Network Working Group   T. Ylonen
Internet-Draft   SSH Communications Security Corp
Expires: September 15, 2005   C. Lonvick, Ed.
                                                      Cisco Systems, Inc.
                                                         March 14, 2005

SSH Connection Protocol
draft-ietf-secsh-connect-25.txt

Status of this Memo

Copyright Notice

Abstract

SSH is a protocol for secure remote login and other secure network
services over an insecure network.

This document describes the SSH Connection Protocol. It provides
interactive login sessions, remote execution of commands, forwarded

TCP/IP connections, and forwarded X11 connections. All of these
channels are multiplexed into a single encrypted tunnel.

The SSH Connection Protocol has been designed to run on top of the
SSH transport layer and user authentication protocols.

Table of Contents

1. Contributors

The major original contributors of this set of documents have been:
Tatu Ylonen, Tero Kivinen, Timo J. Rinne, Sami Lehtinen (all of SSH
Communications Security Corp), and Markku-Juhani O. Saarinen
(University of Jyvaskyla). Darren Moffit was the original editor of
this set of documents and also made very substantial contributions.

Many people contributed to the development of this document over the
years. People who should be acknowledged include Mats Andersson, Ben
Harris, Brent McClure, Niels Moller, Damien Miller, Derek Fawcus,
Frank Cusack, Heikki Nousiainen, Jakob Schlyter, Jeff Van Dyke,
Jeffrey Altman, Jeffrey Hutzelman, Jon Bright, Joseph Galbraith, Ken
Hornstein, Markus Friedl, Martin Forssen, Nicolas Williams, Niels
Provos, Perry Metzger, Peter Gutmann, Simon Josefsson, Simon Tatham,
Wei Dai, Denis Bider, der Mouse, and Tadayoshi Kohno. Listing their
names here does not mean that they endorse this document, but that
they have contributed to it.

2. Introduction

The SSH Connection Protocol has been designed to run on top of the
SSH transport layer and user authentication protocols. It provides
interactive login sessions, remote execution of commands, forwarded
TCP/IP connections, and forwarded X11 connections. The service name
for this protocol is "ssh-connection".

This document should be read only after reading the SSH architecture
document [SSH-ARCH]. This document freely uses terminology and
notation from the architecture document without reference or further
explanation.

3. Conventions Used in This Document

All documents related to the SSH protocols shall use the keywords
"MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
"SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" to describe
requirements. These keywords are to be interpreted as described in
[RFC2119].

The keywords "PRIVATE USE", "HIERARCHICAL ALLOCATION", "FIRST COME
FIRST SERVED", "EXPERT REVIEW", "SPECIFICATION REQUIRED", "IESG
APPROVAL", "IETF CONSENSUS", and "STANDARDS ACTION" that appear in
this document when used to describe namespace allocation are to be
interpreted as described in [RFC2434].

Protocol fields and possible values to fill them are defined in this
set of documents. Protocol fields will be defined in the message

definitions. As an example, SSH_MSG_CHANNEL_DATA is defined as
follows.

```
    byte      SSH_MSG_CHANNEL_DATA
    uint32    recipient channel
    string    data
```

Throughout these documents, when the fields are referenced, they will
appear within single quotes. When values to fill those fields are
referenced, they will appear within double quotes. Using the above
example, possible values for 'data' are "foo" and "bar".

[4]. Global Requests

There are several kinds of requests that affect the state of the
remote end globally, independent of any channels. An example is a
request to start TCP/IP forwarding for a specific port. Note that
both client and server MAY send global requests at any time, and the
receiver MUST respond appropriately. All such requests use the
following format.

```
        byte      SSH_MSG_GLOBAL_REQUEST
        string    request name in US-ASCII only
        boolean   want reply
        ... request-specific data follows
```

The value of 'request name' follows the DNS extensibility naming
convention outlined in [SSH-ARCH].

The recipient will respond to this message with
SSH_MSG_REQUEST_SUCCESS or SSH_MSG_REQUEST_FAILURE if 'want reply' is
TRUE.

```
        byte      SSH_MSG_REQUEST_SUCCESS
        .....     response specific data
```

Usually the 'response specific data' is non-existent.

