and Client agree on a symmetric key (DH protocol)
- Server signs the exchanged information using its private key.
- Server sends its public key (along with a certificate if any).

Packet Sniffing during SSH Server Authentication


eth1 [Wireshark 1.6.7]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ssh
Expression... Clear Apply

```
No. Time     Source       Destination  Protocol Length Info
 11 0.246993 176.9.47.209 192.168.1.43 SSHv2      87 Server Protocol: SSH-2.0-OpenSSH_5.3\r
 13 0.247216 192.168.1.43 176.9.47.209 SSHv2     107 Client Protocol: SSH-2.0-OpenSSH_5.9p1 De
 15 0.301715 192.168.1.43 176.9.47.209 SSHv2    1338 Client: Key Exchange Init
 16 0.305625 176.9.47.209 192.168.1.43 SSHv2     850 Server: Key Exchange Init
 19 0.409090 192.168.1.43 176.9.47.209 SSHv2      90 Client: Diffie-Hellman GEX Request
 21 0.460265 176.9.47.209 192.168.1.43 SSHv2     218 Server: Diffie-Hellman Key Exchange Reply
 23 0.461011 192.168.1.43 176.9.47.209 SSHv2     210 Client: Diffie-Hellman GEX Init
 24 0.516000 176.9.47.209 192.168.1.43 SSHv2     786 Server: Diffie-Hellman GEX Reply
 25 0.517287 192.168.1.43 176.9.47.209 SSHv2      82 Client: New Keys
 27 0.605084 192.168.1.43 176.9.47.209 SSHv2     114 Encrypted request packet len=48
 29 0.654926 176.9.47.209 192.168.1.43 SSHv2     114 Encrypted response packet len=48
 30 0.657820 192.168.1.43 176.9.47.209 SSHv2     130 Encrypted request packet len=64
 33 1.092023 176.9.47.209 192.168.1.43 SSHv2     146 Encrypted response packet len=80
 59 1.161540 192.168.1.43 176.9.47.209 SSHv2     434 Encrypted request packet len=368
 61 1.216446 176.9.47.209 192.168.1.43 SSHv2     146 Encrypted response packet len=80
 63 1.216648 192.168.1.43 176.9.47.209 SSHv2     594 Encrypted request packet len=528
 64 1.272644 176.9.47.209 192.168.1.43 SSHv2     146 Encrypted response packet len=80
 65 1.272790 192.168.1.43 176.9.47.209 SSHv2     434 Encrypted request packet len=368
 66 1.327042 176.9.47.209 192.168.1.43 SSHv2     146 Encrypted response packet len=80
```

```
encryption_algorithms_client_to_server string: aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-
encryption_algorithms_server_to_client length: 157
encryption_algorithms_server_to_client string: aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-
mac_algorithms_client_to_server length: 105
mac_algorithms_client_to_server string: hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160,hmac-ripemd160@
mac_algorithms_server_to_client length: 105
mac_algorithms_server_to_client string: hmac-md5,hmac-sha1,umac-64@openssh.com,hmac-ripemd160,hmac-ripemd160@
compression_algorithms_client_to_server length: 21
compression_algorithms_client_to_server string: none,zlib@openssh.com
compression_algorithms_server_to_client length: 21
compression_algorithms_server_to_client string: none,zlib@openssh.com
languages_client_to_server length: 0
languages_server_to_client length: 0
```

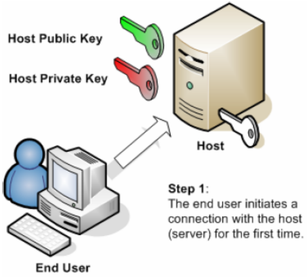
Verifying the Server Public Key

- First connection:
  - Client checks the certificate, or
  - Client requests user to authenticate public key by other means.
  - Client stores the public key in a local database (.ssh/known_hosts)
- Next connections:
  - Client checks the public key from the local database.

Critical Assumptions

- Public key authentication in first connection must be secure, otherwise server can be impersonated.
- The local database must have integrity protection, otherwise the server public key can be replaced by another one.

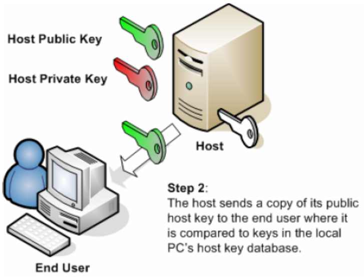
Illustrated Recap (1/3)



Step 1:
The end user initiates a connection with the host (server) for the first time.

