ENTERPRISE OPERATIONS MANAGEMENT

THE SSH PROTOCOL

Duncan Napier

INSIDE

A Brief History of the Secure Shell Protocol and Its Implementations; How SSH Works; Installation; Configuration; Using SSH

INTRODUCTION

SSH (Secure Shell) is a protocol for running secure network services over an insecure network. The protocol serves as the basis for many implementations of SSH that are now widely available as commercial or noncommercial products. These products encompass a wide variety of platforms, including virtually all flavors of UNIX, DOS/Windows, and Macintosh operating systems as well as many other environments, including OpenVMS, BeOS, OS/2, PalmOS, and Java to name a few. SSH runs on top of TCP/IP and is generally invisible to the software applications layer. As a result, SSH can be made completely transparent to end users and does not require any additional user training.

The term "Secure Shell" originates from the early days of SSH in 1995, when Tatu Ylönen, a researcher at Helsinki University of Technology, wrote an application to facilitate secure, encrypted login access for UNIX hosts. SSH was originally designed as a secure drop-in replacement for rsh, UNIX remote shell, as well as remote login and file transfer applications such as telnet and rcp. These traditional UNIX services either bypass or offer limited user authentication (i.e., logon and password). They are vulnerable to IP spoofing or DNS table manipulation because they offer neither the means to authenticate the identity of hosts being logged on to, nor that of the login client. They also pass all data — including login names and passwords — in plaintext over connections that can be silently hijacked. Plaintext data can be easily modified or corrupted in transit without the knowledge of end users.

Conceptually, the SSH protocol runs as an application on top of the TCP/IP layer. The SSH protocol is implemented through separate client and server applications that authenticate and negotiate protocols before

PAYOFF IDEA

SSH (Secure Shell) is a protocol for authenticating, encrypting, and checking the integrity of information traversing TCP/IP-based networks. This article describes SSH, how to install it, and how to use it.

Auerbach Publications

deciding whether to establish a secure connection. The protocol provides for cryptographic host and user authentication, strong encryption, integrity protection, and the simultaneous tunneling of multiple data channels. These features lend themselves to more than just secure remote logins. The feature set of SSH includes:

- Secure remote login
- Secure remote command execution
- Secure remote file transfer
- TCP port forwarding
- Cryptographic key control
- Authentication agents (including single sign-on)
- Configurable access control
- Data compression

Before discussing the above features in detail, it is useful to summarize the history of the SSH protocol to understand some of the rationale behind its design and its numerous implementations.

A BRIEF HISTORY OF THE SECURE SHELL PROTOCOL AND ITS IMPLEMENTATIONS

The first incarnation of the SSH protocol (known henceforth as SSH1) was developed by Tatu Ylönen in 1995. SSH1 was released on the Internet as free software with source code in July 1995. Ylönen documented the software as an Internet Engineering Task Force (IETF) Internet Draft that became the specification for the SSH1 protocol.

By the end of 1995, there were an estimated 20,000 users and Ylönen started SSH Communications Security (http://www.ssh.com) to commercially sell, support, and develop SSH. The freeware versions of SSH continued to be available, but SSH Communications Security imposed restrictions on the terms of their use. In 1996, SSH Communications Security introduced a new version of the protocol, SSH2. The IETF founded a public working group for standardization of the secure shell, SECSH, in 1996. By February 1997, the first Internet Draft for the SSH2 protocol was completed. In 1998, SSH Communications Security released an SSH2 implementation based on the SSH2 protocol.

The SSH2 protocol fixed some problems and shortcomings in SSH1 but these fixes rendered SSH2 incompatible with SSH1. SSH Communications Security also placed more restrictions of the use of its SSH2 product. These limitations may explain why after all this time, SSH1 is still probably the more widely used protocol at the time of writing. SSH Communications Security has since removed some of the restrictions on its SSH2 product and there has also been a steady proliferation of free and Open Source implementations that support both SSH1 and SSH2 protocols. An indication that SSH is entering the public mainstream is the SSH functionality offered with many of the staple terminal emulation packages, both

free (e.g., TeraTerm) and commercial (e.g., Van Dyke Secure CRT and April Systems Anita).

The many free, Open Source, and commercial implementations of the SSH1 and SSH2 protocols make it difficult to talk about the practical properties and features of SSH in general without biasing the discussion in the direction of a particular implementation. While virtually all implementations are faithful to the fundamental features of the protocol and can be made to interoperate smoothly, different implementations may have enhancements or idiosyncrasies that are unique to a particular product. In addition to the obvious incompatibility between SSH1 and SSH2 protocols, the author has found subtle and not-so-subtle problems, often relating to the interoperability of current products with many older releases.

To alleviate the problem in describing a fictitious "generic" implementation of the SSH protocols, the focus here is on the OpenSSH implementation of SSH. Open SSH is based on Björn Grönvall's fix of Ylönen's last free version of SSH1 (release 1.2.12), which Grönvall named OSSH. By early 2000, the OpenBSD (http://www.openbsd.org/) team had taken over OSSH and renamed the project OpenSSH using Grönvall's work. OpenSSH has reached its current form through the work of Markus Friedl and others.

