Secure SSH: Background & Risks

Greg Kent
Vice President, SecureIT

SECURE IT

Secure SSH: Background & Risks

Introduction

This eBook provides an introduction to Secure Shell's (SSH) key-based authentication mechanism and the security and auditing challenges created by SSH's default configuration which places key generation and management in the hands of users. In future eBooks, we will discuss risk management for SSH implementations, including key discovery, establishing a key association registry, and centralizing key provisioning. We will also cover the risks inherent in specific types of key associations that must be considered when creating new or validating existing associations and provides suggestions for mitigating these risks. This series of eBooks provides a comprehensive overview of managing risks associated with SSH.

What is SSH?

The SSH network protocol is widely deployed to provide secured connectivity between systems. SSH provides a secure alternative to telnet or ftp services, which transmit data in clear text and may expose user credentials and sensitive information to eavesdroppers. SSH provides an encrypted tunnel through which users can enter commands, transfer files, or even use an X Windows graphical user interface. For many years, auditors have been advocating wide deployment of SSH as a cost-effective solution to the security problem of clear text network transports. OpenSSH is the most commonly deployed implementation of the SSH protocol. The price is right—it's free—and it does not require the complexities of a Public Key Infrastructure (PKI) for generating keys. However, many organizations that have large OpenSSH deployments have found that SSH can introduce new security problems that can be as significant as the problem of clear text transmissions.

Secure SSH: Background & Risks

SSH provides a wide-range of authentication mechanisms: password authentication, host-based authentication (which is generally avoided), and authentication via public and private key pairs for users. The user key-based authentication option played a big part in the wide adoption of SSH. Although some of the support for SSH keys was driven by a sincere belief that keys provide greater security than traditional passwords, it was the ease-of-use argument that helped make the switch to SSH palatable to system administrators and other users. Users would not have to remember passwords anymore, but could jump from machine to machine, using keys—what could be easier than that? Although keys helped make it easier to sell SSH, the uncontrolled deployment of SSH keys, especially in large organizations, has been the cause of significant security problems.

Since keys are used for authentication, good security practices require that organizations definitively know who can use the keys to connect to accounts. Questions like “who does this key belong to?” or “who can use this key to connect to that account?” are critical, but often very difficult to answer. Establishing management controls over SSH keys was too often neglected when organizations began deploying keys. As a result, many organizations with legacy SSH deployments with thousands of keys and millions of associations spread across hundreds of servers face a huge security problem that exposes them to a significant amount of risk. In order to properly secure and manage SSH, traditional controls need to be applied to key associations, including centralized provisioning and de-provisioning, periodic recertification, and centralized tracking of user entitlements. These reasonable steps can help organizations manage and control the deployment of OpenSSH keys and key associations.

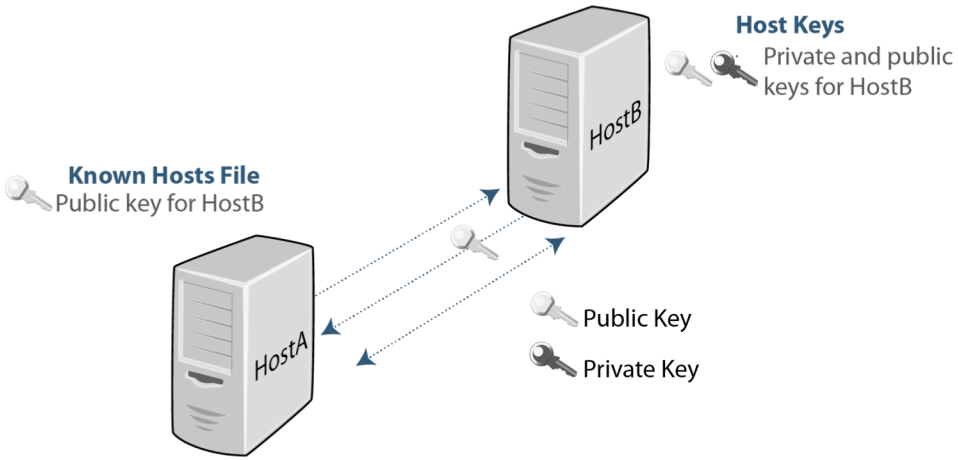
Background on SSH Keys and Associations

SSH uses asymmetric keys, that is, a public-private key pair, to authenticate users. There is a mathematical relationship between the private key and the public key. In SSH, public keys are published on remote systems to indicate users that are allowed to connect. Private keys are controlled by, and should only be accessible to, the key owner. When a user attempts to connect, he provides a private key. SSH authenticates users by verifying the relationship between the published authorized public key and the private key provided by the user.

