SSH protocol, tips & tricks

Damien Miller
djm@mindrot.org

Overview

- Introduction
- History of SSH
- Theory
  - Protocol version 1
  - Protocol version 2
  - Security features
- Practical session
- Q & A

Introduction to SSH

SSH replaces rlogin, rsh and telnet

Almost completely compatible with rlogin and rsh

Authentication and the resulting session are encrypted to prevent traffic "sniffing"

Many additional features above legacy protocols

- Simple public key authentication
- Port-forwarding
- X11 forwarding
- Authentication of server (mitigate MITM attacks)
- File transfer

History of SSH

- 1995
  - Tatu Ylonen releases ssh-1.0.0
  - Forms SSH Communications Security
- 1997
  - IETF secsh working group formed
  - Work on SSH protocol v.2 begun
- 1999
  - OpenSSH project started, based on open-source ssh-1.x code
- 2000
  - SSH protocol v.2 support added to OpenSSH
- 2002
  - SSH support added to Solaris 9 (OpenSSH derived)
  - SSH protocol v.2 nears release as IETF RFC

SSH Protocol

- Runs on TCP port 22, initiated by client
- Client and server exchange banners at connect time:
  - SSH-1.5-SoftwareName => SSH protocol v.1
  - SSH-2.0-SoftwareName => SSH protocol v.2
  - SSH-1.99-SoftwareName => both protocols
- SoftwareName is the implementation name and version
  - Usually used for backwards (bug) compatibility
- Server always has a public/private key pair
  - Public key is sent during connection setup
  - Server’s public key is cached by client to detect MITM

SSH Protocol v.1

- SSH protocol v.1 is the original version released in the free ssh-1.x code by Tatu Ylonen
- Revised between 1995 and 1997
- Final version of the protocol is officially "1.5"
- Never standardised
  - Some desire to have it published as an informational RFC
- Monolithic protocol

SSH Protocol v.1 packet format

- 4 bytes Packet length
- 1-8 bytes Padding, length = (8 - length % 8)
- 1 byte Packet type
- ... Payload
- 4 bytes CRC(padding, type, payload)

SSH Protocol v.1 packet format continued

- Packet payload may optionally be compressed
  - Before CRC calculation
- Packets are then encrypted with a symmetric cipher
  - (3)DES
  - Blowfish
  - IDEA (patented, not widely used today)
  - RC4 (deprecated)
  - TSS (deprecated)

SSH Protocol v.1 protocol start (step 1)

- Server and client exchange banners
- Banners include exact protocol version (e.g. "1.5")
- Matching of implementation name in banner is frequently used
  - Workarounds for bugs
  - Switch off features not implemented

SSH Protocol v.1 protocol start (step 2)

- Server sends SMSG_PUBLIC_KEY to client, containing:
  - Server host public key and temporary public key
  - Allowed ciphers and authentication methods
  - 64-bit random nonce

SSH Protocol v.1 protocol start (step 3)

- Client and server both compute 128-bit "session id"
  - MD5 hash of server key and cookie
- No data sent in this step

SSH Protocol v.1 protocol start (step 4)

- Client responds with CMSG_SESSION_KEY, containing:
  - Selected cipher
  - Protocol flags
  - 256-bit random session key
    - Encrypted with both server keys
    - First 128 bits are XORed with session id
- The server should have calculated the same session-id in step 3 as the client, it should be able to decrypt the session key

SSH Protocol v.1 protocol start (step 5)

- Server responds with SMSG_SUCCESS
- Protocol now moves to authentication

SSH Protocol v.2

- Unlike the monolithic v.1 protocol SSH v.2 is several protocols
- Transport protocol
  - Underlying protocol
  - Handles encryption, compression, integrity
  - Provides "services" based on text strings
- User Authentication protocol
  - Responsible for authentication of user to server
  - Supports various authentication methods
    - Password, Public key, Challenge-response, Host based
- Connection protocol
  - Interactive logins, Command execution, Port forwarding, X11 forwarding

SSH Protocol v.2 packet format

- 4 bytes Packet length
- 1 byte Padding length
- ...   Payload
- padding_len Random padding
- ...   MAC[seqno, packet (sans MAC)]

SSH Protocol v.2 packet format

- Multiple MAC algorithms supported
  - hmac-md5, hmac-sha1, truncated MACs, none
- Payload may optionally be compressed prior to MAC
- Packets are optionally encrypted with a symmetric cipher
  - 3-des-cbc (MUST)
  - blowfish-cbc (RECOMMENDED)
  - twofish-cbc, aes-cbc, serpent-cbc (OPTIONAL)
  - arcfour, idea-cbc, cast128-cbc (OPTIONAL)

SSH Protocol v.2 protocol start

- Server and client exchange banners
- Client and server both send MSG_KEXINIT packet
  - Random nonce
  - Supported/Allowed algorithms to use for key exchange
  - Supported/Allowed server host key formats
  - Supported/Allowed symmetric algorithms (both ways)
  - Supported/Allowed MAC algorithms
  - Supported/Allowed compression algorithms
  - Supported/Allowed languages
  - Flag indicating "KEX guess"
- The suppored/allowed lists are comma-separated strings
  - E.g. "aes128-cbc, 3des-cbc, arcfour"

SSH Protocol v.2 protocol start - continued

- Upon receipt of KEXINIT packet, both client and server
  - Calculate intersection of supported/allowed ciphers, etc
  - Run selected key exchange algorithm
    - Usually Diffie Hellman
- D-H key exchange
  - Secure way for two parties to derive a shared key
    - Safe against eavesdroppers
    - Perfect Forward Secrecy
  - Exchange is authenticated with hash of
    - Client & server version strings
    - Payloads of client & server KEXINIT packets
    - Server host key
    - Intermediate D-H exchange values
    - Derived shared secret

SSH Protocol v.2 protocol start - continued

- Output from key exchange is two keys:
  - Server to client
  - Client to server
- The client & server may have different options. e.g.
  - client may send data encrypted with 3des-cbc, no compression
  - server may send data encrypted with arcfour, zlib compression
- MSG_KEXINIT may be sent at any time during session
  - "Re-keying" may change compression, ciphers as well as keys
- Once keyed, protocol moves on to authentication

Major security differences between protocols

- Protocol v.1 CRC is weak integrity mechanism
  - Not collision resistant, easily spoofed
  - Elaborate tricks used to detect attacks
  - Attacks can not be prevented (just detected)
- Protocol v.1 is more susceptible to a MITM attack
  - Only a problem before server key has been learned
  - D-H key exchange in v.2 avoids this when using PubKey auth
- Protocol v.2 is recommended:
  - \+ More secure
  - \+ In IETF standardisation process
  - \+ More extensible
  - \- More per-packet overhead
- OpenSSH defaults to protocol v.2, but still supports v.1

Enough of the theory!