Source: Van Dyke Software, 2004

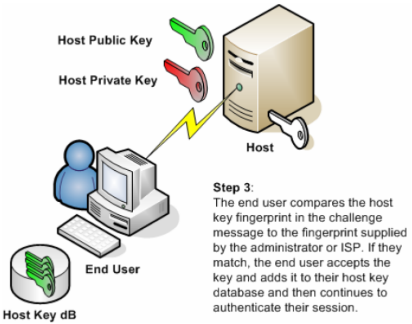
Illustrated Recap (2/3)



Step 2:
The host sends a copy of its public host key to the end user where it is compared to keys in the local PC's host key database.

Source: Van Dyke Software, 2004

Illustrated Recap (3/3)



Step 3:
The end user compares the host key fingerprint in the challenge message to the fingerprint supplied by the administrator or ISP. If they match, the end user accepts the key and adds it to their host key database and then continues to authenticate their session.

Source: Van Dyke Software, 2004

Main Client Authentication Methods

- The user provides a password.
- The client provides once his public key to the server, and then use his private key upon server request (e.g. Git repositories).

FILE TRANSFER

- Primer
- Security Mechanism
- File Transfer
- Port Forwarding
- Conclusion

File Transfer with SSH

- We can transfer files in a secure way using the SCP command:
  - scp mylocalfile.txt username@remote-server.com:myremotefile.txt
  - scp username@remote-server:myremotefile.txt .
- No NAT-related problem (cf. active/passive FTP).

PORT FORWARDING

- Primer
- Security Mechanism
- File Transfer
- Port Forwarding
- Conclusion

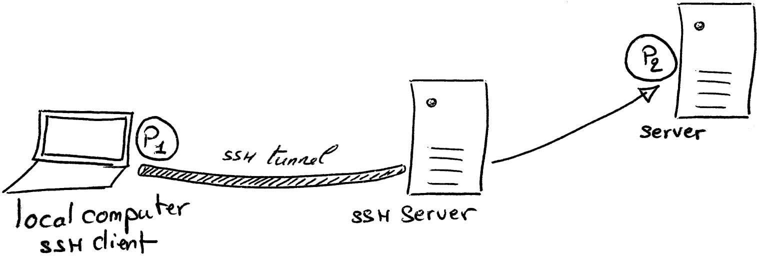
Port Forwarding

- Port forwarding: allows carrying any TCP connection across a SSH connection.

- Only the connection between the SSH client and the SSH server is protected.

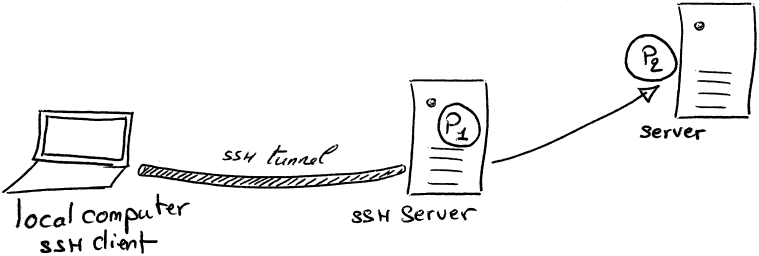
Local Port Forwarding

- The SSH client can forward a local port towards a given destination via an intermediate SSH server.
- The client application is configured to connect to a local port instead of a remote server.



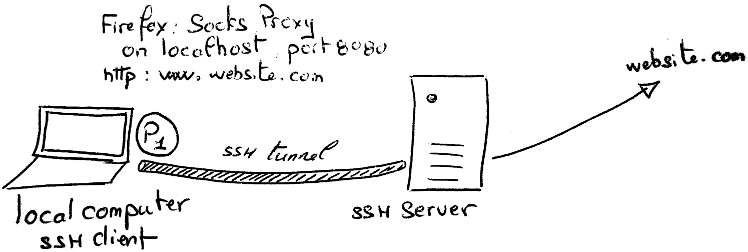
Remote Port Forwarding

- Port forwarding can also be done in the other direction.
  - Eg. tunnel from home to work initiated before leaving the office.
  - Eg. X Windows sessions, with SSH –X.