Secure SSH: Background & Risks

OpenSSH was designed to provide ease-of-use for organizations that do not have an enterprise PKI capability. In most instances when public/private key pairs are widely used in an organization, PKI provides the mechanism for issuing, tracking, managing, renewing, and suspending keys. However, establishing a PKI capability can be quite challenging and expensive. As an alternative, OpenSSH supports a self-service model for key deployment, which makes it easy to adopt and implement. SSH provides a key generating utility that can produce key pairs as needed. In other words, the default implementation of OpenSSH allows users to create their own keys and key associations. This self-serve key management model is the root cause of the problem. Any security process that is self-serve can quickly degrade into an uncontrolled and unmanaged mess, which is precisely what many organizations have painfully experienced with SSH keys.

When considering key management with SSH, it is important to maintain the distinction between the following two components of a key association:



- First, there is a key identity, which identifies the user to whom the key belongs. This consists of a private key that is created for a user. (The public key is stored as part of the private key and can be extracted/generated from the private key on demand.) In terms of an SSH connection, the key identity is related to the source account on the source system, which we will call “From Account” and “From System” in the rest of this article, respectively. Conceptually, the key identity represents the key’s owner. By default, the key identity resides within the user’s home directory, so the key’s owner is the owner of the home directory.

- The second key management component is called an authorized key. One or more authorized keys define the public keys that are authorized to connect to a system as a particular user. The authorized key defines an association (or trust relationship) via the public key(s). Any user that submits a private key associated with an authorized public key is allowed to logon. In terms of an SSH connection, an authorized key is related to the destination account on the destination system, or as we will call them, the “To Account” and “To System”, respectively.

Secure SSH: Background & Risks

The following table summarizes the elements of a key association:

Key Components	Purpose	Default Location	Associated SSH Configuration Element
Key Identity	A private key for a “From Account” on a “From System” that is generally located in the home directory of the “From Account”	~/.ssh/id_rsa or ~/.ssh/id_dsa	IdentityFile in SSH client configuration
Authorized Key	A public key that is trusted by a “To Account” on a “To System,” such that any user with the corresponding private key is authorized to log on as the “To Account”	~/.ssh/authorized_keys	AuthorizedKeysFile in SSH server configuration

A key association, which essentially defines a trust relationship, always involves two systems (a “From System” and a “To System”) and two users (a “From Account” and a “To Account”):

- The connection is initiated by the “From Account” on the “From System.”
- The destination of the connection is the “To System” using the security context of the “To Account.”
- The “To Account” controls who is authorized to connect to its account through authorized keys. The public keys that are authorized to connect to the “To Account” are listed in the authorized keys file.
- If the private key of the “From Account” corresponds to an authorized public key, then the connection is allowed.

Secure SSH: Background & Risks

The path name of the file containing authorized keys is specified by the AuthorizedKeysFile sshd configuration parameter. The authorized keys file contains one line per key and can include additional restrictions on how users can connect and what they are allowed to do once they have connected. The public keys contained within an authorized keys file define a key association, which is a relationship between a “To Account” and a public key.

It is important to note that the authorized key association is defined only via a public key, not to a specific “From Account.” Anyone in the network who has a private key that corresponds to the authorized public key can use the association to connect through SSH. The fact that keys, rather than specific user names, are defined in the authorized keys file makes it difficult to determine exactly which users are trusted to connect. Good security practice requires that organizations know who has access to connect to their system, and this is a common sense question that any auditor would ask. It is not particularly helpful or reassuring to answer that anyone with a corresponding key (whoever that might be) can access the system. However, many organizations know little more than this because the default implementation of SSH does not provide sufficient visibility to provide a more definitive answer to the question of who has access. This is a significant problem.

“…the default implementation of SSH does not provide sufficient visibility to provide a more definitive answer to the question of who has access. This is a significant problem.

The ssh-keygen utility that creates key pairs can include a comment field showing user@host at the end of public keys. When this field exists, the user@host does not enforce any restrictions on access – it is just a comment. The comment is added when the key is created, so it is not changed if a key is copied to another computer or sent to another user. For this reason, the comment is not helpful in identifying who is using the key or where they are connecting from. In addition, the comment can be changed easily (with the –C option) and therefore the comment is often inaccurate. Therefore, the comment in public keys should not be relied upon as providing any meaningful security protection. It may provide a helpful clue as a starting point for investigating the key, but is not reliable beyond that.

