SSH: Secure Shell

- Readings
  - RFC 4251- RFC 4254
  - Manual page of ssh command
What is SSH?

• **SSH – Secure Shell**
  – Program vs. company vs. protocol
  – Will concentrate on SSH-2 protocol

• **SSH is a protocol for secure remote login and other secure network services over an insecure network**
  – Replacement for telnet, rsh, rlogin, etc

• Developed by SSH Communications, Finland

• Specified in a set of Internet drafts

• **Two distributions are available:**
  – Commercial version
  – Freeware ([www.openssh.com](http://www.openssh.com))
SSH Layers

<table>
<thead>
<tr>
<th>SSH Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>User authentication Protocol</td>
</tr>
<tr>
<td>Transport Layer Protocol</td>
</tr>
<tr>
<td>TCP</td>
</tr>
</tbody>
</table>
Major SSH Components

• **SSH Transport Layer Protocol**
  – Provides server authentication, confidentiality, and integrity services
  – May provide compression too
  – Runs on top of any reliable transport layer (e.g., TCP)

• **SSH User Authentication Protocol**
  – Provides client-side user authentication
  – Runs on top of the SSH Transport Layer Protocol

• **SSH Connection Protocol**
  – Multiplexes multiple logical channels into secure tunnel provided by Transport Layer and User Authentication Protocols
  – Logical channels can be used for a wide range of purposes
    • Secure interactive shell sessions
    • Forwarding X11 connections
    • TCP port forwarding
SSH Security Features

• Strong algorithms
  – Uses well established strong algorithms for encryption, integrity, key exchange, and public key management

• Large key size
  – Requires encryption to be used with at least 128 bit keys
  – Supports larger keys too

• Algorithm negotiation
  – Encryption, integrity, key exchange, and public key algorithms are negotiated
  – It is easy to switch to some other algorithm without modifying the base protocol
SSH Transport Layer Protocol – Overview

TCP connection setup

SSH version string exchange

SSH key exchange
(includes algorithm negotiation)

SSH data exchange

termination of the TCP connection
Key Exchange - Overview

- **Overview**
- **Execution of the selected key exchange protocol**
- **SSH_MSG_KEXINIT**
- **SSH_MSG_NEWKEYS**
- Client uses new keys and algorithms for sending
- Server uses new keys and algorithms for receiving
Diffie-Hellman Key Exchange
(with Explicit Server Authentication)

1. – Client generates a random number \( x \) and computes \( e = g^x \mod p \)
   – Client sends \( e \) to the server

2. – Server generates a random number \( y \) and computes \( f = g^y \mod p \)
   – Server receives \( e \) from the client
   – It computes \( K = e^y \mod p = g^{xy} \mod p \) and \( H = \text{HASH}( \text{client version string} | \text{server version string} | \text{client kex init msg} | \text{server kex init msg} | \text{server host key} K_{\text{srv}} | e | f | K ) \)
   – It generates a signature \( s \) on \( H \) using the private part of the server host key (may involve additional hash computation on \( H \))
   – It sends \( (K_{\text{srv}} | f | s) \) to the client

3. – Client verifies that \( K_{\text{srv}} \) is really the host key of the server
   – Client computes \( K = f^x \mod p = g^{xy} \mod p \) and the exchange hash \( H \)
   – Client verifies the signature \( s \) on \( H \)
Deriving Keys and IVs

• Any key exchange algorithm produces two values
  – a shared secret K
  – an exchange hash value H
• H from the first key exchange is used as the session ID
• Keys and IVs are derived from K and H as follows:
  – IV client to server = HASH( K | H | “A” | session ID )
  – IV server to client = HASH( K | H | “B” | session ID )
  – encryption key client to server = HASH( K | H | “C” | session ID )
  – encryption key server to client = HASH( K | H | “D” | session ID )
  – MAC key client to server = HASH( K | H | “E” | session ID )
  – MAC key server to client = HASH( K | H | “F” | session ID )
Server Authentication