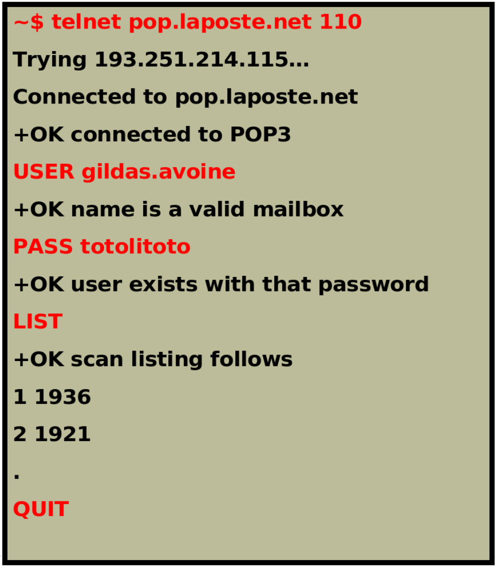


Dynamic Port Forwarding

- The SSH server behaves like a SOCKS Proxy.



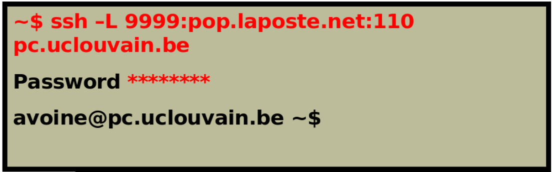
Example 1: Local Port Forwarding



```
~$ telnet pop.laposte.net 110
Trying 193.251.214.115...
Connected to pop.laposte.net
+OK connected to POP3
USER gildas.avoine
+OK name is a valid mailbox
PASS totolitoto
+OK user exists with that password
LIST
+OK scan listing follows
1 1936
2 1921
.
QUIT
```

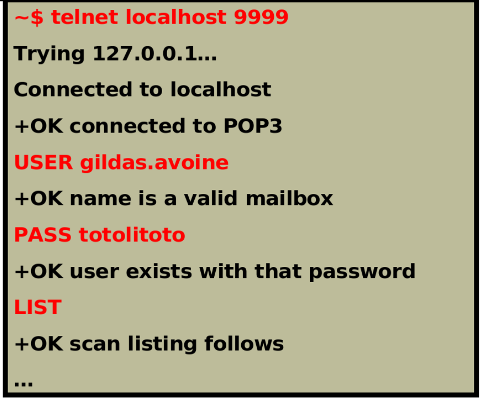
Example 1 (cont')

Set up of a SSH port forwarding on a remote machine.



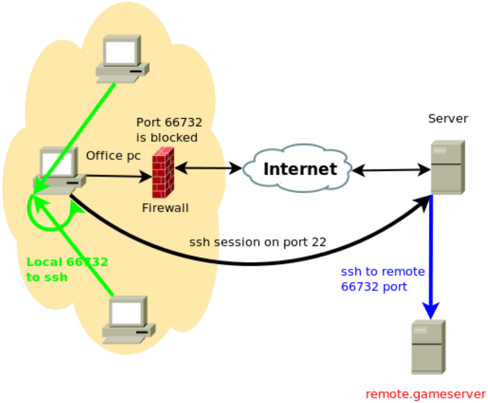
```
~$ ssh -L 9999:pop.laposte.net:110
pc.uclouvain.be
Password ********
avoine@pc.uclouvain.be ~$
```

Connection to the port 9999 of the localhost that is forward to the remote pop server.



```
~$ telnet localhost 9999
Trying 127.0.0.1...
Connected to localhost
+OK connected to POP3
USER gildas.avoine
+OK name is a valid mailbox
PASS totolitoto
+OK user exists with that password
LIST
+OK scan listing follows
...
```

Example 2: Local Port Forwarding



Source: http://toic.org/blog/2010/ssh-port-forwarding/#.VQCI6-HS07y

```
ssh -L 0.0.0.0:66732:remote.gameserver:66732 username@our.server
```

CONCLUSION

- Primer
- Security Mechanism
- File Transfer
- Port Forwarding
- Conclusion

Conclusion

- `ssh -L <local port>:<destination address>:<destination port> <ssh server>`
- `ssh -R <remote port>:<destination address>:<destination port> <ssh server>`
- `ssh -D <local port> <ssh server>`

References

- Comparison of SSH clients.
  - http://en.wikipedia.org/wiki/Comparison_of_SSH_clients
- Port forwarding
  - http://www.securityfocus.com/infocus/1816