

The Risks of Using SSH

Information Systems Security

ELE 1741

Fall 1999

Professor Hinton

December 5, 1999.

David Tam

tamda@ecf.utoronto.ca

Executive Summary

This paper deals with security issues surrounding the use of Secure Shell (SSH). SSH is a replacement for telnet, rlogin, ftp, rsh, rcp, rdist, and other r*-based programs. It offers a secure communication channel between computers on an insecure network. Authenticity, confidentiality, and integrity are provided. Despite these features, SSH has several weaknesses that render it insecure under certain conditions. This paper examines these circumstances and weaknesses, proposes solutions, and suggests conditions which lead to effective use of SSH.

The existence of insecure access methods bypasses the security offered by SSH. Public host keys might not undergo authentication, leading to a risk in man-in-the-middle attacks. The potential dependency on NFS (Sun Microsystems' Network File System) leads to the lack of confidentiality, integrity, and authenticity in the access control files. The tunnelling capability of SSH poses a risk to firewalls. In general, SSH will not provide security under any of the following situations: (1) dumb-terminals or X-terminals are used; (2) NFS is used; (3) public host keys cannot be certified as authentic. Users should not use SSH and should not revert to insecure remote access methods under the following conditions: (1) the public key of the remote host cannot be proven authentic; (2) the local or remote host uses NFS; (3) there exists an insecure segment along the communication path; (4) the local or remote host is not trusted.

Problem Definition

Introduction

Traditional remote access programs such as telnet, ftp, rlogin, rsh, rcp, rdist, and other r*-based programs transmit all data, including passwords, across networks in clear text. While this method of transmission was acceptable prior to the world wide web phenomenon, it now creates a serious security risk. The growing popularity of the Internet among ordinary people and hackers, and the growing need to access remote systems presents a sea of opportunity for password sniffing.

Secure Shell (SSH) was designed as a replacement for these traditional remote access programs. It offers a secure communication channel across insecure networks. All data, including passwords, are sent in encrypted form. SSH provides confidentiality, integrity, and authenticity. Confidentiality is provided by symmetric key encryption. Integrity is provided by message authentication codes (MACs). Authenticity is provided by public key algorithms.

Despite these features, use of SSH alone is not a sufficient condition for achieving secure communications. SSH has several weaknesses that render it insecure under certain conditions. This paper will examine these circumstances and weaknesses. Solutions will be proposed and suggestions will be provided as to when usage of SSH will improve security. The mentality of "total paranoia" will be adapted. A partially secure system is inadequate. A system is either secure or insecure.

SSH is presently available in two versions. SSH1, released in 1996, provides a basic set of encryption and authentication methods. SSH2, currently under IETF review, includes additional methods and optionally supports authentication using OpenPGP, X.509, and SPKI certificates.

Problems

SSH is not widely supported when compared to the traditional remote access programs. Consequently, mobile users who do not have access to SSH must either revert to the traditional insecure methods or forfeit connectivity. Using security terminology, this lack of access can be seen as a problem in availability. If the insecure methods are used, security is compromised and all the benefits of SSH are lost.

In user authentication, SSH provides backwards compatibility with r*-based programs by supporting `.rhosts` and `/etc/hosts.equiv` configuration files. Providing this feature encourages the use of traditional insecure means of connection. Naturally, systems which remain configured in this manner are at risk of traditional r*-based attacks. Kerberos is also supported as a method for user authentication although this system is known to have its own set of security problems.

In remote host authentication, SSH1 uses the RSA public/private key method. The default configuration permits users to accept new public keys of remote hosts without authentication through certificates. Unfortunately, users who choose to accept these public keys are vulnerable to man-in-the-middle attacks. To prevent such an attack, system administrators are responsible for managing the public keys of commonly used hosts. SSH2 addresses this shortcoming by optionally supporting various certificate formats.

SSH relies on configuration and key files to determine access rights. Systems that use Sun Microsystems' Network File System (NFS) to access these files pose a major security risk. Since the NFS specification is widely available and packets are transmitted across the local area network (LAN) in clear text, hackers can easily employ NFS sniffers to obtain secret keys, alter public keys, and add public keys.

Similar problems are present in systems that employ dumb-terminals and X-

terminals on a LAN. On these terminals, all processing occurs on other computers located across the network so the flow of unencrypted data (especially passwords) can be intercepted. Therefore, SSH is rendered insecure on these terminals.

