SSH protocol outline

Simon Foley

November 19, 2013

SSH

Runs on top of TCP/IP providing server authentication, strong encryption, integrity protection and zip compression.

Replacement for remote services rlogin, rcp, ftp and may also provide port forwarding for VPN-like tunneling.

Supports multiple authentication mechanisms: simple passwords, public-key based authentication, or plug-in mechanisms (eg SecureID).

SSH2 Transport Layer Protocol Outline

- $C \rightarrow S$: `SSH-<protoversion>-<swversion>`
- $S \rightarrow C$: `SSH-<protoversion>-<swversion>`
- $C \rightarrow S$: `SSH_MSG_KEXINIT` Algorithm negotiation (encryption, MAC,...)
- $S \rightarrow C$: `SSH_MSG_KEXINIT` Algorithm negotiation (encryption, MAC,...)

  [Client and server agree on how to proceed]
- $C \leftrightarrow S$: Authentication and Key exchange.

  [DH-exchange required, others possible]
  [Result from exchange is shared secret K and hash value H]
  [K, H used to derive IV, encryption keys and integrity keys.]

SSH Session Encryption

3DES-CBC, IDEA-CBC,...
Encrypted data in all packets in one direction should be considered a single data stream.
IV's passed from the end of one packet to the beginning of next.
Ciphers in each direction run independently on each other.

SSH Session Integrity

MAC-SHA1, HMAC-MD5, ...
mac = MAC(Key, sequence#||unencrypted packet)
mac in each direction run independently of one another.

SSH DH Key Exchange

K_S: S's public host key; formats: ssh-dss, x509v3, spki, pgp public key.

Fixed parameters (DH): large safe prime p, generator g and order q.
Suitable values for p, g and q defined by internet draft.

V_S, V_C: server/client version string. I_S, I_C: `KEXINIT` strings.

1. [C generates a random $1 < x < q$, computes $e = g^x \bmod p$]
   $C \rightarrow S$: `SSH_MSG_KEXDH_INIT`,e.
2. [S generates a random $1 < y < q$, computes $f = g^y \bmod p$, $K = e^y \bmod p$, $H = hash(V_C||V_S||I_C||I_S||K_S||e||f||K)$, and signs H using K_S^{-1} to give s]
   $S \rightarrow C$: `SSH_MSG_KEXDH_REPLY`,K_S, f, s

SSH DH Key Exchange, continued

3. [C verifies K_S is the host key (certificates or local database), C is also allowed to accept the key without verification; however, doing so will render the protocol insecure against active attacks (but may be desirable for practical reasons in the short term in many environments). C computes $K = f^x$ mod p, $H = hash(V_C||V_S||I_C||I_S||K_S||e||f||K)$, and verifies the signature s on H.]

   $C \rightarrow S$: `SSH_MSG_NEWKEYS`

   $S \rightarrow C$: `SSH_MSG_NEWKEYS`

Results from Key Exchange

If authentication/key exchange successful then both parties share secret K and hash H. H is used to identify session.
Encryption keys computed as:

- Initial IV client to server: $HASH(K||H||\texttt{"A"}||sessionid)$.
- Initial IV server to client: $HASH(K||H||\texttt{"B"}||sessionid)$.
- Encryption key client to server: $HASH(K||H||\texttt{"C"}||sessionid)$.
- Encryption key server to client: $HASH(K||H||\texttt{"D"}||sessionid)$.
- Integrity key client to server: $HASH(K||H||\texttt{"E"}||sessionid)$.
- Integrity key server to client: $HASH(K||H||\texttt{"F"}||sessionid)$.

Keys are regenerated (re-run protocol) after each gigabyte of transmitted data or after each hour of connection time, whichever comes sooner.

A variation of the SSH protocol

Principal A (owning public key K_A) establishes a shared secret K with principal B (owning public keys K_{Bh} and K_{Bs}).

Msg1: $A \rightarrow B : \quad N_A$

Msg2: $B \rightarrow A : \quad N_B$

Msg3: $B \rightarrow A : \quad K_{Bh}, K_{Bs}$

Msg4: $A \rightarrow B : \quad \{\{h(previous\ messages), K\}_{K_{Bs}}\}_{K_{Bh}}$

Msg5: $A \rightarrow B : \quad \{A, K_A, \{h(A, N_A, N_B)\}_{K_A^{-1}}\}_{K'}$

N_A and N_B are nonces, $h(\ldots)$ a one-way hash function and K' is a secret derived from K, N_A and N_B.

We need separate secrets for the channel-MACs in both directions in addition to secrets for channel-confidentiality. For example, define $K_{AB} = h(K', \text{`A'})$, $K_{BA} = h(K', \text{'B'})$, $K_{AB}^{mac} = h(K', \text{'C'})$, $K_{BA}^{mac} = h(K', \text{'D'})$, We use the same strategy to generate the IV's for the cipher, etc.

Persistent and Ephemeral Keys

Public keys K_{Bh} and K_{Bs} correspond to *persistent* and *ephemeral* protocol keys.

The persistent key is a long-term key used to authenticate B; its private key is presumed stored in some protected file/device.

The ephemeral key is a short-term key used for a short period of time and then refreshed; its corresponding private key should not be stored in any permanent way (hard disk, etc).

If the private long term key is compromised then, given that the ephemeral key regularly changes and its private key is not saved anywhere, then the attacker will not be able to decrypt past messages that he may have accumulated.