If the recipient does not recognize or support the request, it simply
responds with SSH_MSG_REQUEST_FAILURE.

```
        byte      SSH_MSG_REQUEST_FAILURE
```

In general, the reply messages do not include request type
identifiers. To make it possible for the originator of a request to
identify to which request each reply refers, it is REQUIRED that
replies to SSH_MSG_GLOBAL_REQUESTS MUST be sent in the same order as
the corresponding request messages. For channel requests, replies

that relate to the same channel MUST also be replied to in the right
order. However, channel requests for distinct channels MAY be
replied to out-of-order.

5. Channel Mechanism

All terminal sessions, forwarded connections, etc., are channels.
Either side may open a channel. Multiple channels are multiplexed
into a single connection.

Channels are identified by numbers at each end. The number referring
to a channel may be different on each side. Requests to open a
channel contain the sender's channel number. Any other
channel-related messages contain the recipient's channel number for
the channel.

Channels are flow-controlled. No data may be sent to a channel until
a message is received to indicate that window space is available.

5.1 Opening a Channel

When either side wishes to open a new channel, it allocates a local
number for the channel. It then sends the following message to the
other side, and includes the local channel number and initial window
size in the message.

```
        byte      SSH_MSG_CHANNEL_OPEN
        string    channel type in US-ASCII only
        uint32    sender channel
        uint32    initial window size
        uint32    maximum packet size
        ... channel type specific data follows
```

The 'channel type' is a name as described in [SSH-ARCH] and
[SSH-NUMBERS], with similar extension mechanisms. The 'sender
channel' is a local identifier for the channel used by the sender of
this message. The 'initial window size' specifies how many bytes of
channel data can be sent to the sender of this message without
adjusting the window. The 'maximum packet size' specifies the
maximum size of an individual data packet that can be sent to the
sender. For example: one might want to use smaller packets for
interactive connections to get better interactive response on slow
links.

The remote side then decides whether it can open the channel, and
responds with either SSH_MSG_CHANNEL_OPEN_CONFIRMATION or
SSH_MSG_CHANNEL_OPEN_FAILURE.

```
   byte      SSH_MSG_CHANNEL_OPEN_CONFIRMATION
   uint32    recipient channel
   uint32    sender channel
   uint32    initial window size
   uint32    maximum packet size
   ...  'channel type' specific data follows
```

The 'recipient channel' is the channel number given in the original
open request, and 'sender channel' is the channel number allocated by
the other side.

```
   byte      SSH_MSG_CHANNEL_OPEN_FAILURE
   uint32    recipient channel
   uint32    reason code
   string    description in ISO-10646 UTF-8 encoding [RFC3629]
   string    language tag as defined in [RFC3066]
```

If the recipient of the SSH_MSG_CHANNEL_OPEN message does not support
the specified 'channel type', it simply responds with
SSH_MSG_CHANNEL_OPEN_FAILURE. The client MAY show the 'description'
string to the user. If this is done, the client software should take
the precautions discussed in [SSH-ARCH].

The SSH_MSG_CHANNEL_OPEN_FAILURE 'reason code' values are defined in
the following table. Note that the values for the 'reason code' are
given in decimal format for readability but that they are actually
uint32 values.

```
        Symbolic name                          reason code
        -------------                          -----------
      SSH_OPEN_ADMINISTRATIVELY_PROHIBITED          1
      SSH_OPEN_CONNECT_FAILED                       2
      SSH_OPEN_UNKNOWN_CHANNEL_TYPE                 3
      SSH_OPEN_RESOURCE_SHORTAGE                    4
```

Requests for assignments of new SSH_MSG_CHANNEL_OPEN 'reason code'
values (and associated 'description' text) in the range of 0x00000005
to 0xFDFFFFFF MUST be done through the IETF CONSENSUS method as
described in [RFC2434]. The IANA will not assign Channel Connection
Failure 'reason code' values in the range of 0xFE000000 to
0xFFFFFFFF. Channel Connection Failure 'reason code' values in that
range are left for PRIVATE USE as described in [RFC2434].

While it is understood that the IANA will have no control over the
range of 0xFE000000 to 0xFFFFFFFF, this range will be split in two
parts and administered by the following conventions.

o The range of 0xFE000000 to 0xFEFFFFFF is to be used in conjunction
   with locally assigned channels. For example: if a channel is
   proposed with a 'channel type' of "example_session@example.com"
   but fails, then the response will contain either a 'reason code'
   assigned by the IANA (as listed above and in the range of
   0x00000001 to 0xFDFFFFFF), or with a locally assigned value in the
   range of 0xFE000000 to 0xFEFFFFFF. Naturally, if the server does
   not understand the proposed 'channel type', even if it is a
   locally defined 'channel type', then the 'reason code' MUST be
   0x00000003 as described above, if the 'reason code' is sent. If
   the server does understand the 'channel type' but the channel
   still fails to open, then the server SHOULD respond with a locally
   assigned 'reason code' value consistent with the proposed, local
   'channel type'. It is assumed that practitioners will first
   attempt to use the IANA assigned 'reason code' values and then
   document their locally assigned 'reason code' values.

o There are no restrictions or suggestions for the range starting
   with 0xFF. No interoperability is expected for anything used in
   this range. Essentially it is for experimentation.