User errors can lead to security breaches because they may not be aware that security is compromised if an insecure channel is traversed anywhere along the communication path. For instance, a user who first telnets to a computer located on the LAN before using SSH to access a remote host will allow hackers to monitor the insecure portion of the path. Such an error is very easy to overlook by the average user and cannot be detected and prevented by SSH.

Since there are numerous security breaches reported and numerous patches issued for SSH [3], [5], [12], [13], system administrators have the tedious task of updating and verifying the security of their system. Due to human nature, system administrators may fail to follow this rapid pace of change. Ignorance may lead to situations similar to the buffer overflow problem where some systems remain unpatched long after a patch has been issued. The original SSH implementation and subsequent patches must be obtained using a secure channel. These packages must be signed by a reputable authority since there is the possibility of obtaining corrupt software. Once a patch is installed, system administrators face the difficult task of verifying that a breach did not occur before the installation.

To the horror of system administrators, SSH allows tunnelling which can be used to subvert firewalls and breach security policies. It creates a large hole in the firewall that can lead to security breaches in a surprisingly different manner. Hackers can target SSH as a means of penetrating firewalls and attacking internal computers.

Proposed Solutions

All traditional remote access programs, which includes the corresponding

daemons and clients, should be removed from the system. Such action will prevent most attempts to use insecure means of communication. Although it may be adequate to remove only the server components (daemons), removing the client components will prevent security breaches on other remote systems.

Strict public host key checking must be enforced. This option is specifiable by the system administrator. New host public keys should never be accepted at face value. If SSH1 is used, connections that present new host public keys should be disallowed unless they can be verified over a secure channel such as through telephone or courier mail. If SSH2 is used, new public host keys should be verified using OpenPGP, X.509, or SPKI certificates. For mobile users of SSH1, public keys of the local system should be stored on a write-protected floppy disk. When away from the local system, the public key can be supplied from the write-protected floppy. Users must still trust the system they are using to access the network. With SSH2, the use of certificates also requires a policy for checking certificate revocation lists.

Since the use of NFS is possible, configuration files and key files should be stored and retrieved in an encrypted form. Currently, only the user private key entry is stored in encrypted form in the best scenario. Even with this precaution, the private key file is prone to integrity attacks since only the individual entry is encrypted. The most secure solution involves encrypting and signing all files to ensure confidentiality, integrity, and authenticity while traversing an insecure LAN via NFS. Unfortunately, this solution cannot be implemented by the system administrator alone since it requires alterations to the SSH protocol to ensure complete security.

SSH should not be allowed on dumb-terminals or X-terminals unless communication to the corresponding compute servers is encrypted. Such a policy may create an inequality gap between workstation and terminal users.

Education must be provided to prevent users from introducing an insecure channel along the communication path to the remote host. It is nearly impossible

for SSH to detect whether all segments of the communication path are insecure since SSH may be used on only a portion of the path. Such as task lies outside the jurisdiction of SSH and may also lie outside the jurisdiction of the local system. For instance, while a mobile employee is on a business trip, he /she initially telnets to a gateway and then uses SSH to access the company network.

To restrict tunnelling, the SSH protocol must be altered to enable monitoring of tunnel entry and exit points. Monitoring would allow policy enforcement, denying certain ports from being tunnelled in or out of the LAN. Since this option is currently not supported, tunnelling remains a serious security risk. The only option remaining is for system administrators to configure SSH with tunnelling disabled, which may be too restrictive where access to X11 is required.

All user private keys should be stored in encrypted form to minimize damage caused by breaches in host security. This option is available in SSH but is not mandatory. The SSH implementation must be altered to enforce this restriction. Under these precautions, a hacker who has gained access to a regular user account would be unable to read the user's private key. The passphrase, which is chosen by the user to encrypt his / her private key, should be checked for adequate strength. As well, the security policy should specify that passphrases must never be stored on any medium other than in the user's head.

Both local and remote hosts must be trusted in order to use SSH. Under SSH1, the local system must possess the authentic public key of the remote system. Even under SSH2, where certificates are used to authenticate remote host public keys, the local system must be trusted to contain the genuine public key of the CA or the trusted PGP key. Unfortunately, these judgments cannot be made by system administrators and are left in the hands of users. For instance, mobile employees must determine whether a host can be trusted before using its SSH facilities to access the corporate network.