- Based on the server's host key $K_{srv}$
- Client must check $K_{srv}$ is really host key of server
- Models
  - Client has a local database that associates each host name with the corresponding public host key
  - Host name – to – key association is certified by a trusted CA and server provides the necessary certificates or client obtains them from elsewhere
  - Check fingerprint of key over an external channel (e.g., phone)
  - Best effort:
    - accept host key without check when connecting first time to server
    - save the host key in the local database, and
    - check against the saved key on all future connections to the same server
Key Re-Exchange

• It is recommended to change keys after each gigabyte of transmitted data or after each hour of connection time

• key re-exchange is processed identically to the initial key exchange
  – except for the session ID, which will remain unchanged
  – algorithms may be changed
  – keys and IVs are recomputed
Service Request

- After key exchange the client requests a service
- Services
  - ssh-userauth
  - ssh-connection
- When the service starts, it has access to the session ID established during the first key exchange
SSH – User Authentication Protocol

• Protocol assumes that the underlying transport protocol provides integrity and confidentiality (e.g., SSH Transport Layer Protocol)
  – Protocol has access to the session ID
• Three authentication methods are supported
  – publickey
  – password
  – hostbased
The “publickey” Method

- All implementations must support this method
- However, most local policies will not require authentication with this method in the near future, as users don’t have public keys
- Authentication is based on demonstration of the knowledge of the private key (the client signs with the private key)
- Server verifies that
  - the public key really belongs to the user specified in the authentication request
  - the signature is correct
The “password” Method

• All implementations should support this method
  – User account
  – password

• This method is likely the most widely used
The “hostbased” Method

- Authentication is based on the host where the user is coming from
  - This method is optional
- Client sends a signature that has been generated with the private host key of the client
- Server verifies that
  - The public key really belongs to the host specified in the authentication request
  - The signature is correct
Hostbased: Try the Following

• To access or run command on remote machine without typing password.

• Remote ssh from machine A to machine B

  Step 1: at machine A: `ssh-keygen -t rsa`
  (do not enter any pass phrase, just keep typing “enter”)
  Step 2: append A:`.ssh/id_rsa.pub` to B:`.ssh/authorized_keys`

• After these steps, (without typing password)
  – You should be able to access remote machine
    • On machine A: `ssh user@B`
  – you should be able to run remote command.
    • On machine A: `ssh user@B “command”`

• We do not recommend this
  – Breaking into one machine, breaking into all machines
SSH – Connection Protocol

• Provides
  – interactive login sessions
  – remote execution of commands
  – forwarded TCP/IP connections
  – forwarded X11 connections

• All these applications are implemented as “channels”

• All channels are multiplexed into the single encrypted tunnel provided by the SSH Transport Layer Protocol

• Channels are identified by channel numbers at both ends of the connection
  – Channel numbers for the same channel at the client and server sides may differ
SSH Port Forwarding or Tunneling

- Frequently as an alternative to a full-fledged VPN
  - A (non-secure) TCP/IP connection of an external application is redirected to the SSH program (client or server)
    - Forwards it to the other SSH party (server or client)
    - In turn forwards the connection to the desired destination host

- Forwarded connection is encrypted and protected on the path between the SSH client and server only

- Primarily useful for tunneling connections through firewalls
  - Ordinarily block that type of connection
  - Encrypting protocols which are not normally encrypted (e.g. VNC)
TCP/IP Port Forwarding Example

• **Real server on remote machine**
  – I want to listen on port 5110 on this machine; all packets arriving here get sent to mailserver, port 110:
  – ssh -L 5110:mailserver:110 mailserver

• **Real server on this machine**
  – All web traffic to my firewall should be redirected to the web server running on port 8000 on my machine instead:
  – ssh -R 80:MyMachine:8000 firewall
X Windows Forwarding

• No setup – already done!
• Run the X Windows application in the terminal window:
  – xclock &
  – The screen display shows up on your computer, and any keystrokes and mouse movements are sent back, all encrypted.
SSL/TLS vs. SSH

- Developed around the same time (mid 90s)
- SSH Transport Layer Protocol roughly equivalent to SSL/TLS
  - SSH could have been implemented using SSL/TLS
- They do have different origins and targeted applications
- SSL/TLS, developed by Netscape for web application
  - Authenticating server is critical
- SSH targets to replace plaintext remote login
  - Authenticating both server and client is critical
Reading Assignment

• Reviewing for final exam