5.2 Data Transfer

The window size specifies how many bytes the other party can send
before it must wait for the window to be adjusted. Both parties use
the following message to adjust the window.

        byte   SSH_MSG_CHANNEL_WINDOW_ADJUST
        uint32   recipient channel
        uint32   bytes to add

After receiving this message, the recipient MAY send the given number
of bytes more than it was previously allowed to send; the window size
is incremented. Implementations MUST correctly handle window sizes
of up to 2^32 - 1 bytes. The window MUST NOT be increased above 2^32
- 1 bytes.

Data transfer is done with messages of the following type.

        byte   SSH_MSG_CHANNEL_DATA
        uint32   recipient channel
        string   data

The maximum amount of data allowed is the current window size. The
window size is decremented by the amount of data sent. Both parties
MAY ignore all extra data sent after the allowed window is empty.

Additionally, some channels can transfer several types of data. An
example of this is stderr data from interactive sessions. Such data
can be passed with SSH_MSG_CHANNEL_EXTENDED_DATA messages, where a
separate integer specifies the type of the data. The available types
and their interpretation depend on the type of the channel.

```
        byte      SSH_MSG_CHANNEL_EXTENDED_DATA
        uint32    recipient_channel
        uint32    data_type_code
        string    data
```

Data sent with these messages consumes the same window as ordinary
data.

Currently, only the following type is defined. Note that the value
for the 'data_type_code' is given in decimal format for readability
but that the values are actually uint32 values.

```
        Symbolic name                   data_type_code
        -------------                   --------------
      SSH_EXTENDED_DATA_STDERR                1
```

Extended Channel Data Transfer 'data_type_code' values MUST be
assigned sequentially. Requests for assignments of new Extended
Channel Data Transfer 'data_type_code' values, and their associated
Extended Channel Data Transfer 'data' strings, in the range of
0x00000002 to 0xFDFFFFFF MUST be done through the IETF CONSENSUS
method as described in [RFC2434]. The IANA will not assign Extended
Channel Data Transfer 'data_type_code' values in the range of
0xFE000000 to 0xFFFFFFFF. Extended Channel Data Transfer
'data_type_code' values in that range are left for PRIVATE USE as
described in [RFC2434]. As is noted, the actual instructions to the
IANA are in [SSH-NUMBERS].

5.3 Closing a Channel

When a party will no longer send more data to a channel, it SHOULD
send SSH_MSG_CHANNEL_EOF.

```
        byte      SSH_MSG_CHANNEL_EOF
        uint32    recipient_channel
```

No explicit response is sent to this message. However, the
application may send EOF to whatever is at the other end of the

channel. Note that the channel remains open after this message, and
more data may still be sent in the other direction. This message
does not consume window space and can be sent even if no window space
is available.

When either party wishes to terminate the channel, it sends
SSH_MSG_CHANNEL_CLOSE. Upon receiving this message, a party MUST
send back a SSH_MSG_CHANNEL_CLOSE unless it has already sent this
message for the channel. The channel is considered closed for a
party when it has both sent and received SSH_MSG_CHANNEL_CLOSE, and
the party may then reuse the channel number. A party MAY send
SSH_MSG_CHANNEL_CLOSE without having sent or received
SSH_MSG_CHANNEL_EOF.

          byte   SSH_MSG_CHANNEL_CLOSE
          uint32   recipient_channel

This message does not consume window space and can be sent even if no
window space is available.

It is recommended that any data sent before this message is delivered
to the actual destination, if possible.

5.4 Channel-Specific Requests

Many 'channel type' values have extensions that are specific to that
particular 'channel type'. An example is requesting a pty (pseudo
terminal) for an interactive session.

