Ssh

Mechiel Lukkien

mechiel@xs4all.nl

ABSTRACT

*Ssh** is a client for the *secure shell 2* protocol, written in Limbo for Inferno. It also includes *sftpfs*, an *sftp* client that translates between sftp and styx/9p messages. They currently support most of the popular key exchange, authentication, encryption and digesting methods used by the *ssh* protocol, but is not yet ready for daily use. No code for a server has been written yet.

Introduction

The *ssh* project aims to bring secure shell 2 support to Inferno. It has both a secure shell client for executing a shell or other commands on an ssh server and an sftp client that gives access to files on the ssh server. No code for an ssh server or a program like Plan 9's *sshnet*(4) for using the server's TCP stack has been written. At the time of writing, the basics work, i.e. logging in to an ssh server; basic sftp works too. Most of the standard key exchange, authentication and encryption and digest methods are supported: *Diffie-Hellman* key exchange with *rsa* and *dss* host verification; password and *rsa* and *dsa* public-key authentication; *sha-1* and *md5* for digests; and *des-cbc*, *3des-cbc*, *aes* (with 128, 192 or 256 bit keys, in *cbc* or *ctr* mode), *arcfour* (with 128 or 256 bit keys) and *idea* (though untested because no one uses it). New *factotum*(4) support handles authentication for the public-key methods. The two major limitations are:

- Channel windows are not updated. Both client and server maintain a window for each communication channel on the ssh connection. The window indicates how many bytes the other side is willing to consume, thus how many bytes can be written without blocking. A large initial window is set by the current code, but the window is never updated during the connection. This works for most types of communication, but will block when all data from the initial window has been consumed.
- Session key renegotiation has not yet been implemented. During connection set up, encryption and digest keys are exchanged between the client and server (and in the process the client verifies the server's *host key*). These are used to encrypt and sign the protocol packets. The protocol specification dictates that keys be renegotiated after they have been used for some time or for a certain number of encryption operations. We currently never do that. Thus, when the server wants new keys, the connection will break.

The protocol

Some details about the protocol. The protocol is specified in *RFCs* 4250-4254. Later *RFCs* clarify, extend and/or deprecate functionality. The protocol is logically layered, with a *transport*, *user authentication* and *connection* 'layer'. These layers are more like phases of the connection. Protocol messages are not actually layered or nested, all have

* *Ssh*, `http://www.ueber.net/code/r/ssh`

the same packet format with one set of simple encoding rules. *Sftp* is not part of the secure shell protocol. It can be used outside of ssh too, though this is uncommon. It can be implemented as a separate program that speaks the sftp protocol over a channel by provided by an ssh connection. Sftp is only described in expired *work in progress RFC* drafts The most recent versions of those drafts should be ignored: they are not commonly implemented and only add complexities such as *ACL* schemes.

Ssh provides secure communication between two systems. The server accepts incoming connections and plays the role of the server. The connection goes through various stages: Key exchange (including host verification), user authentication and finally normal operation during which communication channels can be created and data sent and received on those channels. An ssh client creates a channel and requests a remote service, typically a login shell. The server starts this service and essentially connects the channel's communication descriptors to the shell's standard input, output and error. The client does a similar thing. The protocol allows many channels to be opened on a single ssh connection. For example for multiple shells, or a shell and an sftp connection.

The sftp service can be requested on a channel, with the same mechanism used to start a shell. The sftp service reads sftp protocol requests from the channel and writes the sftp responses to the channel. Sftp maps surprisingly directly to styx, but (as most such protocols) cannot do all of a *wstat* operation atomically, needing multiple sftp requests. *Sftpfs* does not wait for an sftp response before sending the next sftp request, so has some accounting to do (e.g. for flushes, and the two-stage *wstat*).

Additional services have been specified: X11 forwarding, authentication agent forwarding. No support for those is planned.

Future work

The two missing bits of important functionality have been explained earlier. Many more small ones exist and it is likely that large chunks of code need to be rewritten. The design might need to change, as a consequence of how it was developed: I wanted to get some useful packets exchanged with a server as soon as possible, so I dialed an ssh server (running *OpenSSH*) and saw it sent a banner. Finding how to respond to that banner was easy, ssh packets followed soon. By the time encryption was needed, generic packet parsing and packing code was usable. In the mean time I had realised I could enable debugging output (including protocol message printing and diagnostics) on the *OpenSSH* server. It would tell me if packets were malformed, unexpected, etc. So the *OpenSSH* server has been a great help during development. This approach resulted in quick initial results, and the protocol was learned along the way. It did not result in very clean code though, but that will be fixed.

Both the ssh client and sftp client need lots of polishing. At some point a terminal emulator for Inferno would be useful, to be able to use *curses* programs on unix systems.

Other missing features:

- Ssh version 1 is not supported. It is being phased out on the internet, few people still use it and every new ssh server deployment supports ssh version 2 and often refuses to speak ssh version 1 because it is less secure. Support for version 1 will probably not be implemented.
- The reasonably popular *blowfish* encryption algorithm is not supported yet. It seems there are various versions of *blowfish* in use, with different endianness for data and/or keys. Newer key exchange methods that use *SHA-2* are also not supported: Inferno does not yet have a *SHA-2* library. Both should be fixed eventually, though there is no hurry.
- The ssh protocol supports compression of the data packets with the *deflate* algorithm. *Ssh* will not support it any time soon because Inferno's *deflate* library does

not support flushing the compressed buffer on command, which is required.

- There is currently no ssh server, or sshnet-like program. Both are useful and may be implemented in the future.
- Host key fingerprints, used to verify that the host is who it claims it is, are stored in a file, `$home/lib/sshkeys`. Since this is security sensitive information, use and management of these keys should perhaps be handled by a *factotum*-like program.
- Inferno's factotum currently always prompts the user for credentials when a key was requested but none found. *Ssh* tries *rsa* and *dss* keys first, then normal passwords. In the quite common case of wanting to authenticate by password, this results in two unwanted factotum key requests.