Issues with SSH Key Associations

User's Control over Key Creation

The default implementation of OpenSSH allows users to create their own keys with an easy-to-use utility called ssh-keygen. The private key file, called an identity file, is stored in a directory within the key owner's home directory. By default, a user's private SSH key is stored in the ~/.ssh/id_rsa or ~/.ssh/id_dsa file within the user's home directory, depending on whether RSA or DSA public/private key pairs are used. The corresponding public key is stored as ~/.ssh/id_rsa.pub or ~/.ssh/id_dsa.pub.

Users can modify this default directory and define where their key identities are stored. The location of the user's private key identity is defined through either configuration parameters of the SSH client or run-time parameters. Configuration parameters for the SSH client program can reside in two places: a centralized system-wide configuration file located at /etc/ssh/ssh_config, or a user-specific configuration file ~/.ssh/config stored in each user's home directories. The IdentityFile <identity_file_name> parameter can define one or more identities that are checked in sequence. There is also a run-time option (e.g., -I <identity_file_name>) that can be used to specify an identity when the SSH client is invoked.

This provisioning model puts control in the hands of the user, and there is little that can be done about it. Anything that is stored in user's home directories is ultimately accessible by the user. Users can issue themselves keys, even multiple keys, and store them anywhere they want. It is, of course, possible to make changes to the system-wide ssh_config file, but users can override those settings by specifying run-time options or using a ~/.ssh/config file, both of which are completely under the user's control. The bottom line is that users can create their keys (e.g., "self-serve") and store them anywhere in the file system. Because keys can be distributed throughout the file system (at least in users' home directories and possibly in other directories as well), it is difficult to find all of the SSH identities that exist on a host. This creates a significant challenge when trying to determine the identity of a particular key's owner. Copies of the corresponding private key could be located throughout the file system of any host in the network that runs an SSH client.

Secure SSH: Background & Risks

Protecting Private Keys from Unauthorized Use

To provide a reliable authentication through keys, SSH private keys need to be protected from unauthorized use. Two mechanisms are available for protecting keys (passphrases and security permissions), but the "self-serve" model of deploying keys can undermine both of these controls. Passphrases are generally used as a second level of control for key-based authentication: security by what you have (e.g., a key) and by what you know (e.g., a passphrase). When a passphrase is entered, SSH encrypts the private key with 3DES to protect it against compromise. Each time the user runs the SSH client to initiate a connection, the system will prompt for the user's passphrase in order to decrypt the private key. If the user does not provide the correct passphrase, then SSH blocks the connection. The OpenSSH utility for creating public/private key pairs always prompts users to enter a passphrase that must be entered in order to use the private key. However, the key generating utility accepts null passphrases, that is, no passphrase.

Since a passphrase is entirely optional, as can be imagined, most users who create their own keys do not use one. A small minority of users who intend to follow good security practices might set a passphrase, but often use phrases that are too short or otherwise too easy to crack. The documentation for SSH key generation program (called ssh-keygen) contains sound guidance for setting a passphrase: "Good passphrases are 10-30 characters long, are not simple sentences or otherwise easily guessable, and contain a mix of upper and lowercase letters, numbers, and non-alphanumeric characters." Few users will follow that advice. A key with no passphrase is more susceptible to compromise.

When no passphrase exists, the only control to protect private keys from unauthorized use is access controls. The permissions on the private key file should be restrictive to allow only the key's owner (and root) to have any access. Permissions are especially important for keys stored in the default locations, as these are known targets for attackers looking to round up unprotected keys.

It is also worth pointing out that there is no way to recover a lost passphrase. If a user loses or forgets their passphrase, their existing keys are unusable. In this event, the user must generate a new key pair and copy the public key to the authorized_keys files on target SSH servers. This can result in the existence of multiple pairs of keys for the same user. In the "self-serve" model, it is unlikely that users will cleanup the environment by removing the unusable keys, so key associations can quickly get cluttered.

Secure SSH: Background & Risks

Protecting Keys Used by System Accounts

System accounts are used by operating systems, middleware, databases, and applications for running processes and allowing system components to communicate with each other. For example, an application may use a system account (applprod) to run an application process, which uses another system account (appldb) to access a backend database, whose DBMS processes are running under an “oracle” account. Each of these system accounts is likely to have sensitive levels of access to system resources. The “applprod” account, for instance, is likely to have the ability to create, delete, and update files related to the application.

Keys assigned to system accounts present a special challenge since there is no human to type a passphrase when the SSH client is initiated. One alternative is to encrypt the private key with a passphrase and store the passphrase in the file system so a system process can access it when it needs to use the key to connect via SSH. Although this might sound more secure than using an unprotected key with no passphrase, in fact both solutions provide similar levels of security. Either way, file permissions are the only control to protect the private key. Authentication controls will be ineffective if an attacker is able to compromise the file system security or obtain a copy of the files off-line (through a backup tape).