All channel-specific requests use the following format.

          byte   SSH_MSG_CHANNEL_REQUEST
          uint32   recipient channel
          string   request type in US-ASCII characters only
          boolean   want reply
          ... type-specific data

If 'want reply' is FALSE, no response will be sent to the request.
Otherwise, the recipient responds with either SSH_MSG_CHANNEL_SUCCESS
or SSH_MSG_CHANNEL_FAILURE, or request-specific continuation
messages. If the request is not recognized or is not supported for
the channel, SSH_MSG_CHANNEL_FAILURE is returned.

This message does not consume window space and can be sent even if no
window space is available. The values of 'request type' are local to
each channel type.

The client is allowed to send further messages without waiting for
the response to the request.

'request type' names follow the DNS extensibility naming convention
outlined in [SSH-ARCH] and [SSH-NUMBERS].

```
        byte      SSH_MSG_CHANNEL_SUCCESS
        uint32    recipient_channel


        byte      SSH_MSG_CHANNEL_FAILURE
        uint32    recipient_channel
```

These messages do not consume window space and can be sent even if no
window space is available.

6. Interactive Sessions

A session is a remote execution of a program. The program may be a
shell, an application, a system command, or some built-in subsystem.
It may or may not have a tty, and may or may not involve X11
forwarding. Multiple sessions can be active simultaneously.

6.1 Opening a Session

A session is started by sending the following message.

```
        byte      SSH_MSG_CHANNEL_OPEN
        string    "session"
        uint32    sender channel
        uint32    initial window size
        uint32    maximum packet size
```

Client implementations SHOULD reject any session channel open
requests to make it more difficult for a corrupt server to attack the
client.

6.2 Requesting a Pseudo-Terminal

A pseudo-terminal can be allocated for the session by sending the
following message.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient_channel
        string    "pty-req"
        boolean   want_reply
        string    TERM environment variable value (e.g., vt100)
        uint32    terminal width, characters (e.g., 80)
        uint32    terminal height, rows (e.g., 24)
        uint32    terminal width, pixels (e.g., 640)
        uint32    terminal height, pixels (e.g., 480)
        string    encoded terminal modes
```

The 'encoded terminal modes' are described in Section 8. Zero
dimension parameters MUST be ignored. The character/row dimensions
override the pixel dimensions (when nonzero). Pixel dimensions refer
to the drawable area of the window.

The dimension parameters are only informational.

The client SHOULD ignore pty requests.

6.3 X11 Forwarding

6.3.1 Requesting X11 Forwarding

X11 forwarding may be requested for a session by sending a
SSH_MSG_CHANNEL_REQUEST message.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient channel
        string    "x11-req"
        boolean   want reply
        boolean   single connection
        string    x11 authentication protocol
        string    x11 authentication cookie
        uint32    x11 screen number
```

It is RECOMMENDED that the 'x11 authentication cookie' that is sent
be a fake, random cookie, and that the cookie is checked and replaced
by the real cookie when a connection request is received.

X11 connection forwarding should stop when the session channel is
closed. However, already opened forwardings should not be
automatically closed when the session channel is closed.

If 'single connection' is TRUE, only a single connection should be
forwarded. No more connections will be forwarded after the first, or
after the session channel has been closed.

The 'x11 authentication protocol' is the name of the X11
authentication method used, e.g., "MIT-MAGIC-COOKIE-1".

The 'x11 authentication cookie' MUST be hexadecimal encoded.

The X Protocol is documented in [SCHEIFLER].

6.3.2 X11 Channels

X11 channels are opened with a channel open request. The resulting
channels are independent of the session, and closing the session
channel does not close the forwarded X11 channels.

```
        byte      SSH_MSG_CHANNEL_OPEN
        string    "x11"
        uint32    sender channel
        uint32    initial window size
        uint32    maximum packet size
        string    originator address (e.g., "192.168.7.38")
        uint32    originator port
```

The recipient should respond with SSH_MSG_CHANNEL_OPEN_CONFIRMATION
or SSH_MSG_CHANNEL_OPEN_FAILURE.

Implementations MUST reject any X11 channel open requests if they
have not requested X11 forwarding.