The decision to use OpenSSH in this discussion is based on a number of reasons, namely that OpenSSH:

- Has been ported to a wide variety of platforms encompassing all the major UNIX flavors, Windows/DOS, MAC OS X, and others
- Is free and has no patented algorithms in its source tree
- Supports both SSH1 and SSH2 in a single, seamless package
- Derives its code base from the OpenBSD project, which has an almost unparalleled track record in the development of secure, stable, and reliable software
- Releases tightly controlled upgrades on a regular and timely basis

OpenSSH is a work in progress and its feature list continues to grow with each successive release.

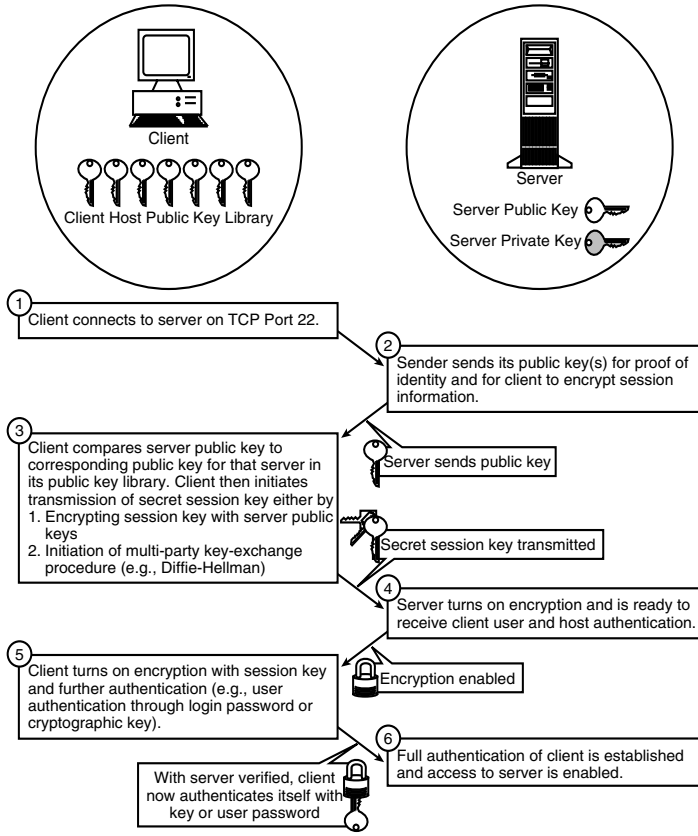
HOW SSH WORKS

Exhibit 1 shows a conceptual schema of how SSH authentication and encryption works. The actual details of the implementation may vary, depending on the version of the protocol (SSH1 or SSH2), the specific implementation involved, and the user's choice of authentication protocols. Essentially, the steps are as shown in Exhibit 1 and describe below.

Step 1

The client host (left) connects to the server host (right), conventionally on TCP port 22. The server and client exchange the protocol versions

EXHIBIT 1 — Steps through which SSH Initiates, Authenticates, and Encrypts Communications



they support and, if compatible, continue the connection. Otherwise, the connection is terminated. The connection may be terminated if SSH1-only and SSH2-only implementations are involved because the two protocols are incompatible. The connection at this stage is unencrypted but uses check-bytes for integrity checking and plaintext-attack prevention on top of the TCP connection to ensure that the connection is not attacked or hijacked at this stage.

Step 2

The server sends its authentication information and session parameters to the client. This includes the server host's public key component of its own public/private key pair, as well as a list of the encryption, compression, and authentication modes that the server supports. In SSH1, an ad-

ditional server key is sent. Public key algorithms supported by most SSH implementations include RSA (Rivest-Shamir-Aldeman) and DSA (Digital Signature Algorithm).

Step 3

The client host checks the server host's public key against the client host's library of public keys. If this is the first time that this particular client and this particular server host have connected over SSH, the user is asked to verify the addition of new a server host public key to the client's public key library. Any future connections to that particular server host will now be verified against that public key reported from their first contact. Once the identity of the server is verified, a secret session key is generated.

In SSH1, the session key is encrypted with the server host public key and the server host's server public keys and sent back to the server host. Because the encrypted session key can only be decrypted by the server host's private key, the secret session key can be safely transmitted over the insecure link.

In SSH2, a multi-party key-exchange algorithm is used. Key-exchange algorithms allow for a shared secret to be agreed to and then securely transmitted between parties. The original and perhaps best-known key-exchange algorithm is the Diffie-Hellman algorithm.

One might wonder why a secret session key is required when public/private key pairs are available. The reason is that while many encryption methods use public key (or asymmetric) encryption, public key encryption is much slower than symmetric encryption methods in which all parties share a single, shared secret key. As a result, the secret key is transmitted and kept secret using public key encryption, but the actual scrambling of the data is done more speedily with the shared secret key called the session key. Public keys also just happen to be ideal for authentication and identification purposes. Secret key algorithms that SSH supports include 3DES (triple Data Encryption Standard), IDEA (International Data Encryption Algorithm), and Blowfish.

Note that at this point, the communication is still unencrypted.

Step 4

Once the secret session key is in the possession of both parties, encryption and integrity checking are turned on.

SSH1 uses a single session key for each session. For improved security, SSH2 can periodically change session keys, a process known as "re-keying." Session keys are typically stored only in memory and are not written to storage for security purposes. SSH1 uses the weak CRC-32 (Cyclic Redundancy Check) method for checking data integrity. SSH2 uses cryptographically stronger MAC (Method Authentication Code) integrity checkers.