A better solution may be to use the ssh-agent utility, which is a program that keeps private keys in memory so that they can be accessed whenever private keys are needed. With an agent, passphrases are loaded into memory once when the system boots using the ssh-add command, and after that, no further prompting for passphrases occurs. Although the passphrases could theoretically be hacked from system memory, use of an agent is more secure because it is harder to attack the memory of a running process than to get access to a file on the file system. The major downside of using an SSH agent is that manual intervention is required at boot-time to restart the agent and load passphrases, so unattended system reboots are not possible.

Secure SSH: Background & Risks

Compromise of Identity File (Private Key)

While system keys tend to be more carefully guarded, it is often typical for less attention to be given to securing keys assigned to human users. Human users with unprotected keys, those without a passphrase, may expose their keys accidently. Since keys are stored in users home directories, users have the ability to place their private keys in jeopardy. Actions such as loosening permissions so that other users can read their private keys, copying private keys to unsecured locations, even knowingly sharing keys with other users can undermine the integrity of the authentication mechanism used by SSH.

The SSH client program will not use an IdentityFile that is accessible to users other than owner when the client is invoked. However, this does not address the risk of an IdentityFile private key that had been exposed in the past. Even if an IdentifyFile is properly secured now, an attacker may have obtained a copy of the key previously when the file was unsecured. Anyone who can access a copy of the private key can authenticate and connect through SSH.

Compromise of Authorized Keys File (Public Key Associations)

The authorized keys files have similar issues in the default implementation, but can be more tightly secured. The authorized keys file for an account (“To Account”) contains the list of all public keys that are authorized by the account owner to connect using his or her account. By default, authorized keys files are stored within the user’s home directory as ~/.ssh/authorized_keys. The location of the authorized keys file is defined by an sshd configuration setting called AuthorizedKeysFile.

This default configuration allows users to “self-serve” and set-up new associations with their own public keys or another user’s public keys. In the default deployment of SSH, authorized key files are accessible by their owners. This enables the account owners to add new associations at will. Since the files are stored in user home directories, it is impractical to attempt to make files inaccessible to users. Even if you try to change file permissions and ownership, the account owner still has access to the parent directory and can delete the file and add a new one with the same name. Furthermore, using the default configuration and storing authorized_keys within individual users’ home directories causes key associations to be stored throughout the file system, wherever home directories are located. Because key associations are distributed throughout the file system, it is difficult to produce an inventory of key associations or provide oversight of them.

Secure SSH: Background & Risks

The authorized keys file is sensitive – if an unauthorized user can obtain write access to the file, they could add their own public key and be able to connect. There is a configuration setting in the sshd_config file (StrictModes yes) that can help protect against a poorly secured authorized keys file. If a user's authorized keys file or home directory is writeable by anyone other than the user (and root), then this setting will keep the SSH server from using the file until the user tightens the restrictions. The StrictModes configuration setting is is a good feature to use since users are warned about poor permissions and can fix them. However, it does not itself address all of the risk. In the intervening period, an unauthorized user could have created an association by adding keys to the authorized keys file. If the permissions of the authorized keys file are ever compromised, reviewing and validating each and every installed public key needs to be performed. However, it is highly unlikely that users go through that extra step.