6.4 Environment Variable Passing

Environment variables may be passed to the shell/command to be
started later. Uncontrolled setting of environment variables in a
privileged process can be a security hazard. It is recommended that
implementations either maintain a list of allowable variable names or
only set environment variables after the server process has dropped
sufficient privileges.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient channel
        string    "env"
        boolean   want reply
        string    variable name
        string    variable value
```

6.5 Starting a Shell or a Command

   Once the session has been set up, a program is started at the remote
   end. The program can be a shell, an application program or a
   subsystem with a host-independent name. Only one of these requests
   can succeed per channel.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient channel
        string    "shell"
        boolean   want reply
```

   This message will request the user's default shell (typically defined
   in /etc/passwd in UNIX systems) to be started at the other end.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient channel
        string    "exec"
        boolean   want reply
        string    command
```

   This message will request the server to start the execution of the
   given command. The 'command' string may contain a path. Normal
   precautions MUST be taken to prevent the execution of unauthorized
   commands.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient channel
        string    "subsystem"
        boolean   want reply
        string    subsystem name
```

   This last form executes a predefined subsystem. It is expected that
   these will include a general file transfer mechanism, and possibly
   other features. Implementations may also allow configuring more such
   mechanisms. As the user's shell is usually used to execute the
   subsystem, it is advisable for the subsystem protocol to have a
   "magic cookie" at the beginning of the protocol transaction to
   distinguish it from arbitrary output generated by shell
   initialization scripts, etc. This spurious output from the shell may
   be filtered out either at the server or at the client.

   The server SHOULD NOT halt the execution of the protocol stack when
   starting a shell or a program. All input and output from these
   SHOULD be redirected to the channel or to the encrypted tunnel.

   It is RECOMMENDED to request and check the reply for these messages.
   The client SHOULD ignore these messages.

   Subsystem names follow the DNS extensibility naming convention
   outlined in [SSH-NUMBERS].

6.6 Session Data Transfer

   Data transfer for a session is done using SSH_MSG_CHANNEL_DATA and
   SSH_MSG_CHANNEL_EXTENDED_DATA packets and the window mechanism. The
   extended data type SSH_EXTENDED_DATA_STDERR has been defined for
   stderr data.

6.7 Window Dimension Change Message

   When the window (terminal) size changes on the client side, it MAY
   send a message to the other side to inform it of the new dimensions.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient_channel
        string    "window-change"
        boolean   FALSE
        uint32    terminal width, columns
        uint32    terminal height, rows
        uint32    terminal width, pixels
        uint32    terminal height, pixels
```

    A response SHOULD NOT be sent to this message.

6.8 Local Flow Control

   On many systems, it is possible to determine if a pseudo-terminal is
   using control-S/control-Q flow control. When flow control is
   allowed, it is often desirable to do the flow control at the client
   end to speed up responses to user requests. This is facilitated by
   the following notification. Initially, the server is responsible for
   flow control. (Here, again, client means the side originating the
   session, and server means the other side.)

   The message below is used by the server to inform the client when it
   can or cannot perform flow control (control-S/control-Q processing).
   If 'client can do' is TRUE, the client is allowed to do flow control
   using control-S and control-Q. The client MAY ignore this message.