Discussion

Since SSH belongs to the first generation of secure replacement software, it will receive a lot of resistance and criticism from legacy systems. To ease the transition, SSH allows the traditional methods to coexist. This generosity should not be taken for granted. System administrators should develop a plan to gradually eliminate the usage of traditional methods. They should adopt the conservative view that a system has successfully implemented SSH only when the traditional methods are completely eliminated.

There are a number of differences between SSH1 and SSH2. While SSH1 uses single-sided secret key generation, SSH2 uses the Diffie-Hellman key exchange algorithm which offers more equality. Both parties contribute equally in forming a shared secret that is eventually used to derive a secret symmetric key. SSH1 and SSH2 protocols are implemented differently and are incompatible. SSH1 assumes prior knowledge of public keys of all remote hosts that will be contacted. This assumption may lead to a situation similar to the /etc/hosts saga that resulted in the creation of the Domain Naming System (DNS). System administrators can only manage a limited number of public keys before the task becomes impractical. The applicability of SSH1 is therefore reduced since system administrator will rarely meet this prerequisite. Fortunately, SSH2 has addressed this shortcoming by supporting various certificates, however, it does not enforce certificate usage.

SSH assumes that both local and remote hosts are secure and implements a solution on top of this infrastructure. However, as explained previously, this case is usually not true. Most systems utilize network services that are provided in an insecure fashion. This situation drastically reduces the applicability of SSH to many systems. The designers of SSH could have taken a slightly more conservative view by reducing the assumptions and compensating accordingly. For instance, as mentioned in the previous section, configuration and key files should be stored in an encrypted and signed format.

Public host key revocation is an unresolved problem. Neither infrastructure nor procedures have been defined in SSH to handle key revocation. Periodic

checking of revocation lists is not specified. The SSH specification should include mechanisms to allow for rapid revocation on trusted hosts over a secure channel. Currently, system administrators must use their own discretion when handling revocation.

Isolation is a principle that should be incorporated into the SSH protocol design. If a local or remote host's security is breached, the effects must be automatically contained. If the root account is compromised, this goal is difficult to achieve. However, if a regular user account is compromised, isolation may be possible. Usage of passphrases provides a last line of defense in these situations. Specifying some kind of poison-pill infrastructure may be useful in preventing further damage.

Given the numerous SSH security incidents and patches that are appearing on the Internet, the task of keeping up to date is tedious. There are a number of issues concerning the design of SSH that must be addressed. Perhaps the SSH design and implementation is not simple enough to verify its correctness. Perhaps too many options are supported, which leads to security holes. The SSH2 protocol proposal indicates that flexibility is a key feature. Its goal is to allow different public key algorithms, key distribution methods, message authentication algorithms, and symmetric key algorithms to be supported. Perhaps its specification needs to be smaller and simpler. Convenience and security appear to have an inverse relationship. SSH favours the convenience end of the spectrum.

For mobile users, a Java-based SSH client is currently being developed. If these users do not have access to SSH, they can use a web browser to run a Java-based version of SSH. However, this applet must be obtained securely and the system in which they are accessing must be trusted. This product offers an adequate solution to availability and backwards compatibility.

In the future, developments such as the Internet Protocol Security Protocol (IPSec) may render SSH obsolete since they operate at a lower level in the protocol stack and offer transparent security.

Conclusion

System administrators should adhere to the following guidelines in determining whether SSH will improve security on their system. SSH cannot improve security on systems that contain dumb-terminals or X-terminals connected to the LAN. Any usage from these terminals will create an insecure segment along the communication path. SSH cannot improve security on systems that make use of NFS. SSH cannot improve security if the public keys of all commonly used hosts cannot be authenticated.

Users should adhere to the following guidelines in determining when usage of SSH is appropriate. If a public host key cannot be proven to be authentic, SSH should not be used to communicate with the corresponding remote host. SSH should not be used if the local or remote host makes use of NFS. SSH should not be used if traditional remote access methods are used anywhere along the communication path. Finally, SSH should not be used if the user does not trust the local host or remote host. If usage of SSH is deemed inappropriate, access to the remote system is not possible and users should not revert to the traditional insecure methods.

From the above restrictions, the current SSH specification has only limited real-world applicability. The major barriers are public host key authentication and NFS restrictions. Authenticating all public host keys is currently impractical since most systems use the older SSH1 standard. Since NFS is implemented on most systems, the final set of applicable systems is fairly small.

Even if the problems presented in this paper are resolved, it is only a matter of time before hackers discover new vulnerabilities. SSH must continue improving and system administrators must treat this critical service seriously by keeping their systems updated. Security is a race between hackers and system administrators. Therefore, evaluating the security of a solution involves

determining how far one party is ahead of the other.

Annotated Bibliography

[1] serves as a good introduction to the field of information systems security. An introduction to SSH can be found in [4]. Many frequently asked questions are answered and there are many links to useful resources. [11] and [14] serve as good user guides to SSH and help in understanding the user interface. SSH appraisal can be found in [6], and [8]. Recent developments on a current implementation of SSH2 can be found in [13]. Security vulnerabilities and patches can be found in [3], [5], and [12]. The current status of the SSH protocol specification, under review by The Internet Engineering Task Force, can be found in [10]. This reference contains links to detailed descriptions of the protocol architecture, transport layer protocol, authentication protocol, and connection protocols. Inadequacies in the SSH authentication protocol are explained in [2]. The Diffie-Hellman key exchange algorithm is explained in [9]. An introduction to PGP can be found in [7]. The current status of IPSec can be found in [15].

Bibliography

- [1] E.G. Amoroso, "Fundamentals of Computer Security Technology," Prentice Hall PTR, Upper Saddle River, New Jersey, 1994.
- [2] M. Abadi, "Explicit Communication Revisited: Two New Attacks on Authentication Protocols", IEEE Transactions on Software Engineering, vol. 23, no. 3, pp. 185-186, Mar. 1997.
- [3] J. Barlow, "SSH Patch Repository," Feb 11, 1999.
http://www.ncsa.uiuc.edu/General/CC/ssh/patch_repository/
- [4] A. Carasik and S. Acheson, "The Secure Shell (SSH) Frequently Asked Questions," rev. 1.1, Nov. 2, 1999. <http://www.employees.org/~satch/ssh/faq/>
- [5] "CERT Advisory CA-98.03," Secure Networks Inc, Mar. 2, 1998.
ftp://info.cert.org/pub/cert_advisories/CA-98.03.ssh-agent
- [6] "Curing remote-access security ailments. ssh, the secure shell, can create a moderately secure network connection," SunWorld, Jan. 1996.
<http://www.sunworld.com/swol-01-1996/swol-01-sysadmin.html>
- [7] A. Engelfriet, "The comp.security.pgp FAQ," ver. 1.5, Oct. 22, 1998.
<http://www.pgp.net/pgpnet/pgp-faq/>
- [8] P. Galvin, "Enter the secure shell. Turn remote login from security hole to security strength with ssh," SunWorld, Feb. 1998.
<http://www.sunworld.com/sunworldonline/swol-02-1998/swol-02-security.html>
- [9] D. Harris, "Diffie-Hellman Key Exchange," Aug. 31, 1998.
<http://spectral.mscs.mu.edu/NetworksClass/DHKeyExch.html>
- [10] P. Metzger (chair), "Secure Shell Charter," The Internet Engineering Task Force, Jun. 4, 1999.
<http://www.ietf.org/html.charters/secsh-charter.html>
- [11] T. O'Boyle, J. Sergeant, and J.S. May, "The Secure Shell," Mar. 6, 1998.
<http://csociety.ecn.purdue.edu/~sigos/projects/ssh/>
- [12] A. Polyakov, "SSH and beyond," Aug 14, 1998.
http://fy.chalmers.se/~appro/ssh_beyond.html
- [13] SSH Communications Security Web Page, Dec. 5, 1999.
<http://www.ssh.net/>
- [14] K. Suominen, "Getting Started With SSH," ver. 1.5, Feb. 24, 1998.
<http://www.tac.nyc.ny.us/~kim/ssh/>

- [15] T. Ts'o (chair), "IP Security Protocol (ipsec) Charter," The Internet Engineering Task Force, Nov. 15, 1999.
<http://www.ietf.cnri.reston.va.us/html.charters/ipsec-charter.html>