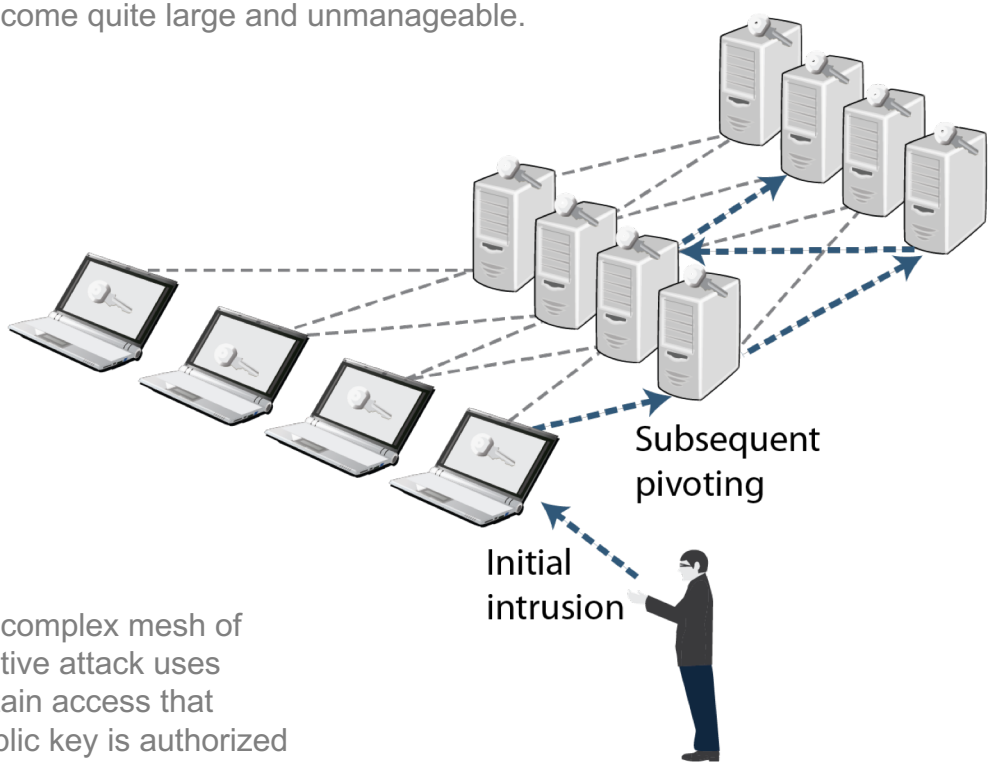
Challenges of Using Keys to Access System Accounts

The OpenSSH model is designed to allow users to add new key associations, and this can create risk, especially for system accounts. IT support personnel may obtain temporary access to system accounts (presumably through a controlled access method) in order to fix production outages or other problems. While logged on as the system account, those users could create new associations to the system accounts by adding new keys to the system account's AuthorizedKeysFile. This may make access easier the next time that there is an outage, but also may allow an alternate access method via SSH keys that circumvent other controls for limiting access to the system account. As a simple example, it is common to restrict knowledge of passwords for system accounts and to change those passwords when support personnel no longer require access. The intent of these password controls is to limit access to the system account except under authorized circumstances. These controls, however, can be circumvented through the use of SSH key based authentication. Regardless of any password change, if a user has a key association defined to a system account, they will still be able to connect as the system account.

Secure SSH: Background & Risks

This also introduces a new challenge to auditors who are trying to determine who can access system accounts with sensitive access. The old question “who knows the password to the account?” is no longer sufficient. Another question is equally important: “who can use the SSH keys listed in the authorized keys file?” Furthermore, over time the number of keys that have associations with an account could become quite large and unmanageable.

Remember that the authorized keys file only list public keys and does not have a mechanism to track the identities (or “From accounts”) of users that can use the key to connect. This creates uncertainty when considering a basic security question like: “who can connect to this system?” Many organizations with OpenSSH implementations are not able to provide a definitive answer to this question.



Transitive Attacks

Worse yet, SSH key associations can create a highly complex mesh of trust that creates the risk of transitive attacks. A transitive attack uses two or more explicitly defined trust associations to obtain access that was not directly assigned. For example, if UserA’s public key is authorized to associate with UserB, and UserB’s public key is authorized to associate with UserC, then effectively UserA can connect as UserC. (This is true unless UserB’s private key has a good passphrase defined, but that is hardly ever the case.) Because the mesh of key associations is so dense, it is likely that transitive attacks could quickly spread throughout many servers in the environment and could potentially reach root-level access. Unfortunately, the key associations that allow for transitive attacks are even harder for organizations to identify and control.

Conclusion

This eBook has provided information on key associations and discussed many of the security issues that can arise from their implementation. In our next eBook, we will discuss risk management for SSH implementations, including key discovery, establishing a key association registry, and centralizing key provisioning.

Partnering with SecureIT

We hope you find this eBook helpful in your organization's path towards achieving a stronger security posture. As demonstrated above, SecureIT understands the operational practices and risks around network access but we also realize that no two organizations are alike. When SecureIT engages with our clients, we invest the time and resources to understand your organization, your software or services solution, and where you are in your journey.

Partnering with SecureIT to discover and mitigate security risks means ensuring that both your immediate and long-term compliance goals are achieved in an efficient manner so you can focus on the core job of increasing sales opportunities and growing revenue. Please contact us today, we would love to learn about your situation.

SECURE IT

About SecureIT

SecureIT provides risk, compliance, and cybersecurity services to enterprises, government entities, and cloud service providers. Our certified professionals assess cyber risk, conduct targeted security assessments, and ensure compliance with regulatory requirements. Every day, we partner with our clients to deliver solutions critical to protecting and growing business.

12110 Sunset Hills Road
Suite 600
Reston, VA USA 20190
703.464.7010
www.secureit.com