```
        byte      SSH_MSG_CHANNEL_REQUEST
        uint32    recipient channel
        string    "xon-xoff"
        boolean   FALSE
        boolean   client can do
```

   No response is sent to this message.

6.9 **Signals**

   A signal can be delivered to the remote process/service using the
   following message. Some systems may not implement signals, in which
   case they SHOULD ignore this message.

            byte   SSH_MSG_CHANNEL_REQUEST
            uint32   recipient channel
            string   "signal"
            boolean   FALSE
            string   signal name without the "SIG" prefix.

   Signal names will be encoded as discussed in the "exit-signal"
   SSH_MSG_CHANNEL_REQUEST.

6.10 **Returning Exit Status**

   When the command running at the other end terminates, the following
   message can be sent to return the exit status of the command.
   Returning the status is RECOMMENDED. No acknowledgment is sent for
   this message. The channel needs to be closed with
   SSH_MSG_CHANNEL_CLOSE after this message.

   The client MAY ignore these messages.

            byte   SSH_MSG_CHANNEL_REQUEST
            uint32   recipient_channel
            string   "exit-status"
            boolean   FALSE
            uint32   exit_status

   The remote command may also terminate violently due to a signal.
   Such a condition can be indicated by the following message. A zero
   'exit_status' usually means that the command terminated successfully.
   o byte   SSH_MSG_CHANNEL_REQUEST
   o uint32   recipient channel
   o string   "exit-signal"
   o boolean   FALSE
   o string   signal name without the "SIG" prefix.
   o boolean   core dumped
   o string   error message in ISO-10646 UTF-8 encoding
   o string   language tag as defined in [RFC3066]

The 'signal name' is one of the following (these are from [POSIX])

                ABRT
                ALRM
                FPE
                HUP
                ILL
                INT
                KILL
                PIPE
                QUIT
                SEGV
                TERM
                USR1
                USR2

   Additional 'signal name' values MAY be sent in the format
   "sig-name@xyz", where "sig-name" and "xyz" may be anything a
   particular implementor wants (except the "@" sign). However, it is
   suggested that if a 'configure' script is used, any non-standard
   'signal name' values it finds be encoded as "SIG@xyz.config.guess",
   where "SIG" is the 'signal name' without the "SIG" prefix, and "xyz"
   be the host type, as determined by "config.guess".

   The 'error message' contains an additional textual explanation of the
   error message. The message may consist of multiple lines. The
   client software MAY display this message to the user. If this is
   done, the client software should take the precautions discussed in
   [SSH-ARCH].

7. TCP/IP Port Forwarding

7.1 Requesting Port Forwarding

   A party need not explicitly request forwardings from its own end to
   the other direction. However, if it wishes that connections to a
   port on the other side be forwarded to the local side, it must
   explicitly request this.

                byte   SSH_MSG_GLOBAL_REQUEST
                string   "tcpip-forward"
                boolean   want reply
                string   address to bind (e.g., "0.0.0.0")
                uint32   port number to bind

   The 'address to bind' and 'port number to bind' specify the IP
   address or domain name and port to which the socket to be listened is

bound. Some strings used for the 'address to bind' have special-case
semantics.

o "" means that connections are to be accepted on all protocol
   families supported by the SSH implementation.

o "0.0.0.0" means to listen on all IPv4 addresses.

o "::" means to listen on all IPv6 addresses.

o "localhost" means to listen on all protocol families supported by
   the SSH implementation on loopback addresses only, [RFC3330] and
   [RFC3513].

o "127.0.0.1" and "::1" indicate listening on the loopback
   interfaces for IPv4 and IPv6 respectively.

Note that the client can still filter connections based on
information passed in the open request.

Implementations should only allow forwarding privileged ports if the
user has been authenticated as a privileged user.

Client implementations SHOULD reject these messages; they are
normally only sent by the client.

If a client passes 0 as port number to bind and has 'want reply' TRUE
then the server allocates the next available unprivileged port number
and replies with the following message, otherwise there is no
response specific data.

        byte   SSH_MSG_REQUEST_SUCCESS
        uint32   port that was bound on the server

A port forwarding can be canceled with the following message. Note
that channel open requests may be received until a reply to this
message is received.

        byte   SSH_MSG_GLOBAL_REQUEST
        string   "cancel-tcpip-forward"
        boolean   want reply
        string   address_to_bind (e.g., "127.0.0.1")
        uint32   port number to bind

Client implementations SHOULD reject these messages; they are
normally only sent by the client.

[7.2](#) **TCP/IP Forwarding Channels**

   When a connection comes to a port for which remote forwarding has
   been requested, a channel is opened to forward the port to the other
   side.

```
         byte      SSH_MSG_CHANNEL_OPEN
         string    "forwarded-tcpip"
         uint32    sender channel
         uint32    initial window size
         uint32    maximum packet size
         string    address that was connected
         uint32    port that was connected
         string    originator IP address
         uint32    originator port
```

   Implementations MUST reject these messages unless they have
   previously requested a remote TCP/IP port forwarding with the given
   port number.

   When a connection comes to a locally forwarded TCP/IP port, the
   following packet is sent to the other side. Note that these messages
   MAY be sent also for ports for which no forwarding has been
   explicitly requested. The receiving side must decide whether to
   allow the forwarding.

```
         byte      SSH_MSG_CHANNEL_OPEN
         string    "direct-tcpip"
         uint32    sender channel
         uint32    initial window size
         uint32    maximum packet size
         string    host to connect
         uint32    port to connect
         string    originator IP address
         uint32    originator port
```

   The 'host to connect' and 'port to connect' specify the TCP/IP host
   and port where the recipient should connect the channel. The 'host
   to connect' may be either a domain name or a numeric IP address.

   The 'originator IP address' is the numeric IP address of the machine
   where the connection request comes from, and the 'originator port' is
   the port on the originator host from where the connection originated.

   Forwarded TCP/IP channels are independent of any sessions, and
   closing a session channel does not in any way imply that forwarded
   connections should be closed.

Client implementations SHOULD reject direct TCP/IP open requests for
security reasons.

[8](). Encoding of Terminal Modes

All "encoded terminal modes" (as passed in a pty request) are encoded
into a byte stream. It is intended that the coding be portable
across different environments.

The tty mode description is a stream of bytes. The stream consists
of opcode-argument pairs where the opcode is a byte value. It is
terminated by opcode TTY_OP_END (0x00). Opcodes 1 to 159 have a
single uint32 argument. Opcodes 160 to 255 are not yet defined, and
cause parsing to stop (they should only be used after any other
data).

The client SHOULD put in the stream any modes it knows about, and the
server MAY ignore any modes it does not know about. This allows some
degree of machine-independence, at least between systems that use a
POSIX-like tty interface. The protocol can support other systems as
well, but the client may need to fill reasonable values for a number
of parameters so the server pty gets set to a reasonable mode (the
server leaves all unspecified mode bits in their default values, and
only some combinations make sense).

The naming of opcode values mostly follows the POSIX terminal mode
flags. The following opcode values have been defined. Note that the
values given below are in decimal format for readability but that
they are actually byte values.

```
opcode   argument        description
------   --------        -----------
  0      TTY_OP_END   Indicates end of options.
  1      VINTR        Interrupt character; 255 if none.  Similarly
                       for the other characters. Not all of these
                       characters are supported on all systems.
  2      VQUIT        The quit character (sends SIGQUIT signal on
                       POSIX systems).
  3      VERASE       Erase the character to left of the cursor.
  4      VKILL        Kill the current input line.
  5      VEOF         End-of-file character (sends EOF from the
                       terminal).
  6      VEOL         End-of-line character in addition to
                       carriage return and/or linefeed.
  7      VEOL2        Additional end-of-line character.
  8      VSTART       Continues paused output (normally
                       control-Q).
  9      VSTOP        Pauses output (normally control-S).
```

```
10    VSUSP      Suspends the current program.
11    VDSUSP     Another suspend character.
12    VREPRINT   Reprints the current input line.
13    VWERASE    Erases a word left of cursor.
14    VLNEXT     Enter the next character typed literally,
                  even if it is a special character
15    VFLUSH     Character to flush output.
16    VSWTCH     Switch to a different shell layer.
17    VSTATUS    Prints system status line (load, command,
                  pid, etc).
18    VDISCARD   Toggles the flushing of terminal output.
30    IGNPAR     The ignore parity flag.  The parameter
                  SHOULD be 0 if this flag is FALSE,
                  and 1 if it is TRUE.
31    PARMRK     Mark parity and framing errors.
32    INPCK      Enable checking of parity errors.
33    ISTRIP     Strip 8th bit off characters.
34    INLCR      Map NL into CR on input.
35    IGNCR      Ignore CR on input.
36    ICRNL      Map CR to NL on input.
37    IUCLC      Translate uppercase characters to
                  lowercase.
38    IXON       Enable output flow control.
39    IXANY      Any char will restart after stop.
40    IXOFF      Enable input flow control.
41    IMAXBEL    Ring bell on input queue full.
50    ISIG       Enable signals INTR, QUIT, [D]SUSP.
51    ICANON     Canonicalize input lines.
52    XCASE      Enable input and output of uppercase
                  characters by preceding their lowercase
                  equivalents with "\".
53    ECHO       Enable echoing.
54    ECHOE      Visually erase chars.
55    ECHOK      Kill character discards current line.
56    ECHONL     Echo NL even if ECHO is off.
57    NOFLSH     Don't flush after interrupt.
58    TOSTOP     Stop background jobs from output.
59    IEXTEN     Enable extensions.
60    ECHOCTL    Echo control characters as ^(Char).
61    ECHOKE     Visual erase for line kill.
62    PENDIN     Retype pending input.
70    OPOST      Enable output processing.
71    OLCUC      Convert lowercase to uppercase.
72    ONLCR      Map NL to CR-NL.
73    OCRNL      Translate carriage return to newline
                  (output).
74    ONOCR      Translate newline to carriage
                  return-newline (output).
```

```
      75   ONLRET       Newline performs a carriage return
                         (output).
      90   CS7          7 bit mode.
      91   CS8          8 bit mode.
      92   PARENB       Parity enable.
      93   PARODD       Odd parity, else even.

     128 TTY_OP_ISPEED  Specifies the input baud rate in
                          bits per second.
     129 TTY_OP_OSPEED  Specifies the output baud rate in
                          bits per second.
```

9. Summary of Message Numbers

   The following is a summary of messages and their associated message
   number.

```
        SSH_MSG_GLOBAL_REQUEST                 80
        SSH_MSG_REQUEST_SUCCESS                81
        SSH_MSG_REQUEST_FAILURE                82
        SSH_MSG_CHANNEL_OPEN                   90
        SSH_MSG_CHANNEL_OPEN_CONFIRMATION      91
        SSH_MSG_CHANNEL_OPEN_FAILURE           92
        SSH_MSG_CHANNEL_WINDOW_ADJUST          93
        SSH_MSG_CHANNEL_DATA                   94
        SSH_MSG_CHANNEL_EXTENDED_DATA          95
        SSH_MSG_CHANNEL_EOF                    96
        SSH_MSG_CHANNEL_CLOSE                  97
        SSH_MSG_CHANNEL_REQUEST                98
        SSH_MSG_CHANNEL_SUCCESS                99
        SSH_MSG_CHANNEL_FAILURE               100
```

10. IANA Considerations

   This document is part of a set. The IANA considerations for the SSH
   protocol as defined in [SSH-ARCH], [SSH-TRANS], [SSH-USERAUTH], and
   this document, are detailed in [SSH-NUMBERS].

11. Security Considerations

   This protocol is assumed to run on top of a secure, authenticated
   transport. User authentication and protection against network-level
   attacks are assumed to be provided by the underlying protocols.

   Full security considerations for this protocol are provided in

[SSH-ARCH]. Specific to this document, it is RECOMMENDED that
   implementations disable all the potentially dangerous features (e.g.,
   agent forwarding, X11 forwarding, and TCP/IP forwarding) if the host
   key has changed without notice or explanation.

12. References

12.1 Normative References

   [SSH-ARCH]
               Lonvick, C., "SSH Protocol Architecture",
               I-D draft-ietf-secsh-architecture-22.txt, March 2005.

   [SSH-TRANS]
               Lonvick, C., "SSH Transport Layer Protocol",
               I-D draft-ietf-secsh-transport-24.txt, March 2005.

   [SSH-USERAUTH]
               Lonvick, C., "SSH Authentication Protocol",
               I-D draft-ietf-secsh-userauth-27.txt, March 2005.

   [SSH-NUMBERS]
               Lonvick, C., "SSH Protocol Assigned Numbers",
               I-D draft-ietf-secsh-assignednumbers-12.txt, March 2005.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
               Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC2434]   Narten, T. and H. Alvestrand, "Guidelines for Writing an
               IANA Considerations Section in RFCs", BCP 26, RFC 2434,
               October 1998.

   [RFC3066]   Alvestrand, H., "Tags for the Identification of
               Languages", BCP 47, RFC 3066, January 2001.

   [RFC3629]   Yergeau, F., "UTF-8, a transformation format of ISO
               10646", STD 63, RFC 3629, November 2003.

12.2 Informative References

   [RFC1884]   Hinden, R. and S. Deering, "IP Version 6 Addressing
               Architecture", RFC 1884, December 1995.

   [RFC3330]   IANA, "Special-Use IPv4 Addresses", RFC 3330, September
               2002.

   [RFC3513]   Hinden, R. and S. Deering, "Internet Protocol Version 6
               (IPv6) Addressing Architecture", RFC 3513, April 2003.

   [SCHEIFLER]
               Scheifler, R., "X Window System : The Complete Reference
               to Xlib, X Protocol, Icccm, Xlfd, 3rd edition.", Digital
               Press ISBN 1555580882, February 1992.

   [POSIX]   ISO/IEC, 9945-1., "Information technology -- Portable
               Operating System Interface (POSIX)-Part 1: System
               Application Program Interface (API) C Language",
               ANSI/IEE Std 1003.1, July 1996.

Authors' Addresses

   Tatu Ylonen
   SSH Communications Security Corp
   Fredrikinkatu 42
   HELSINKI FIN-00100
   Finland

   Email: ylo@ssh.com

   Chris Lonvick (editor)
   Cisco Systems, Inc.
   12515 Research Blvd.
   Austin 78759
   USA

   Email: clonvick@cisco.com

[Appendix A](#). Trademark Notice

   "ssh" is a registered trademark in the United States and/or other
   countries.

   Note to the RFC Editor: This should be a separate section like the
   subsequent ones, and not an appendix. This paragraph to be removed
   before publication.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment