draft-ylonen-sshkeybcp-01.txt
Network Working Group
Internet-Draft
Expires: October 06, 2013

T. Ylonen
SSH Communications Security
G. Kent
SecureIT
M. Souppaya
NIST

April 04, 2013

# Managing SSH Keys for Automated Access - Current Recommended Practice

## Abstract

This document presents current recommended practice for managing SSH user keys for automated access. It provides guidelines for discovering, remediating, and continuously managing SSH user keys and other authentication credentials.

Various threats from poorly managed SSH keys are identified, including virus spread, unaudited backdoors, illegitimate access using leaked keys, lack of proper termination of access, use of legitimate access for unintended purposes, and accidental human errors.

Hundreds of thousands, even over a million SSH keys authorizing access have been found from the IT environments of many large organizations. This is many times more than they have interactive users. These access-granting credentials have largely been ignored in identity and access management, and present a real risk to information security.

A process is presented for discovering who has access to what, bringing an existing IT environment under control with respect to automated access and SSH keys. The process includes moving authorized keys to protected locations, removing unused keys, associating authorized keys with a business process or application and removing keys for which no valid purpose can be found, rotating existing keys, restricting what can be done with each authorized key, and establishing an approval process for new authorized keys. A process is also presented for continuous monitoring and controlled authorized key setup.

Finally, recommendations are made for security policy makers for ensuring that automated access and SSH keys are properly addressed in an organization's security policy.

Specific requirements are presented that address the security issues while keeping costs reasonable.

Guidance is also provided on how to reduce operational cost while addressing the threats and how to use tools to automate the management process.

## Status of This Memo

This Internet-Draft will expire on October 06, 2013.

## Copyright Notice

## Table of Contents

## 1. Introduction

### 1.1. Purpose and Scope

This document focuses on risks related to poorly managed automated access in information systems and particularly SSH user keys, and how to reduce the risks. It documents current best practice of managing SSH keys for cost-effectively minimizing the risks, and provides security policy recommendations and auditing guidelines relating to SSH keys and other automated access.

### 1.2. Audience

This document is intended for information security policy makers, auditors, security managers, IT operations managers, system administrators, and others who are responsible for specifying, acquiring, testing, implementing, maintaining, and auditing identity and access management and SSH solutions. Portions of the document may be of interest to technically advanced end users and systems programmers involved with SSH and other automated access technologies.

### 1.3. Structure of This Document

Section 1.4 specifies what certain words indicating level of requirement for compliance with this standard mean.

Section 1.5 defines impact levels for information systems, including some important definitions relating to information systems having low impact themselves but having automated access to higher-impact information systems.

Section 2 summarizes the basics of the SSH protocol and implementations, with particular emphasis on authentication methods for automated access and typical use cases for automated access.

Section 3 describes threats arising from poorly managed SSH user keys. Most of the threats are also relevant for other kinds of automated access. However, because of the ubiquity of SSH keys and the acuteness of addressing them the discussion focuses on SSH keys.

Section 4 introduces simple metrics and questions that are useful in scoping the risks related to SSH user keys and gaining basic understanding of the size of the problem in an organization. The approach is suitable for both IT auditors responsible for assessing compliance with security policies as well as government and other policy makers wanting to measure the overall state of compliance and security across agencies.

Section 5 introduces the process for detailed analysis of existing automated trust relationships and risks (with an emphasis on SSH user keys), as well as recommended steps for remediating the risks. This section also discusses the specific threats addressed by each remediation step and risks involved in not implementing a particular step.

Section 6 provides recommendations for continuous monitoring and management of SSH user keys and other automated trust relationships, as well as for auditing steps to be taken for ensuring that an organization keeps automated access under control after an initial remediation phase.

Section 7 provides recommendations for an organization's security policy for properly addressing SSH user keys and automated access.

Section 8 summarizes issues to consider when planning use of automated software tools for managing automated access with SSH and particularly SSH user keys. It also illustrates how to achieve cost savings in existing operational processes.

Section 9 summarizes security considerations. Most of this document is about security and managing elements of security and access, and this section serves as a conclusion and summary of this document.

Section 11 provides a glossary of the technical terms used in this document.

### 1.4. Words Signifying Level of Requirement

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

### 1.5. Impact Levels for Information Systems

The appropriate level of security and effort expended on security often depends on the level of impact from a failure or compromise of an information system. FIPS Publication 199 [FIPS199] provides designations for impact levels on organizational information systems and a process for categorizing information systems.

This document makes reference to the impact levels described FIPS 199 (please see original document for exact definitions, this is just a simplifying summary):

Low impact: Unauthorized disclosure, modification, destruction, or disruption of access have limited adverse effect on organizational operations, organizational assets, or individuals.

Moderate impact: Unauthorized disclosure, modification, destruction, or disruption of access could be expected to have a serious adverse effect on organizational operations, organizational assets, or individuals.

High impact: Unauthorized disclosure, modification, destruction, or disruption of access could be expected to have a severe or catastrophic adverse effect on organizational operations, organizational assets, or individuals.

FIPS Publication 199 analyzes impact levels separately for confidentiality, integrity, and availability. For considerations around automated access, the impact level of access to an account on an information system is taken to be the highest level of these three principles for the information system, since this specification primarily relates to operating system level access to information systems, and operating system level access can often be used to breach all three objectives simultaneously. Furthermore, experience has shown that once an attacker has operating-system level access to one user account on a computer, various attack vectors (including bugs in system software and misconfigurations) can often be utilized to escalate the access to high-level administrative access. That definitely compromises all three principles of information security.

Configured trust relationships for automated access (e.g., using SSH user keys) may permit access from low-impact information systems to high-impact information systems without providing a password or other authentication credential from a user. This is particularly relevant if the authentication credential or authorized key permits access on the high-impact information system without restrictions on the commands that can be executed on the high-impact information system. In this case, access to the low-impact information system implies access to the high-impact information system. The information system owner inherently accepted this risk by allowing a low-impact system access to a high-impact system with providing compensation controls. There may also be situations where the high-impact system owner may not know the key has been copied to a low-impact system, or from one low-impact system to another.

Therefore, whenever a low-impact information system has a configured trust relationship permitting it to access a high-impact information system without a restriction on the commands that can be executed on the high-impact information system, the low-impact information system MUST be treated as having the impact level of the highest-impact

information system that it can access using automated trust relationships.

This implies that to avoid treating low-impact information systems as high-impact systems, there must be a well-defined boundary in the IT environment that trust relationships can only cross in the direction allowing access from higher-impact systems into lower-impact systems, but not vice versa.  If such boundary is relied on, it MUST be audited and continuously monitored to enforce its existence.  Such a boundary could exist, for example, between development and production systems.

Sometimes otherwise low-impact systems are used for producing code, software distributions, or data sets that will be used on higher-impact systems.  Such systems SHOULD be treated as higher-impact systems in view of Advanced Persistent Threat (APT) scenarios where an attacker could insert a backdoor in software that eventually gets copied to production systems.

It should be noted that several current SSH implementations (including OpenSSH) only permit configuring command restrictions for access based on SSH user keys.  It is currently not possible to configure command restrictions for Kerberos-based authentication, host-based authentication, hard-coded passwords, or passwords coming from password vaults, which has implications for the above requirement.

Command restrictions are a compensation control that can be leveraged to minimize the exposure to the additional risks exposed to a high-impact system from allowing limited access to the hosted resources from a low-impact system.  Command restrictions used for this purpose MUST be designed to be effective in limiting what actually can be done with the access, and MUST prevent interactive access and port forwarding.  For example, a command restriction permitting arbitrary commands or interactive shell is not effective.

Furthermore, if a trust relationship has a command restriction that limits its use to file transfers but the directories from which files can be read or modified using it have not been restricted, it exposes the server to a more limited risk.  The trust relationship may be used to read any file or directory on the server accessible to the user account used for file transfers, even those outside the intended directories.  It may also be used to write any file that is writable by the user account; it is fairly common to have configuration files on servers that are inappropriately world-writable, in which case these files could be modified.

If a trust relation is restricted to file transfers but does not limit the directories that can be accessed, the originating information system SHOULD be considered as having at least the impact level of the highest-impact information system to which it has such access.

## 2. The Basics of SSH Protocol and Implementations

SSH (Secure Shell) is a protocol and software tool for logging into a remote machine, executing commands remotely, and transferring files with a remote machine over a computer network. SSH can also be used for implementing a protected tunnel for delivering other services.

SSH is very widely used for administering Linux and Unix systems, virtual machines, routers, firewalls, and other network devices. It is also embedded in many leading file transfer solutions, systems management solutions, identity management solutions, and privileged access management solutions. It is widely used for integrating IT systems and automating their operation.

### 2.1. The SSH Protocol

The SSH protocol is an IETF standards track protocol and is described in RFC 4251 [RFC4251], RFC 4252 [RFC4252], RFC 4253 [RFC4253], and RFC 4254 [RFC4254].

Many independent commercial and open source implementations are available, including Tectia SSH, OpenSSH, and many others. SSH is available for nearly all platforms, including Linux/Unix, Windows, mainframes, routers, telephone exchanges, mobile devices such as smartphones and tablets, various embedded devices, and many industrial automation systems.

In the SSH protocol, an SSH client application initiates a TCP/IP connection over a network to a destination server, negotiates encryption, authenticates the remote server, and then sends a destination user name and authentication credentials to the server. Server authentication is done using host keys, and its primary purpose is to prevent man-in-the-middle attacks; however server authentication is beyond the scope of this document.

### 2.2. User Authentication in SSH

The SSH protocol supports several mechanisms for authenticating users, including passwords, public key authentication, Kerberos, and host-based authentication.

### 2.2.1. Password Authentication

There are two kinds of password authentication mechanisms in SSH: basic password authentication and keyboard-interactive authentication. Keyboard-interactive authentication can support various types of challenge-response systems and various other authentication mechanisms.

Password authentication is commonly used for interactive users, but less commonly for automated access (through it is sometimes seen with hard-coded passwords in scripts and management systems, especially for managing routers and file transfers).

### 2.2.2. Public Key Authentication

Public key authentication in SSH uses user keys or certificates to authenticate/authorize a connection. An SSH client has an identity key, typically an RSA or DSA private key, and the server must have the corresponding public key configured as an authorized key for the destination user. The private key may be protected by a passphrase, in which case it is encrypted by a key derived from the passphrase (passphrases are common for interactive users but rarely used for automated access).

Many widely used SSH implementations support configuring restrictions for SSH user keys. These may be used for limiting what can be done on the server using the key (command restrictions) and for limiting the IP addresses from which the key can be used (source restrictions).

Public key authentication is by far the most frequently used method of configuring automated access using SSH at the time of this writing and represents best current practice.

### 2.2.3. Kerberos Authentication

Many organizations use Kerberos or Active Directory authentication with SSH. Kerberos (usually together with LDAP) implements single sign-on and allows user accounts to be stored in a centralized directory.

In practice, Kerberos is rarely used for non-interactive accounts. While it can be configured to use keytab files or cached tickets for functional accounts, these approaches rely on having long-term credentials stored on the host or at least accessible to the process on the host that is obtaining tickets. These credentials can be exploited by an attacker largely in the same way as SSH user keys, e.g., by using them to obtain a ticket granting ticket (TGT) for the

functional account and using the ticket to gain access to other hosts or accounts that the functional account can access (see virus spread threat below).

One problem with Kerberos for automated access is that the single sign-on feature implies that once access has been gained to one account using Kerberos, it is usually possible to log in to any other server that has the same account and is in the same domain without further authentication. This can very easily create lots of unwanted implicit trust relationships. Existing implementations also do not support command restrictions for Kerberos.

### 2.2.4. Host-Based Authentication

Host-based authentication uses the source host's host key to authenticate the source host and to vouch for the identity of the user on the client side. It is rarely used and does not permit configuring command restrictions. Therefore its use for automated access is NOT RECOMMENDED.

### 2.2.5. Comparison of the Authentication Methods

All these authentication methods fundamentally rely on some secret information, and when used for automated access, this secret information must be stored on or accessible to the source host.

A major advantage of public key authentication over the other methods is that it allows configuring a command restriction. The command restriction can be used for preventing virus spread and other attacks, as described in Section 3. It also does not create any implicit trust relationships and the permitted access can be reliably determined by inspecting the destination host (except for OpenSSH's proprietary certificate authentication, which SHOULD NOT be used because it cannot be reliably audited). For these reasons, public key authentication is the RECOMMENDED authentication mechanism for automated access with SSH.

Password authentication SHOULD NOT be used for automated access, because hard-coded passwords may be obtained by attackers and password vaults are also not particularly secure against local attacks on the client software. If password authentication is used for automated access, the passwords MUST be rotated every three months.

### 2.2.6. Dangers of Unverified and Shared Host Keys

Many file transfer applications, privileged access management systems, and systems management applications do not check host keys

for hosts that they connect to. This permits a man-in-the-middle attack to be performed in the network. Many tools are available for this and any device connected to a network through which the connection goes can be used for the attack - including, e.g., reprogrammed smart switches.

Man-in-the-middle attacks are a risk regardless of the authentication method if hosts keys are not properly verified. The attack permits injection of arbitrary commands into the session, and reading and modifying any transferred files (including injection of bogus file transfers). A successful man-in-the-middle attack from the network gives the same power as being able to use a trust relation leading to the destination host.

Such man-in-the-middle attacks are practical, and are exploited in freely available attack tools and malware, as well as security software from multiple vendors for co-operative auditing purposes.

Besides applications that do not check host keys, there are also some large enterprise that share the same host key on thousands of machines (for example, one Fortune 500 company is known to use the same host key on 14000 computers at the time of this writing). If any of the computers is compromised, they all become vulnerable to man-in-the-middle attacks.

Therefore, while this document is not really about host keys, the destination host MUST be properly authenticated by the client for all automated access and a unique host key MUST be used for each host. An exception may be made for the very first connection to a server to simplify system administration.

## 2.3. Common Use Cases

### 2.3.1. Interactive Use

SSH has become the standard used by system administrators for configuring and managing Unix and Linux computers, routers, and various other equipment remotely. It is also widely used by software developers, and in some organizations by ordinary end users for running applications remotely (particularly text-based legacy applications). Public key authentication is often used by advanced end users for single sign-in. Sometimes it is also used on jump servers.

### 2.3.2. Automated Access

SSH is very frequently used for automated access between systems. Such automated access is necessary for cost-efficiently managing

large IT environments, for integrating applications, and for cost-effectively provisioning virtual machines in cloud services.

Automated access refers to accessing a computer from another computer in an automated fashion.  Automated access may be unrestricted, allowing any commands to be executed, or may be limited to specific commands or operations, such as file transfers (perhaps limited to a specific directory).

Automated access is most commonly used with functional accounts, system accounts, service accounts, and other non-interactive user accounts (sometimes also called non-user accounts).  Such accounts are used by operating systems, middleware, databases, and applications for running processes.  System or service accounts are likely to have sensitive levels of access to system resources (in that case they are often called privileged accounts).

Automated access using SSH is common also in Windows and mainframe environments, especially for file transfer applications.  There are also various native mechanisms on Windows that can be used for automated access, but such mechanisms are beyond the scope of this document.

Automated access has been largely ignored in Identity and Access Management.  Hundreds of thousands to over a million authorized SSH keys have been found from the IT systems of several large enterprises.  This means that they have many times more entry points for automated access configured on their servers than they have interactive users in the organization!  It is clear that such entry points cannot be ignored.

### 2.3.3.  File Transfers

SSH is frequently used as a file transfer tool in itself, using the "scp" and "sftp" tools.  The SFTP [SFTP] protocol is gaining popularity in commercial and open source file transfer products, and a substantial fraction of the world's file transfers now use the SFTP protocol.  Automated file transfers using SSH typically use public key authentication or hard-coded passwords, and they are often also used between organizations.

## 3.  Threats Arising from Poorly Managed Automated Access and SSH Keys

This section outlines some of the threats that have been identified with poorly managed SSH keys.  The guidelines and recommendations of this document are intended to address these while minimizing the administrative burden from managing the keys.

Several of the problems described below are present with many technologies for automated access besides SSH keys.  The issues must be addressed regardless of technology.

### 3.1. Virus Spread Threat

Malware can be engineered to use SSH keys to spread to most servers within an organization once it has penetrated one server.  Experience has shown that viruses frequently manage to penetrate individual servers in an organization.  Malware often uses multiple attack vectors to penetrate an organization and could use SSH user keys (or other trust relationships such as Kerberos) to spread within the organization's server infrastructure in minutes after penetrating the first server, thereby defeating layered security defenses.

The Morris worm in 1988 utilized automated access trust relationships to spread in a similar manner (at that time based on ".rhosts" authentication).  This attack vector can be very powerful, and its importance is increasing as systems management becomes more automated.  Many computer forensics experts are aware of cases were SSH keys have been used to spread an attack from one server to another, and several high profile incidents in the last year have used SSH keys as an attack vector.

Experience has shown that most large organizations have 3 to 100+ SSH keys configured granting access to each Unix/Linux server.  Some keys grant high-level administrative access or access to sensitive accounts, such as those storing database files or critical software. In practice it has been found that in many organizations approximately 10% of SSH keys grant access to root accounts or other privileged accounts.

The mesh of automated access relationships is so dense in many cases that it is likely that an attack can spread to most servers in an organization after penetrating the first few servers, especially if other attack vectors are used to escalate privileges.

Implementing SSH keys as an attack vector in malware is quite easy, requiring only a few hundred lines of code.  Once the malware has penetrated a server, it may use the server to further the attack and/or, e.g., leave a backdoor, steal, alter, or corrupt data, subvert encryption systems or databases, or outright destroy the server.

The virus spread threat can be reduced by combining several approaches:

> Mandating forced command restrictions for as many trust relationships as possible.

Removing trust relationships that are no longer needed.

Minimizing the number of trust relationships leading to root accounts (directly or indirectly).

Minimizing implicit trust relationships arising from privilege escalation (e.g., "sudo"), single-sign-on (e.g., Kerberos), and host equivalence.

### 3.2. Unaudited Backdoor Threat

Many large organizations mandate that all privileged access to their servers take place through a privileged access management system that records any actions performed. Key-based access (and other automated trust relationships) can be used for creating backdoors that bypasses such privileged access management systems.

System administrators and production support personnel regularly gain access to various accounts in the course of legitimate work. An administrator or production support person may add a new authorized key to an account with a single command (e.g., "echo ...keydata... >>~/.ssh/authorized_keys"). As of this writing, most organizations never audit authorized keys for their user accounts, and thus the added key may remain unnoticed for years. Such a key can then be used to log into the account using any SSH client, bypassing the privileged access management system. It thus provides a relatively permanent unaudited backdoor.

Key-based backdoors can also circumvent password vaults and systems that change root (or other privileged account) passwords regularly.

Experience has shown that many organizations have no control or tracking of trust relationship creation. Any system administrator or production support personnel can create and install a user key pair as needed without any reporting, logging, or authorization. Such practice undermines traditional controls for privileges access.

The unaudited backdoor threat can be reduced by the following:

Prevent non-root/non-Administrator users from granting automated access to accounts without proper approval. For example, move all authorized keys files to root-owned directories that are not writable by normal users.

Continuously monitor the environment to detect unauthorized trust relationships configured by someone having root access.

Require proper explanation and valid purpose for the existence of every trust relationship and remove any unused trust relationships.

Use privileged access management systems that capture SSH and other protocols using automated access on the network level or at the server (transparent access auditing). Enforce privileged access management for connections using automated trust relationships.

### 3.3. Leaked Keys May Provide Access for Extended Periods

Most security policies and regulations mandate that all passwords must be changed regularly, e.g., every three months. Some security standards mandate that encryption keys must also be changed regularly. However, very few security policies at this time make it explicit that authentication/authorization keys should also be regularly changed. In a sense, authentication keys are even more critical than encryption keys, because once access to a user account has been gained, it is generally possible to access and modify any data for that user account - including reading and modifying memory of processes running under that user account and/or modifying any executable programs owned by that user account, thus subverting encryption systems and other critical applications.

At the time of this writing, most organizations do not track which SSH keys their users, administrators, backup operators, and janitors may have had access to and copied over the years. In addition, they never change their SSH keys. Most environments do not use source restrictions on authorized keys. Therefore, a leaked key may be used from any computer or network within the organization (unless limited by internal firewalls).

This means that anyone who may have obtained a copy of a key (e.g., by copying it from a host, accessing a backup, or having acquired some decommissioned equipment that was not securely wiped) may gain access to production servers in the organization.

No audit or discovery process can ever guarantee finding all copies of identity keys, as they are small files that could be hidden anywhere, and there could be copies on laptops, tablets, USB sticks, offline, and even on paper (they are small enough to be typed in). Identity keys can easily be taken out from an organization on a single piece of paper, or by taking a photograph of a screen using a cellphone.

There have also been many instances where private keys have been uploaded to, e.g., "github" (a repository used by many open source

software development projects). In January 2013, hundreds of private keys and passwords were found from the repository, some of which were being used for attacks. Obviously, identity keys MUST NOT be uploaded into any public repository.

The problems created by leaked or unchanged keys can be reduced by:

Rotating all keys regularly to guarantee the eventual termination of access.

Configuring source restrictions for authorized keys, making it more difficult to exploit copied identity keys.

Using certificate-based authentication, which can provide revocation and expiration, but is cumbersome to manage (and still does not protect from some of the other threats).

Using Kerberos authentication, which allows terminating access to the account; however, Kerberos authentication does not in itself prevent leaked keys that have not been changed from being used.

Besides rotating keys at regular intervals to avoid their leakage and to limit the duration of the exploitation window should they leak, periodic rotation also applies to credentials used for obtaining actual authentication credentials; for example, it is not enough to periodically obtain a new Kerberos ticket - one must also regularly change the authentication credentials used for obtaining an initial ticket. It is also not enough to issue a new certificate for the same private key - the private key must also be replaced by a newly generated private key.

### 3.4. Lack of Proper Termination of Access

Most security standards mandate proper termination of access when an employee leaves or changes roles. If the user remains in possession of identity keys that continue to have access to the organization's information system, access is not being properly terminated.

Since administrators can quite easily copy identity keys (and may have legitimately configured key-based access from their personal account to various accounts they used in their previous role), the only practical way to guarantee proper termination of access is to remove or rotate any keys that the employee may have had access to.

At the time of this writing, most organizations do not know what each key is used for. Without this knowledge, they seldom remove or rotate keys, because something could break if they accidentally remove a key that is needed for some important business process or

omit some authorized keys corresponding to a public key being rotated.

Some organizations have used manual spreadsheets for tracking key usage. However, it has turned out they are usually out of date, inaccurate, and have not been maintained throughout the organization. Many organizations have no monitoring whatsoever of automated access or new user key setups.

Proper termination of access can be ensured by:

Moving all authorized keys files to root-owned directories that are not writeable by non-privileged users.

Regularly rotating keys (ensures termination of access by copied keys latest at next key change).

Triggering immediate key rotation for private keys on accounts accessible by the person whose access is being terminated.

### 3.5. Use for Unintended Purpose

Firewalls commonly have rules that permit specific communications for file transfer purposes. When the file transfer is using SSH (or SFTP), it is important that a forced command be used on the server to ensure that the access permitted through the firewall cannot be used for other purposes, such as executing shell commands on the server.

Another related use case is employees creating their own backdoors into the enterprise to circumvent corporate policies against uncontrolled remote access by opening an SSH connection from the office to their home machine with a port forwarding from the home machine back to the office machine. Such backdoors may provide hackers an entry point into the company intranet, especially if the home machine is compromised and the user's password is obtained using, e.g., key logger malware.

Various commercial products are available for auditing SSH connections at a firewall to enforce that opened ports are not used for unintended purposes regardless of server configuration.

Various SSH implementations permit port forwarding even when forced commands are used. Therefore, a trust relationship that is intended only for file transfer may actually be used to obtain a connection to any port at any host on the internal network (or external network, for hiding the source of an attack).

The threat of unintended use can be minimized by:

Allowing SSH/SFTP through a firewall only when required and restricting sources and destinations to fewest systems required.

Configuring forced commands for authorized keys used by external parties.

Implementing tools to audit SSH connections at the firewall and monitoring use to ensure that access was not misused.

Avoiding trust relationships that cross security boundaries or allow connections from low-impact systems to higher-impact systems.

### 3.6. Accidental Data Transfers and Human Errors

Not all risks associated with unmanaged automated access arise from malicious behavior. If there is automated access from non-production systems to production systems, data may accidentally be copied from non-production systems into production systems, where the incorrect data may cause substantial loss of money. Alternatively, data may be inadvertently copied from production systems to non-production systems, where the data may be exposed due to looser security controls.

People are also known to make human errors when manually setting up new trust relationships. For example, it is fairly easy for a security administrator to accidentally copy an authorized key to the root account on a host instead of some other account that was intended. Such errors can go undetected for years.

Some key setups involve thousands of hosts. It is easy to miss one or more hosts when copying an authorized key to so many hosts manually. Debugging such errors can be very time consuming. Also, when manually fixing such problems, security administrators are likely to just copy the missing keys to the proper accounts, without, for example, checking whether they have accidentally been copied to the root account.

The threat of accidents and human errors can be minimized by:

Automating key provisioning to implement the authorized keys exactly as they were requested and approved.

Configuring source and command restrictions for authorized keys.

Enforcing policies for preventing trust relationships between systems that cross security zone boundaries.

### 3.7. Problem Under Radar

The SSH key management problem has been recognized in various circles for some time. The scope of the problem, and its relation to automated access overall, however, has not been widely understood.

The problems have remained under the radar because they are deeply technical and obscure, within the domain of system administrators. Each system administrator typically only sees a small corner of the IT environment and does not have a full picture. Although administrators and their managers may recognize that there is a problem, they simply have not had time to analyze the scope or possible implications of the problem. There have also been no guidelines or training materials on how to address it.

Most IT auditors do not have SSH key management or automated access more generally on their checklists or audit programs, yet it is central to identity and access governance given the prevalence of automated access entry points to systems.

SSH keys, or control of credentials for automated access more generally, has not been sufficiently highlighted in security control frameworks and auditing guidelines for FFIEC, SOX, PCI, FISMA, HIPAA, NERC, or COBIT. Even many CISOs are only vaguely aware of the problem, and many CIOs and IT risk management professionals have never heard of it.

Training, books, and systems in the identity and access management space have largely only dealt with actual human users and control of interactive access by people. Automated access by machines has been largely ignored, despite many organizations now having many times more credentials for automated access to their systems than they have interactive accounts.

Despite the risk, the problem will likely not be addressed until IT security auditors, IT operations managers, security architects, CISOs, and IT risk management professionals understand the issue. It must be addressed in security regulations, guidelines, standards, and internal security policies. Education and training will also be needed to ensure that the SSH key management problem and remediation actions are understood. It must be evaluated during audits to ensure that action is taken.

## 4. Assessing the SSH Key Management Situation and Risks

Addressing threats related to automated access and SSH keys begins with understanding the extent to which automated access and SSH keys are used in an organization, understanding how they are managed, and identifying areas likely to require further attention.

Risks associated with SSH key management are generally relevant for organizations where at least one of the following is true (the list is not exhaustive, and other automated access technologies affect other organizations):

significant number of Unix or Linux systems;

significant use of SSH or SFTP on Windows or Mainframe;

virtualization or cloud services that are internally managed using SSH (possibly inside automated scripts/tools/frameworks);

web server farms that are managed over SSH;

network devices (e.g., routers, switches, xDSL models, firewalls) configured and managed using SSH and/or automated management systems;

significant number of automated file transfers using SFTP;

password management or privileged access management tools using SSH to connect to end servers; or

systems management tools using SSH to communicate with managed systems.

Results of the scoping phase help estimate risk exposure and the probability of non-compliance with mandatory regulations.  This information also helps auditors and decision-makers determine whether a more detailed discovery and remediation project is warranted.

Certain types of relatively easily obtainable information are useful in understanding the scope of the problem in an organization.  This information may easily be obtained as part of an audit or regular review.

Some preliminary indicators about the level of risk can be obtained by reviewing the sshd_config file for a sample of SSH servers.  The AuthorizedKeyFile parameter indicates the location of files that store user's authorized key files.  If the AuthorizedKeyFile is located within the user's home directory (which is the default), then

it is likely that a significant problem exists because users are able to provision new trust relationships.  On the other hand, if the AuthorizedKeyFile is defined within the /etc directory, for example, then the risk of inappropriate trust relationships is significantly lessened.  Another significant configuration parameter is PermitRootLogin.  A value of "yes" or "without-password" indicates that SSH keys can be used for root access, which significantly increases the potential impact of poorly managed keys.  However, if this parameter is set to "no" or "forced-commands-only", then the potential impact is substantially lessened since interactive root access is disallowed.

Examining authorized keys provides a meaningful indication of the level of risk.  The following metrics generally give insight into whether an organization is affected by the issue:

total number of authorized keys in the environment;

total number of authorized keys in the environment that grant access to a root account (any account with user id 0);

total number of authorized keys in the environment that grant access to a system account or service account;

total number of authorized keys without a command restriction; and

total number of non-root service accounts or system accounts that have access to add new authorized keys at will.

There are also a number of scoping questions that can give insight into the severity of the problem.  These questions are well-suited for a questionnaire or interview of knowledgeable personnel.  An affirmative answer indicates that SSH keys are being managed; a ``no'' indicates that risk exists.  The questions are provided below:

Does installing a new authorized key require approval from a system resource owner or authorized manager 1) for keys granting root access, 2) for keys granting access to non-root service accounts or system accounts, or 3) for keys granting access to interactive user accounts?

Are such approvals enforced through the provisioning process?

Are non-root users technically prevented from installing new authorized keys, e.g., by moving the authorized keys files to root-owned locations?

Has this configuration been verified to be the case 1) across all high-risk systems and 2) all moderate-impact systems?

Is a continuous monitoring process in place for detecting authorized keys that are added outside of the provisioning process and without proper approvals 1) for root accounts, 2) for service and system accounts, and 3) for interactive user accounts?

Is a policy in place for rotating SSH user keys?  Are monitoring procedures in place to verify that all user keys have actually been rotated in accordance with the policy (and the private keys actually changed)?

For all authorized keys for the root account (and critical system and service accounts), can the organization easily identify who is using the keys to connect?

Has the reason of existence for every authorized key, including the application or business process it relates to, been documented 1) for high-impact systems, 2) moderate-impact systems, and 3) low-impact systems?

Are SSH keys systematically removed when they are no longer needed?

Do all authorized keys used for external SFTP/SCP file transfers with other organizations have a command restriction?

Are command restrictions enforced for trust relationships leading to moderate-impact and high-impact systems?

Preliminary scoping information can be obtained relatively easily and scoping questions can be answered without having to install new software on servers.  However, the answers are only approximate. Experience has shown that many organizations do not know clear or definitive answers to the questions, and sometimes management perceptions do not match reality.  Therefore the answers are best obtained and analyzed as part of a regular audit that actually verifies the answers, at least by representative sampling.

Another interesting diagnostic exercise to gauge the level of risk is to obtain listings from a few servers of the public keys (or signatures) from the authorized keys file of root and other sensitive accounts (such a system or service accounts).  If the organization can readily identify who is using those keys (or could use the keys) to connect and why, then it is likely that the organization is effectively managing SSH users key for access control.  If the organization is unable to identify who can use keys to obtain

sensitive access to systems, then the access control problem and risk is self-evident.

Freely available scripts and tools for doing a proper scoping analysis are available at http://www.ssh.com/auditing [1]. The scripts and tools will be useful for internal security professionals, system administrators, and auditors working with customers.

## 5. Key Remediation Solution Planning and Deployment

Once it has been determined that further analysis of automated access and/or SSH keys in an organization is warranted, the organization typically engages in a multi-step process consisting of additional discovery and remediation of existing trust relationships, establishment of appropriate policies, and continuous monitoring to ensure that automated access is only enabled in accordance with policy.

A typical key remediation process consists of:

discovering all existing trust relationships based on SSH keys (and other trust relationships, if applicable);

moving authorized keys files to protected locations to prevent non-root users from adding new authorized keys;

monitoring use of trust relationships and authorized keys in the existing environment;

removing trust relationships that are no longer in use;

associating each trust relationship with an application or some other valid purpose;

implementing an approval process for setting up new trust relationships;

rotating existing SSH user keys;

configuring forced command restrictions on authorized keys; and

configuring IP address restrictions on authorized keys.

While it is possible to perform all the remediation steps manually, in a larger environment the use of software tools to assist in the process can save a huge amount of work. Parts of the process can be fairly labor-intensive, for example, associating each trust relationship with an application or valid purpose may require a

substantial amount of manual work, and removing of unused trust relationships needs to be done with care to avoid any problems with critical business applications.  (See Section 8 for more information.)

Automating management of SSH keys and other trust relationships can also bring substantial cost savings.  Many organizations spend a substantial amount of administrator time setting up and maintaining trust relationships, and the cost of such manual key management can often be eliminated by automating the process.  Ideally, new trust relationships are approved in the organization's normal security entitlement approval system and automatically implemented throughout the IT environment by software for managing SSH authorized keys and/or other trust relationships.  Automation can also reduce human errors and radically reduce the number of administrators requiring root access on servers.  (See Section 8.1.)

In some environments it may be desirable to prohibit public key authentication for interactive logins to ordinary user accounts. This can help enforce ordinary interactive logins to go through a privileged access management system (unless some administrators have copied private keys, which is generally possible).

## 5.1. Discovering SSH Keys and Trust Relationships

The purpose of the discovery phase is to obtain reliable and reasonably complete information about configured SSH keys and trust relationships throughout an IT environment.  Discovery should ideally include all Unix/Linux systems, Windows systems (at least those running SSH servers, SSH clients, or file transfer solutions running SFTP), Mac servers, workstations, laptops, mainframes, and other systems using the SSH protocol, including file transfer solutions using SFTP, virtualization platforms, and privileged access management gateways.

Since it is not possible to know what trust relationships exist in an IT environment without scanning all systems for SSH authorized keys and identity keys, all organizations SHOULD perform initial discovery of SSH user keys on all systems that use the SSH protocol.

Although some organizations may want to focus only on high-impact systems, a limited scope discovery process provides only limited visibility into the security risk of the current environment.  It is important to identify which low-impact systems can access high-impact systems, and this can only be known by scanning low-impact systems too.  An identity key intended to allow access between two high-impact systems could be copied onto a low-impact system.  Unless source restrictions are defined for the authorized key, the identity

key can be used from the low-impact system to access the high-impact system. Therefore, the scope of the discovery process SHOULD include all systems in the network, including even low-impact systems.

Ideally, routers, BIOS management ports, and other specialized computing devices should also be included, but at the time of this writing, software is not yet available for full SSH key discovery for these devices. This is expected to change in the future. It is also be possible to audit them manually.

The following MUST be determined during discovery:

Configured authorized keys for all user accounts on all servers of interest (accounts may be local, in LDAP, in Active Directory, in NIS, or any combination)

Configured identity keys on all user accounts on all servers and clients (workstations, laptops, etc) of interest. It should be understood, however, that one can never be certain that all identity keys have been found, because some could be copied into non-standard directories, stored offline, or even printed on paper. Thus not finding an identity key on a particular system does not guarantee that the key will not eventually be used from that system later. On a broader scale, even if the discovery process fails to find a particular identity key anywhere on the network, this does not necessarily mean that the key cannot be used later.

Configured restrictions for each authorized key, such as command restrictions and source restrictions

Established trust relationships (source host, source account, destination host, destination account, and restrictions)

The following SHOULD be determined during discovery (these may become MUST in the future):

Kerberos-based trust relationships for automated access, particularly access using keytab files, cached tickets, or service processes holding tickets.

Implicit trust relationships arising from Kerberos single sign-on.

Implicit trust relationships arising from sharing the same home directory across multiple accounts. Many organizations share the same home directory for multiple accounts. Adding an authorized key for any of the accounts implies adding it to all of the accounts. Thus, any accounts that can write to the shared home

directory effectively have access to all accounts sharing the home directory.

Trust relationships configured using host-based authentication (".shosts", ".rhosts", "hosts.equiv", or "shosts.equiv").

The following MAY be determined during discovery (some of these may become SHOULD or MUST in the future):

Trust relationships configured using password authentication (whether hard-coded in scripts or in password vaults). (In practice it may be difficult to do this reliably. However, it may be possible to, e.g., recognize certain syntactic patterns and commands from scripts as using hard-coded password authentication.)

Implicit trust relationships configured using "sudo" or some other privilege escalation tool.

Implicit trust relationships arising from user accounts that have NFS-mounted home directories. NFS is usually not configured to provide security against network-level attacks, and an attacker who gains access to a network segment may be able to read and modify the NFS traffic of any host on the network and impersonate any other host or user on the network (including reading and modifying any file on NFS file systems). When home directories are stored in NFS, a sophisticated attacker with root-level access to any device on the network (e.g., any unconfigured smart switch where the attacker can replace the firmware) may add new authorized keys to any home directory in an NFS file system.

Implicit trust relationships arising from user accounts that have home directories on a Windows share. Windows file sharing (CIFS, Common Internet File System) may suffer from same issues as NFS, and thus the same considerations may also apply to it.

It is important to understand that even though the discovery process finds keys, and in the short term most trust relationships are configured using SSH keys, the primary concern is not with the keys themselves or the underlying cryptography. Rather, the primary purpose of discovery is to determine who (or what) can access what and how such access is restricted. In terms of cryptography, all SSH keys created with default settings since version 1.0 use the equivalent of at least 1024 bit RSA key, which is still relatively safe, especially since servers never disclose authorized keys (cryptographic attacks on the key would first require access to an authorized keys file, which usually already means access to the host). Thus verifying key sizes or algorithms is not critical

(though may be required by policy); however, knowing who (or what) can access what is critical and addresses a real security concern. SSH user keys grant access and are fundamentally authentication credentials, rather than encryption keys. The whole issue is fundamentally not an encryption key problem, but an identity and access management problem!

Doing discovery properly is complicated. At least the following aspects need to be properly considered when planning discovery:

> SSH user keys cannot be discovered by a network scan because the SSH protocol was designed not to reveal authorized keys. It is possible to query whether an already known key is acceptable for a particular user, but an SSH server will not reveal an authorized key that is otherwise unknown. This means that the discovery process will need to connect to each host and access the authorized keys through the file system. (Host key discovery, on the other hand, is possible over a network, but host keys are beyond the scope of this document.)
>
> Different SSH versions are commonly deployed. Many large organizations have some combination of OpenSSH, Tectia SSH, SunSSH, Reflection for Secure IT, and various other products. Not all SSH implementations use the same key formats, store SSH keys in the same locations, or use the same key fingerprint format. Furthermore, OpenSSH comes in many flavors and patch set combinations, and some vendors pack a version of OpenSSH with another product - sometimes without providing a proper way of identifying the particular version. The discovery process (and tools) should be able to properly analyze keys and trust relationships for any SSH version that is deployed in the environment.
>
> Many organizations use the "root_squash" option for NFS exports, which converts file system accesses by root to accesses by an unprivileged account "nobody". A side effect of this is that a key discovery process running as root may not be able to read SSH keys in NFS home directories.
>
> Systems using SELinux may not allow reading SSH authorized key files or identity key files by ordinary processes. Reading such files may require special authorization or the use of special programs, such as "ssh-keycat".
>
> In a large environment, some servers are down for maintenance at any time, and in a geographically dispersed organization some networks may not be reachable at the time the discovery is initially run. Thus, discovery cannot be assumed to succeed on

all servers the first time. This problem is compounded for discovery of SSH clients since laptops may be disconnected from the network and therefore be unreachable for scanning.

### 5.2. Moving Authorized Keys to Protected Locations

Moving authorized keys to protected locations, or locking down authorized keys, MUST be performed on all moderate-impact and high-impact information systems (including systems having automated access to such systems). It MAY be performed on low-impact information systems.

Moving authorized keys to a protected location may be performed, e.g., by copying authorized keys for each user to a root-owned directory, and modifying the system-wide SSH server configuration file to specify the authorized keys file path (typically using a pattern that refers to a user name).

Failure to move authorized keys to protected locations allows system administrators and other users with legitimate access to create new trust relationships as unaudited backdoors and makes ensuring termination of access very difficult. Locking down authorized keys files helps to enforce the requirement that new trust relationships be properly approved. (See Section 5.3 for discussion of using an approval process for setting up new trust relationships.)

It is important to lock down authorized keys files early in the remediation process to create a stable environment for discovery. Otherwise, inappropriate authorized keys that continue to be added may not identified during the discovery process.

Similarly, authorized keys MUST be moved away from home directories susceptible to active network-level attacks (e.g., unencrypted NFS and CIFS home directories - in practice this includes most NFS home directories today) on all moderate-impact and high-impact systems (including systems having unrestricted automated access to such systems). It is RECOMMENDED that the same be performed on low-impact information systems.

Failure to move authorized keys away from NFS and CIFS home directories may allow a network-level attacker (whether human or automated) to add new authorized keys to any account that accepts authorized keys from such a directory, permitting unrestricted access to such accounts. Such attacks are known to have been performed by some penetration testers and are certainly within the capabilities of Advanced Persistent Threat (APT) groups.

Furthermore, identity key files SHOULD be moved away from home directories susceptible to network-level attacks (e.g., unencrypted NFS and CIFS home directories) on all moderate-impact and high-impact systems.  Otherwise, an attacker who gains privileged access one one host on the network may be able to read identity keys from user's home directories and use them for attack (this technique is known to have been used by attackers).  It is RECOMMENDED that the same be performed on low-impact systems.

Failure to move identity keys away from NFS and CIFS home directories may allow a network-level attacker (whether human or automated) to obtain copies of identity keys for later use or for immediately furthering the attack to other systems.  It substantially increases the risks associated with leaked keys and substantially expands the group of people who may be able to obtain copies of identity keys. Some penetation testers are known to use this technique for attacks.

## 5.3.  Monitoring Use of Trust Relationships and Keys

After the initial discovery phase, the environment SHOULD be monitored for some time (preferably several months) to collect data on how authorized keys are actually used.  While the process can be performed manually in a small environment, use of scripts or commercial tools is highly recommended in larger environments. Software tools can gather and correlate log data from many hosts to determine the following types of information: which keys are currently being used, which source hosts they are used from, which keys are external keys, and what commands they are used with.  This information about the use of trust relationships will help the organization in later phases of remediation, such as deciding which authorized keys will be removed for non-use (see Section 5.4) and which keys can be easily configured with command and source restrictions (see Section 5.8 and Section 5.9).  In addition, information gathered during monitoring will help the organization understand automated access patterns in the existing environment and further evaluate the concrete risks.

Monitoring use of trust relationships may involve configuring SSH servers and clients to use a higher level of logging and collecting and analyzing log data.  To determine which authorized keys are still being used, for example, an organization can configure SSH servers with a log level that causes the fingerprints of keys used for public key authentication to be logged, collect such log data for an extended period (several weeks to an year), and analyze the data to determine which authorized keys were actually used during the log collection period.

On SSH clients, identity keys that have not been accessed for a long time (according to file system's file access timestamps) are also good candidates for removal. However, many programs and commands may access identity key files, and a recent access time does not necessarily mean that the key was actually recently used for authentication. Furthermore, the identity key file timestamp provides no information regarding the destination host for the last connection. An identity key may still be in use for some destination host where it is authorized but not for another.

It should be noted that orphaned keys (authorized keys without a corresponding identity key) may be either unused or external keys. Thus they SHOULD NOT be removed without monitoring, as if they are external keys, trust relationships with hosts outside the managed environment could be inadvertently broken.

From a project management perspective, the monitoring period can well be used for assigning impact levels to systems, defining internal boundaries, and defining host groupings. In practice in large environments, different parts of the IT environment may be at different stages of remediation during the project. However, some of the remediation steps require a reasonably complete picture from longer-term monitoring before they can be safely performed.

Identifying trust relationships crossing certain boundaries, such as access from test and development systems into high-impact production systems, is of high interest to auditors and security managers. Detecting and controlling such unwanted access is an important audit objective. This information generally becomes available with reasonable certainty during the monitoring phase, after internal boundaries have been configured and impact levels for information systems determined.

Information collected during the monitoring stage will be helpful in later stages of a remediation project. The information gathering takes some time to get a reliable picture. It is RECOMMENDEED that a sufficient period of time be given for monitoring use of keys: at least 3-6 months or even a year. The remediation project should not be rushed, as it increases risk of having incomplete information about existing trust relationships or external keys, which in turn increases the risk that remediation activities may disrupt operations.

Failure to perform the monitoring step properly increases risk of disruption when unused keys are removed (see Section 5.4), and may even make removing unused keys impossible (one cannot remove unused keys without knowing with a high degree of certainty which keys are unused). Relying solely on the knowledge of application teams and

managers to identify unused keys is generally insufficient due to the large number of legacy trust relationships, personnel changes, and poor documentation of trust relationships.

Failure to perform the monitoring step properly also risks missing some external trust relationships and external keys. This may cause key rotation (see Section 5.7) to break external connections with systems outside the managed environment, such as data transfers with suppliers, contractors, distributors, or regional offices.

### 5.4. Removing Trust Relationships That Are No Longer Used or Otherwise Inappropriate

Various security standards and prudent information security require that access to information systems must be properly terminated when it is no longer needed. If trust relationships for automated access are left enabled on systems when no longer needed, they accumulate. Real-world experience has shown they sometimes accumulate at the rate of dozens of incoming trust relationships per year per system, even in security-sensitive environments.

Therefore, all organizations MUST remove trust relationships leading to moderate-impact or high-impact information systems (including low-impact systems having automated access to such systems) that are no longer needed as part of the initial remediation process. Unused trust relations leading to low-impact information systems SHOULD be removed.

Unused keys SHOULD NOT be removed until it is known with reasonable certainty which keys are really unused. This is usually accomplished by monitoring key usage over a period of time (see Section 5.3). It is further RECOMMENDED that unused trust relationships be reviewed by respective application owners to reduce the possibility of disruption from removal of a trust relationship that is actually needed. It is important to test infrequently used functionality, such as disaster recovery systems, reasonably soon after removing unused trust relationships.

It is also likely that the remediation process will identify many trust relationships that are still being used but that have no legitimate business purpose (see Section 5.5), that cross configured boundaries in inappropriate ways, or that lead to accounts (e.g., root) that should not be accessible. Such inappropriate trust relationships should also be removed (and alternate steps taken to implement any functionality they support in a more appropriate way, if required). Some of such trust relationships may have been created by attackers and may warrant further forensics investigation, such as identifying when they were created and by whom.

Failure to remove authorized keys for unused trust relationships increases the risk that key-based attacks for unauthorized access may succeed and spread throughout the network, allows previously created unaudited backdoors (using keys that are not regularly used) to remain in existence, and allows leaked keys (that are not regularly used) to remain usable indefinitely (if not rotated).

### 5.5. Associating Trust Relationships with Application and/or Purpose

Because authorized keys provide access to systems, their existence should be understood, justified, and controlled just like any other form of access control. After trust relationships that are not used have been removed, it is important to analyze the remaining active trust relationships to distinguish those authorized keys that support a valid application or business purpose and those keys that do not. Just because a trust relationship is being used does not actually guarantee that it is needed or legitimate. It may be used, e.g., by an attacker, by a user who created an inappropriate authorized key as a backdoor for access, or by a stale cron job relating to a decommissioned application. Determining the purpose of trust relationships is important for detecting such illegitimate trust relationships so that they can removed (see Section 5.4).

After having removed unused authorized keys, the existence of every remaining incoming trust relationship MUST be justified for moderate-impact and high-impact information systems (including low-impact systems having access to such systems).

This is an area where the justification can be more lax on low-impact systems. However, many low-impact systems, such as those used for internal software development or packaging, may generate binaries or distributions that later get installed on production systems. An attacker could use such systems to gain access to production servers, especially in an Advanced Persistent Threat (APT) scenario. It is thus RECOMMENDED that even access to low-impact systems be prudently justified, or alternatively systems with data/code paths to production be treated as moderate-impact or high-impact systems.

One practical approach for low-impact systems may be to review discovered incoming trust relationships in bulk (perhaps for a group of hosts belonging to the same business process), and label them as legacy trust relationships relating to the associated business process. While not ideal, such an approach may make a reasonable compromise between cost and security in many environments.

It is RECOMMENDED that each trust relationship be associated with the business process and/or application that it supports, and that the application/business purpose be documented for future reference.

This will help with removing trust relationships when applications are replaced and business processes re-engineered, and serves to assign responsibility for each trust relationship to somebody (the business process or application owner). Assigning trust relationships to applications or business processes effectively assigns ownership for the trust relationships (and related authorized keys) to the application or business process owner (or group). This owner MAY be permitted to approve new trust relationships leading to the same account on the same host, MAY schedule rotation of keys, and MAY be asked to periodically validate the existence of each trust relationship relating to the application or business process.

Failure to associate trust relationships with a purpose, business process, or application means that there remains access to information systems without reason or justification. Illegitimate backdoors may remain unnoticed and unnecessary trust relationships may remain in place that can be used by attackers, especially if keys are leaked (and not rotated).

### 5.6. Implementing Approval Process for Setting Up New Trust Relationships

Real-world experience has shown that many enterprises do not have a well-defined process for approving new trust relationships for automated access, and almost no enterprise today systematically enforces or audits approvals for automated access.

Organizations MUST implement an approval process for ensuring the validity of new trust relationships granting access to moderate-impact and high-impact information systems. It is further RECOMMENDED that organizations implement an approval process for trust relationships granting access into low-impact systems. This reduces risk of low-impact systems being used for staging attacks into high-impact systems. See also Section 8.1 for ideas on how automated setup of approved trust relationships can reduce costs.

New trust relationships SHOULD NOT be approved without a proper justification and association with a business process, application, or other valid purpose. In addition, the approval for new trust relationships MUST specify any command or source restrictions that should be implemented to limit security exposure. (See Section 5.8 and Section 5.9)

The approval process SHOULD carefully review whether trust relationships violate internal boundaries, such as allowing access from test or development systems into production or allowing access from low-impact systems into high-impact systems. Documented justification for crossing boundaries and secondary approval by

higher-level management SHOULD be required for such trust relationships. Organizations MAY also use the approval process to enforce other internal boundaries, such as those between business units or functions, or "Chinese walls" between, e.g., retail banking and investment banking.

Special approvals SHOULD also be required for trust relationships leading to root accounts or other highly privileged accounts on moderate-impact and high-impact systems.

The approval for each such trust relationship MUST be documented and MUST be retained for later audit. Approvals MUST be organized so that it is possible find the approval for each new authorized key.

Required approvals for new authorized keys MUST be enforced so that users cannot bypass the approval process. Enforcement typically involves both continuous monitoring and securing authorized keys files:

> Continuous monitoring (as discussed in Section 6) is needed to detect trust relationships that were implemented without approval. Existing trust relationships must be regularly audited against approved trust relationships. This requires periodically re-performing discovery to find all existing trust relationships so that they can be compared against a database of approved trust relationships. If software tools are used to perform this auditing, enforcement may be performed in real time or very frequently, e.g., once an hour or once per day.
>
> Another way to help enforcement is to move all authorized keys to protected locations (as discussed in Section 5.2) and tightly control access to root accounts using a privileged access management system (preferably one that also logs key-based access to root accounts for accountability). However, regular audits should still be performed, e.g., annually, to catch any trust relationships that may have been missed in the normal process.

Although the focus of this section has been on approving new trust relationships, existing legacy trust relationships should also be approved, or at least associated with a business process, application, or proper purpose. This was discussed in Section 5.5.

Failure to implement or enforce approvals means it is impossible to ensure that new trust relationships are valid and appropriately restricted. Not knowing what each trust relationship is used for makes it very difficult to know which trust relationships can be removed without substantial risk of disruption to business processes. Lack of up-to-date documentation of trust relationships, including

lack of knowledge of which application or business process they relate to, is one of the main causes of the current poor situation with SSH user keys in many organization.  Failure to implement and enforce approvals for trust relationships also implies that system administrators can continue to create unaudited backdoors to production systems, bypassing most existing privileged access management systems.

## 5.7.  Rotating Existing SSH User Keys

SSH user keys are authentication credentials, like passwords.  They should be rotated (i.e., changed) regularly.

Rotating an SSH user key for a trust relationship means generating a new identity key (key pair), storing (and configuring, if applicable) the identity key for the source account of the trust relationship, configuring the corresponding public key as an authorized key for the destination account (with the same restrictions as the old key for the trust relationship), and finally removing the old authorized key from the destination account and the old identity key from the source account.  If the same identity key can access more than one destination account (i.e., is used for more than one trust relationship), then it the authorized key must be copied to (and the old key removed from) all such destination accounts.

Rotating external keys (i.e., keys used with hosts outside the managed environment) require special care and coordination between the organizations responsible for the respective hosts.  The basic principle is that the new key should be added as an authorized key on all destination hosts where the old identity key is used before the old identity key is removed.

Authentication credentials for all trust relationships leading to moderate-impact and high-impact systems MUST be rotated every 12 months, and it is RECOMMENDED that trust relationships leading to low-impact systems be rotated every 12 months.  It is recommended that all keys be rotated as part of a remediation process to ensure that any previously leaked keys cease to be usable.

If two or more users have had access to a shared account that has access to an identity key, the identity key and any trust relationships using it and leading to moderate-impact or high-impact systems MUST be rotated during the remediation process and thereafter every three months.

If an employee leaves or changes roles, immediate rotation for all identity keys the employee is known to have accesss to and leading to moderate-impact or high-impact systems SHOULD be triggered.

If a security breach is suspected, all identity keys stored on affected servers SHOULD be immediately rotated.

If certificates are used for access, such certificates MUST be renewed (with new private keys) annually if they can be used for accessing moderate-impact or high-impact systems.  If Kerberos is used for configuring trust relationships, then the Kerberos credentials used for authentication MUST be rotated annually if they can be used for accessing moderate-impact or high-impact systems.

Failure to rotate keys allows leaked keys to continue working forever.

Failure to rotate keys in response to an employee leaving or changing roles means that there is no proper termination of access.  Many industries must comply with mandatory regulations that require proper termination of access.

Failure to rotate keys in response to a suspected breach means that keys copied by the attacker may be used to attack the systems again, and there can be no guarantee that the system has been properly cleaned up after the attack.

### 5.8.  Configuring Command Restrictions on Authorized Keys

Command restrictions limit what can be done with a trust relationship on the destination host.  Typically, a command restriction (also called "forced command") specifies the only command that can be executed on the server using that key.  If any other command is attempted, the configured command will be executed instead or the attempt is rejected.

A command restriction may further limit directories that can be used for file transfers (if supported by the SSH implementation) and whether writing files is allowed.

On some implementations, it may be necessary to prevent pseudo-tty allocation for command restrictions to be effective.

It is usually desirable to prevent TCP/IP forwarding for all authorized keys.  Otherwise such keys could be used to mask the source of attacks by redirecting them using port forwarding.

All non-interactive trust relationships leading to moderate-impact or high-impact information systems MUST be configured with a command restriction, unless an exemption has been approved as specified in the organization's security process and based on a valid reason for not having a forced command restriction (the relatively small effort

of configuring the command restriction not being a valid reason). The specific command MUST be part of the approval, and a new approval MUST be required if the command is later changed.

Trust relationships that are used for interactive access SHOULD NOT have a command restriction (command restrictions that permit running a shell and then arbitrary commands SHOULD NOT be used, because they may be mistaken as real command restriction; if they are detected in an audit, they SHOULD be flagged).

Regardless of impact level of the destination system, all trust relationships intended for use with the SFTP protocol by external parties or by lower-impact information systems MUST have a command restriction that limits the use of the trust relationship to SFTP and prevents interactive use.

Failure to configure command restrictions for keys increases virus spread risk and can be used for other attacks.  It also increases risk from leaked keys.

Failure to configure command restrictions for trust relationships used with external parties may allow a virus or attack to enter the organization.

### 5.9. Configuring IP Address Restrictions on Authorized Keys

Source restrictions (also called "from" option in authorized keys files) specify from which IP addresses an authorized key can be used.

Trust relationships permitting interactive access to moderate-impact and high-impact systems SHOULD specify a source restriction to hardened jump servers (privileged access management systems) or a transparent access auditing solution SHOULD be used to ensure such access is properly controlled and audited.  If any such trust relationships have been approved, they MUST be listed in an annual audit report and their existence rejustified annually.

Source restrictions SHOULD be used for all trust relationships leading to high-impact systems.  Otherwise, the use of source restrictions is OPTIONAL.  They are laborious to configure manually and make, e.g., IP renumbering and IPv6 transition painful.  It is also easy to make mistakes where, e.g., a secondary server for some critical service is not permitted by a source restriction, which could increase risk of outages under unusual operating conditions. On the other hand, they can significantly reduce the exploitation potential of leaked keys.

Failure to configure source restrictions has only mild security impact if other recommendations are followed. It is in part compensated by regular key rotation that also reduces the potential for exploitation of leaked keys, and thus a reasonable balance may be to not implement source restrictions for most trust relationships. However, source restrictions can completely prevent the exploitation of leaked keys (without sophisticated active network-level IP spoofing attacks), and thus is warranted for high-impact systems.

## 6. Continuous Monitoring and Management of SSH Keys and Automated Access

The remediation process (as discussed in Section 5) addresses both the one-time analysis and clean-up of existing legacy SSH trust relationships and the implementation of an ongoing approval process for validating, documenting, and restricting new trust relationships that are added to the environment. Following the approval process (discussed in Section 5.6) for all new authorized keys added to the environment serves as a preventive control. Continuous monitoring of trust relationships is needed to provide ongoing detection of non-compliance, including instances where the approval process was too lenient or was bypassed altogether. Continuous monitoring is also important for identifying trust relationships that violate policy, that can be removed because they have become unused or otherwise not needed, or that require keys to rotated.

### 6.1. Continuous Monitoring of Changes to Trust Relationships

Proper management of automated access requires continuous monitoring of the IT environment because system administrators operating as root may add new trust relationships for any user account. Continuous monitoring is also prudent for detecting keys that are no longer used, identifying external keys, and identifying changes in the patterns of usage of automated access.

The main rationale for the continuous monitoring of the environment and annual audits and requiring reporting and revalidation of certain aspects of automated access annually is to enforce proper policy (policies usually do not get implemented if their implementation is not enforced or if waivers are too easily available). However, IT environments are complex and sometimes there is a need to have automated access relationships for special purposes that would not otherwise be advisable. Special waivers and corresponding approvals can be used for implementing such special cases, but they MUST be revisited annually and MUST NOT be used to circumvent remediating the existing environment.

Ideally, continuous monitoring should be a real-time or near-realtime process.  For some areas, hourly or daily analysis would generally be perfectly sufficient.  Using automated tools allows monitoring to be performed more frequently, cost-effectively, and more thoroughly.  On the other hand, if implemented manually using audits, cost constraints may limit continuous monitoring to annual audits.  Even when continuous monitoring is performed using software tools, auditors SHOULD do some random sampling and testing annually to verify that the continuous monitoring tools are working properly.

In some respects, continuous monitoring resembles re-performing discovery on an ongoing basis.  Configured SSH user keys and trust relationships throughout the environment need to be discovered, and checked for validity.  Alerts, audit findings, or reports may be produced based on the results of the checks.  As in the discovery phase, the continuous monitoring process MUST identify every trust relationship and authorized key throughout the managed IT environment so that they can be compared against a database of approved trust relationships.

For each found authorized key, the trust relationship should be analyzed to identify possible instances of non-compliance or excessive security risk.  Trust relationships leading to moderate-impact or high-impact hosts with the following attributes MUST be reported for further investigation:

Trust relationships without proper approval

Trust relationships without proper justification and an associated application/business process

Trust relationships that have no command restriction configured

Trust relationships with command restrictions that do not match the command restrictions specified during approval

Trust relationships from low-impact hosts with no command restrictions

Trust relationships that cross defined internal boundaries

Trust relationships that have not been used in the last 12 months (or other time period specified by policy)

Trust relationships whose keys have not been rotated in the last 12 months (or other time period specified by policy)

Trust relationships leading to low-impact hosts with the following attributes SHOULD be reported for further investigation:

Trust relationships without proper approval

Trust relationships without proper justification and an associated application/business process

Trust relationships leading to privileged accounts that have no command restriction configured

Trust relationships that have not been used in the last 12 months (or other time period specified by policy)

Trust relationships whose keys have not been rotated in the last 12 months (or other time period specified by policy)

If trust relationships have existing waivers (e.g., for having no command restrictions, crossing boundaries, or not being used or rotated), then special approval of the waiver MUST be verified and waivers SHOULD be re-justified and approved annually.  Trust relationships that are flagged by continuous monitoring MUST be investigated and resolved.  Possible resolution activities consist of the following:

Obtaining approvals and justifications (including the associated application/business process) for trust relationships that are valid, including getting secondary approval by higher-level management for trust relationships that cross boundaries.  This would retroactively apply the approval process described in Section 5.6.

Adding command restrictions to the authorized keys file to limit access according to policy.  (See Section 5.8)

Removing trust relationships that are unused, not needed, or otherwise invalid.  (See Section 5.4)

Rotating private keys.  (See Section 5.7)

Obtaining waivers with appropriate levels of approval.

Even if waivers are obtained, the resulting risk needs to be considered.  For example, if the trust relationship from a low-impact host to a medium-impact or high-impact host has inadequate command restrictions, then the low-impact host MUST be reclassified as having the impact level of the higher-impact host, even if a waiver is obtained.

Failure to monitor SSH trust relationships prevents the organization from enforcing policies related to SSH user keys. Policy enforcement and detection of non-compliant trust relationships is needed to prevent new keys from re-creating the same type of problems that existed in the legacy population of user keys. Failure to enforce approvals for newly-added trust relationships allows users to create unaudited backdoors or trust relationships that cross boundaries or are unrestricted. If there is no continuous monitoring for unapproved or inappropriate trust relationships, such trust relationships will be essentially permanent.

### 6.2. Removal of Trust Relationships

Trust relationships MUST be removed when they are no longer needed. Ideally, the business or application owner of a trust relationship SHOULD expressly request that it be removed as soon as it is no longer needed. In addition, the owner MAY periodically recertify and validate the continuing need for each trust relationship.

Sometimes a trust relationship may be removed by express request, e.g., when a business process is changed so that it is no longer needed.

Sometimes a trust relationship may be removed because the application or business process it relates to is decommissioned or replaced by another application.

Sometimes a trust relationship may be removed because continuous monitoring detects that it is no longer being used. This basically implies that something changed in the environment, but the trust relationship was inadvertently not removed at that time. (This scenario appears to be very common in practice). In addition, some trust relationships may be removed because continuous monitoring detected an unapproved or otherwise invalid trust relationship.

When trust relationships are removed, the associated authorized key (if it is key-based) MUST be removed from the authorized keys file of the destination server.

When there are no trust relationships remaining using a particular identity key, the identity key SHOULD be removed.

### 6.3. Periodic Rotation of Trust Relationships

Keys must be regularly rotated as specified in Section 5.7.

## 7. Policy Recommendations

Effective security policies are important for defining expectations for controls and acceptable user behavior.  Well-defined policies are no less important for governing SSH user keys than for other elements of an organization's security program.  In fact, because few people understand the problem and poor SSH user key management practices are so pervasive, policies are essential to the success of any SSH key management remediation process.

To support the key remediation and continuous monitoring steps outlined elsewhere in this document, there is a common core set of policy statements that should be adopted by all organizations.  The following policy statements are RECOMMENDED (with limited organizational-specific customization and optionally limited to apply to moderate-impact and high-impact systems) to provide the governance framework for controlling SSH user keys:

All SSH servers shall be configured to store authorized keys in a root-owned /etc directory (or other suitable directory not writable by normal users).

Users shall not create new identity keys or authorized keys, shall not share identity keys with other users, and shall not copy or move identity keys to other SSH client systems.

SSH identity keys and authorized keys shall be provisioned only by the access management group.

SSH user key requests shall follow the standard provisioning process.  All requests for SSH authorized keys shall be provisioned only when required by a valid business need and approved by the destination account's owner.

Trust relationships shall not cross security zone boundaries.  If this is a requirement for a trust relationship, then the new user key request shall provide a rationale and require a waiver approved by the server operations director and information security director.

Trust relationships shall not allow access from low-impact systems to higher-impact systems.  If such trust relationships are required, then those low-impact systems shall be reclassified as higher-impact systems and shall be subject to the higher security requirements of higher-impact systems, unless command restrictions prevent obtaining an interactive shell and writing arbitrary files using such trust relationships.

Trust relationships for non-interactive access shall be configured with command restrictions.  If commands cannot be restricted, then

the new user key request shall provide a rationale and require a waiver approved by the server operations director and information security director.

Trust relationships permitting interactive access (especially to privileged accounts) shall enforce source restrictions to authorized, hardened jump servers or transparent access auditing solutions are used that ensure such access is properly controlled.

A registry of SSH user keys shall be maintained for tracking trust relationships (including their owner, purpose, approval, restrictions, and business purpose) throughout the environment.

SSH user keys and corresponding trust relationships shall be removed when no longer needed or no longer used.

Usage of SSH user keys shall be tracked so that unused authorized keys can be identified.

All SSH user keys shall be rotated annually.

When a user terminates employment or transfers to new job responsibilities, all keys assigned to that user shall be rotated, or the corresponding authorized key relationships shall be removed.

If a key is compromised or shared by two or more users, then the key shall immediately be rotated, or the corresponding authorized key relationships shall be removed.

SSH authorized keys shall be revalidated annually by the destination account owner to ensure that trust relationships continue to be valid and proper.

Authorized keys for privileged accounts such as root shall be revalidated annually and approved by the server operations director and information security director.

Trust relationships throughout the network shall be monitored at least annually to enforce compliance with this policy.  At a minimum, monitoring activities shall be in place to detect the following types of non-compliance for immediate resolution:

> SSH user key trust relationships that bypassed the formal provisioning process and were not authorized and configured by the access management group

SSH user key trust relationships that cross security zone boundaries

SSH user keys that have been not rotated in over a year

Dormant trust relationships that have not been used

Other policy statements are highly dependent on the risk tolerance and context of each organization. Depending on the unique circumstances of the organization, these policy statements may or may not be applicable. During the remediation process, organizations often make risk-based decisions about how to cost-effectively control and manage their SSH keys in their own context. It is critical that these decisions be properly reflected in security policy in order to influence user behavior and provide a framework for organization-specific controls. Examples of these types of policy statements are provided below:

All SSH user keys assigned to human users for interactive logins shall be assigned a passphrase that is at least 15 characters long. (The reason for this policy is self-evident.)

SSH trust relationships for human accounts shall be limited to other human accounts. Human accounts shall never have trust relationships to system accounts or service accounts. (This policy makes sense for organizations with lots of keys and transitive trust relationships that are too difficult to manage. Eliminating human-to-system account trust relationships can help simplify the mesh of trust and therefore minimize the risk of inadvertently allowing unneeded access.)

SSH user keys shall be used only for automated access and shall not be used for interactive logins by human users. (An organization may decide to do this to reduce the number of keys in the environment and lighten the load on the provisioning process, for example, if no automation is available and provisioning is done manually.)

SSH servers shall be configured to deny connections to the root account. (If key-based connections to root are not required, then setting "PermitRootLogin no" can significantly contain the damage that can be done through unauthorized use of keys).

Unique SSH host keys shall be created for every system. (This is essential when SSH host-based authentication is used and for protecting against man-in-the-middle attacks.)

## 8. Considerations for Software Tools

All requirements specified in this document can be implemented manually and with regular audits, without using software tools. Use of software tools is OPTIONAL. However, automated software tools for managing SSH keys are commercially available from multiple vendors and their use is RECOMMENDED in large environments, as they can substantially reduce the time, cost, and effort needed for remediating existing SSH user keys and provide substantial ongoing cost savings for continuously managing and monitoring SSH keys in an organization.

Here are certain key things to consider in planning an SSH key management remediation solution and its deployment:

> Does the solution support all required operating systems where SSH keys need to be managed (including mainframe, if applicable)?
>
> Does the solution support all SSH implementations and versions that are use in the environment, including their key formats and fingerprint formats?
>
> Does the solution support keys moved to protected, root-only-writable locations? Can it help move keys to such locations? How does it determine where the keys are stored on each host?
>
> Can trust relationships that are not actually used be automatically detected and proposed for removal (with selective approval)?
>
> Can the solution associate trust relationships and keys with an application, business process, or other purpose? Can it enforce that all authorized keys have a documented purpose? How is this implemented for legacy trust relationships (from time before deployment of the solution)? Can it distinguish legacy keys from those that are set up afterwards?
>
> How does the solution implement approvals for new keys? How does it integrate to existing workflows and tools? Does it support an approval workflow which integrates into external systems?
>
> Can creation of new keys and trust relationships be automated based on approvals done in an existing IT change control system? If no existing IT change control system is in use in the organization, does the solution provide one to enforce approvals?
>
> Does the solution support grouping systems based on the impact of their disruption or compromise?

Does the solution support rotating SSH user keys?

How is key rotation implemented for external trust relationships/ external keys?  Can it automatically recognize external keys?

Does the solution support configuring command restrictions for authorized keys/trust relationships?  Does it support requiring special approvals for trust relationships that do not have a command restriction?

Does the solution support configuring source restrictions for authorized keys/trust relationships?

Does the solution provide continuous monitoring capabilities as specified in Section 6?

If the management system is unavailable for some reason, will normal operation of managed hosts be disrupted (other than not being able to create/modify trust relationships)?

Will the solution run as root on managed hosts, or can it use a non-root account and "sudo" (or equivalent) to perform limited operations as root?

Is the solution able to retry discovery, key setups, etc.  on hosts that are down or unreachable at the time of the initial attempt?  How does the solution cope with poor network connectivity?

Does the solution support user accounts stored in LDAP or Active Directory?  How does it prevent crashing LDAP or Active Directory servers by reading directory contents from all servers simultaneously?

Can the solution discover keys from directories that are not readable by root (e.g., NFS directories using the "root_squash" option)?

Does the solution work with SELinux, if such support is needed?

How can the solution save operational costs in SSH user key management in the organization?  Have existing user key management costs been estimated on an annual level?

### 8.1.  Reducing Cost and Improving Security by Automation

Some large organizations are seeing over a hundred thousand new authorized keys being configured every year.  Some trust relationship

setups may involve installing the same authorized key on thousands of servers.  Given that setting up and a manual trust relationship can easily take fifteen minutes or more, the cost can be millions of dollars per year.

Some software tools allow integration into existing security entitlement approval systems, and can implement a suitably formatted trust relationship setup request automatically, without manual intervention.

Such automation provides several benefits:

Substantial cost savings by eliminating the manual work associated with trust relationship setups.

Substantial reduction in outages due to errors in manual key setups.

Need for root access is significantly reduced, as SSH user key setups no longer require root access.

Substantial security improvements from eliminating root access (or the need for being able to install new trust relationships) from most system administrators (having five people with access to the software tool system is much more secure than having two hundred administrators able to manually install keys).

## 9. Security Considerations

This document is all about security, including how to evaluate the impact of disruption or compromise of information systems, how to reduce the risk to information system from automated access, how to remediate current unmanaged base of SSH user key based trust relationships for automated access, and how to manage and continuously monitor automated access as an ongoing process.

Section 1.5 defined information system impact levels based on FIPS 199, but expanding on it by considering information systems having automated access to higher-impact information systems as also having the impact level of the higher-impact information system.

Section 2.2.6 briefly discussed unmanaged host keys and how they can be used to compromise authentication and integrity protection using active network-level man-in-the-middle attacks.

Section 3 discussed various threats arising from poorly managed automated access and SSH user keys, including virus spread threat, unaudited backdoor threat, leaked keys granting near-permanent

access, and lack of proper termination of access when an employee leaves or changes roles. It also discussed how ports opened in firewalls may be used for unintended purposes, including command execution, access to internal services, or for hiding source of attacks, if not properly controlled.

Section 4 discussed assessing the threats and exposure of an organization to them as a quick precheck during audit, before engaging in a full discovery and remediation project.

Section 5 provided recommendations on how to bring existing trust relationships for automated access, particularly SSH user keys, under control.

Section 6 provided recommendations for continuous monitoring and management of automated access and SSH user keys.

Section 7 provided recommendations for organizational security policy.

As a summary, automated access between systems MUST NOT be overlooked in identity and access management. It has become so prevalent that many organizations have many times more credentials for automated access to their information systems that they have user accounts for employees.

Management of SSH keys is about managing access, with strong ties to identity and access management, security architecture, privileged access management, IT change control, and security audits. Cryptographic properties of the keys are in practice of little importance, as all keys generated with default settings by most commonly used SSH implementations are still cryptographically reasonably strong.

Virus spread threat using automated trust relationships may remove defense in depth against attacks and malware. Automated access may provide pathways for bypassing existing privileged access management systems. Rogue administrators may use SSH user keys to create near-permanent unaudited backdoors, and leaked keys may be used for breaking into production servers. Even accidental access using poorly configured trust relationships has in the past caused substantial financial losses.

Risks of unmanaged, unaudited automated access are sufficiently high and the state of their management in some of the largest organizations in the world so appalling that all organizations should evaluate to what extent they use automated access within and between their information systems, how it is managed and audited, and whether they are exposed to the risks.

IT security auditors, policy makers, and security architects are urged to take automated access and SSH keys on their agenda.

## 10. Acknowledgements

The authors thank and acknowledge the contribution of the following people to the development of this document and/or the underlying ideas: Bruno Canamasas, Roman Hernandez, Jan Hlinovski, Kalle Jaaskelainen, Mitch Klein, Sami Lehtinen, Sami Marttinen, Matthew McKenna, S.  Moonesamy, Tim Polk, Joe Scaff.  We also wish to thank anyone else who has helped by providing comments or input.

## 11. Glossary

account:  A user account on a computer.  An account may belong to an actual person (interactive user) or may be used internally in a system (in which case it is sometimes called a functional account, process account, system account, or non-user account).

Active Directory:  A directory service created by Microsoft for Windows domain networks, providing a central repository for user information, user groups, and various other kinds of configuration information.  Active Directory makes use of the LDAP and Kerberos protocols, among others, and can serve as an LDAP directory and Kerberos Key Distribution Center (KDC).

Advanced Persistent Threat (APT):  A group, such as a government, with the capability and intent to persistently target an entity using a variety of cyberwarfare techniques, such as espionage, social engineering, custom malware, and sophisticated hacking and evasion techniques.

authorized key:  A public key that has been configured as authorizing access to an account by anyone capable of using the corresponding private key (identity key) in the SSH protocol.  An authorized key may be configured with certain restrictions, most notably a forced command and a source restriction.

automated access:  Access to a computer without an interactive user, generally machine-to-machine access.  Automated access is often triggered from scripts or schedulers, e.g., by executing an SSH

client or a file transfer application. Many programs may also use automated access using SSH internally, including many privileged access management systems and systems management tools.

automated trust relationship: A trust relationship for automated access.

command restriction: See forced command.

certificate: A public key signed by a certification authority (CA) key, together with additional information about the public key. X.509 [RFC3280] is a widely used standard for certificates. OpenSSH also implements its own proprietary certificate format; however, use of the proprietary format is NOT RECOMMENDED (in part because OpenSSH's authorization model does not permit reliably determining which trust relationships exist granting access to a server).

CIFS: Common Internet File System, a protocol used on Windows for file sharing. The protocol is unencrypted and may be read and subverted by a network-level attacker. The protocol is extremely widely used in Windows environments, less frequently with Unix/Linux.

CISO: Chief Information Security Officer. A person responsible for IT security in an organization.

COBIT: Control Objectives for Information and Related Technology, a framework created by ISACA (Information Systems Audit and Control Association) for information technology (IT) management and IT governance.

CryptoAuditor: A product from SSH Communications Security for controlling and auditing content of SSH sessions and other encrypted communications, including file transfers. Can also be used for auditing use of SSH/SFTP connections at a firewall and for privileged access auditing for key-based access.

destination account: In an SSH connection or trust relationship, the user account for which authentication is provided and under which any commands or other operations performed by the connection are executed (acknowledging that some commands, such as "sudo", may further escalate privileges).

destination host: In an SSH connection or trust relationship, the destination host of the connection. A destination host would typically run an SSH server.

DSA: Digital Signature Algorithm. An algorithm for public-key cryptography, particularly digital signatures. It is a United States government standard, specified in FIPS 186-3.

external key: An authorized key that is used from outside the organization (or outside the environment considered for SSH user key management purposes), or an identity key that is used for authenticating to outside the organization (or outside the environment considered for SSH user key management purposes). Key rotation can break external keys, and therefore it must be ensured that the other side of trust relationships involving external keys is also properly updated as part of rotation. Alternatively, rotation of external keys may be prevented, but that is not a sustainable solution long-term.

fingerprint: A hash value of a (public) key encoded into a string (e.g., into hexadecimal). Several fingerprint formats are in use by different SSH implementations.

FISMA: Federal Information Security Management Act of 2002, a United States law that mandates how US government agencies must implement their it security.

forced command: A restriction configured for an authorized key that prevents executing commands other than the specified command when logging in using the key. In Tectia SSH and OpenSSH, forced command can be configured by using a "command=" restriction in an authorized keys file.

functional account: An account used for running applications or other processes, as opposed to an interactive account normally used by a person. Functional accounts are sometimes also called process accounts, system accounts, or non-user accounts (with slight nuances in meaning).

host: A computer or other device on a network. A host may be a physical computer, a virtual machine, or any other logical or physical device that can communicate on a network, typically using one or more IP addresses. Some hosts may be multi-homed, meaning that they have more than one IP address.

host certificate: A certificate for a host key for host authentication in the SSH protocol (typically an X.509v3 certificate). Host certificates can eliminate the need for distributing host keys to all communicating hosts, greatly simplifying management and rotation of host keys. Widely used with Tectia SSH to avoid copying host keys and to make rotating them easier.

host credential:  A Kerberos credential that is used for authenticating to a Kerberos KDC as a host principal.

host key:  A public key used for authenticating a host in the SSH protocol to hosts that want to communicate with it (each host also generally has its own private host key).  Some hosts may have more than one host key (e.g., one for each algorithm).  Host keys are used for authenticating hosts (machines) themselves, not users or accounts, whereas identity keys and authorized keys relate to authenticating users/accounts and authorizing access to accounts on hosts.  See also Section 2.2.6.

identity key:  A private key that is used for authentication in the SSH protocol; grants access to the accounts for which the corresponding public key has been configured as an authorized key.

indirect trust relationship:  A sequence of trust relationships that indirectly leads to another account.  For example, account A may be able to log into account B, which may be able to log into account C; then, account C indirectly trusts account A (and B directly trusts A and C directly trusts B).  Indirect trust relationships may involve many kinds of trust relationships (e.g., SSH keys, Kerberos and privilege escalation).

interactive user:  A person (human) that uses a computer (and may type passwords or provide other authentication credentials as needed), as opposed to a computer that performs operations on another computer in an automated fashion.

jump host:  A server that a user logs into for the purpose of logging infrom there to another server.  They are used for privileged access management, centralizing configuration of access to a large number of servers (e.g., at retail locations), and for accessing restricted subnets that do not have normal routing from the rest of the organization.

KDC:  Key Distribution Center, a component of Kerberos.

Kerberos:  A centralized authentication and single-sign on system. Also used as part of Active Directory.  See RFC 4120 [RFC4120].

key:  A cryptographic key.  In this document, keys generally refer to public key cryptography key pairs used for authentication of users and/or machines (using digital signatures).  Examples include identity key and authorized keys.  The SSH protocol also uses host keys that are used for authenticating SSH servers to SSH clients connecting them.

Key Distribution Center: A component of Kerberos and Active Directory infrastructure that verifies credentials and issues tickets to principals (e.g., users and hosts). An Active Directory server includes a KDC. Frequently multiple KDCs synchronize information for redundancy.

known host: A host whose host key is known (to a particular SSH client).

LDAP: Lightweight Directory Access Protocol, a protocol for accessing and maintaining distributed directory information services. See RFC 4511 [RFC4511].

locking down keys: This refers to moving authorized keys to root-owned (or otherwise protected) locations, so that non-root users cannot add new authorized keys to themselves. This helps prevent system administrators and users from creating key-based backdoors that may survive the termination of their account and bypass privileged access management systems. See Section 5.2 for more information.

NERC: North American Electric Reliability Corporation, an organization that, among other things, maintains the Critical Infrastructure Protection (CIP) standards that set minimum security requirements for protecting power generation and distribution infrastructure.

NFS: Network File System, a file sharing protocol widely used in Unix/Linux environments in enterprises and universities. The protocol is unencrypted and may be subverted by a network-level attacker, permitting modification of any file. (NFS4 adds some security but is rarely used at the time of this writing, or is used with the security features disabled.)

OpenSSH: An open source implementation of SSH based on Tatu Ylonen's original free version of SSH from 1995 and further developed by the OpenBSD group.

orphaned key: An authorized key for which no corresponding public key can be found. An orphaned key may be currently unused, or the identity key might just be on a server that was not part of the discovery process (it could be an external key). Therefore orphaned keys should not be removed without first monitoring whether they are actually used.

password logger: A software or hardware module for recording keystrokes, especially user names and passwords, typed by an interactive user. Password loggers are nowadays commonly included

in various malware and used as part of Advanced Persistent Threat (APT) attacks.  Hardware-based key loggers may used in conjunction with physical access to a desktop or laptop (perhaps using a social engineering attack, such as getting hired as a janitor) to obtain passwords for entry into information systems.

PCI DSS:  A set of Data Security Standards defined by the Payment Card Industry Security Standards Council, an organization originally formed by major credit card companies.

PKI:  See Public Key Infrastructure.

privilege escalation mechanism:  A means for escalating a user's (or processes) privileges from those of one account to those of another account (particularly a root or Administrator account). Examples of privilege escalation mechanisms include intentional provilege escalation tools such as "sudo" and unintentional privilege escalation possibilities based on vulnerabilities and configuration errors (experience has shown that it is very often possible to find vulnerabilities or misconfigurations on that enable privilege escalation once inside a host).  An attacker having access to an account can generally change the configuration of the account to cause the user to unknowingly run the attacker's programs that may, e.g., steal the user's password and then use the password to spread the attack.  Also, having high-level access on one host on a network may effectively imply access to every user account on every host whose home directory is in networked storage accessible through the same network as the compromised host.  Against advanced attackers, even vulnerable embedded devices such as switches, printers and copiers can be used to perform network-level active attacks against other hosts.  Some limit will have to be put on what theoretical possiblities are considered, however.  Privilege escalation possibilities effectively imply additional trust relationships that may in turn imply a multitude of indirect trust relationships.

Public Key Infrastructure:  An arrangement that binds public keys with respective user identities using digital signatures issued by a certificate authority (CA).  See RFC 3280 [RFC3280].

Putty:  An Open Source SSH client for Windows.

Reflection for Secure IT:  A commercial version of SSH from Attachmate.

root account:  In Linux/Unix, a privileged account that is usually able to do anything in a computer (including reading any files and modifying any programs).  In Windows, Local Administrator and

Domain Administrator have similar or even broader power. (This document mostly talks about root access as SSH is mostly used on Linux/Unix and embedded devices, but the same issues often also apply to other technologies and the Windows environment.)

rotating a key: Key rotation means changing the key, i.e., replacing it by a new key. The places that use the key or keys derived from it (e.g., authorized keys derived from an identity key, legitimate copies of the identity key, or certificates granted for a key) typically need to be correspondingly updated. With SSH user keys, it means replacing an identity key by a newly generated key and updating authorized keys correspondingly. See also external key.

RSA: An algorithm for public-key cryptography based on the difficulty of factoring large integers, invented by Ron Rivest, Adi Shamir and Leonard Adleman.

SELinux: Security-Enhanced Linux, a Linux feature that provides mechanisms for supporting access control security policies. SELinux is enabled by default on several Linux distributions (at least in what is called "targeted" mode, where it protects selected services).

SFTP: SSH File Transfer Protocol, a file transfer and file sharing protocol typically used with the SSH protocol and originally developed by Tatu Ylonen for ssh-2.0. The protocol is unofficially described in SFTP [SFTP]; there is no normative reference available at the time of this writing.

source account: In an SSH connection or trust relationship, a source account is the user account on the host initiating the connection, typically the user account under which an SSH client runs.

source host: In an SSH connection or trust relationship, a source host is the host initiating the connection (typically by running an SSH client).

source restriction: A restriction configured for an authorized key that limits the IP addresses or host names from which login using the key may take place. In OpenSSH, source restrictions can be configured by using a "from=" restriction in an authorized keys file.

SOX: Sarbanes-Oxley Act of 2002, also known as the Public Company Accounting Reform and Investor Protection Act, a United States law that, among other things, sets requirements for protecting certain sensitive information in listed companies.

SSH:  SSH (Secure Shell) is a protocol and tool for remote system administration, file transfers, and for tunneling TCP/IP communications securely, originally developed by Tatu Ylonen.

SSH Communications Security:  A company founded by Tatu Ylonen, the inventor of SSH, with products improving security and operational efficiency of large IT environments, particularly for large SSH environments.  See http://www.ssh.com [2].

sudo:  A privilege escalation mechanism/tool on Unix/Linux that can be used for executing commands as root from a non-root account. The operation of "sudo" depends on its configuration.  In some configurations, certain accounts may perform any command as root using "sudo".  In some other systems, certain users, such as members of a "wheel" group can perform commands as root by confirming the operation with the user's password.  Several commercial tools also exist for the same purpose.

Tectia Manager:  A product for managing SSH host keys and configurations, from SSH Communications Security.

Tectia SSH:  A commercial version of SSH servers and clients for Windows, z/OS (IBM mainframes), Unix, and Linux from SSH Communications Security.

transparent access auditing:  A method of doing privileged access management and auditing on the network (using a co-operative man-in-the-middle attack to transparently gain access to the connection) or at an SSH server (by having auditing code built into the server).  See, e.g., the CryptoAuditor solution.

trust relationship:  Something that permits a source account to log in to a destination account (possibly on a different computer). In a sense, the destination account trusts the source account, and if the source account is compromised, so is the destination account.  An example is an authorized key (and corresponding identity key) configured for public key authentication in SSH. See also indirect trust relationship and privilege escalation.

Universal SSH Key Manager:  A product from SSH Communications Security for managing and monitoring SSH keys and other trust relationships for automated access.

user key:  A key that is used for granting access to a user account in the SSH protocol (as opposed to a host key, which does not grant access to anything but serves to authenticate a host).  Both authorized keys and identity keys are user keys.

X.509: A standardized widely used certificate format for public key infrastructure (PKI). See RFC 3280 [RFC3280].

## 12. References

[FIPS199] Evans, D. L., Bond, P. J., and A. L. Bement, "Standards for Security Categorization of Federal Information and Information Systems", FIPS Publication 199, February 2004.

[FIPS200] Gutierrez, C. M. and W. Jeffrey, "Minimum Security Requirements for Federal Information and Information Systems", FIPS Publication 200, March 2006.

[NIST800-53] Locke, G. and P. D. Gallagher, "Recommended Security Controls for Federal Information Systems and Organizations", NIST Special Publication 800-53 (revision 3 with updates as of 05-01-2010), August 2009.

[KENT] Kent, G. and B. Shrestha, "Unsecured SSH - The Challenge of Managing SSH Keys and Associations", SecureIT White Paper, 2010.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4251, April 2002.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4251, July 2005.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[RFC4253] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

[RFC4254] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.

[RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[SFTP] Galbraith, J. and O. Saarenmaa, "SSH File Transfer Protocol", draft-ietf-secsh-filexfer-13.txt (work in progress), June 2006.

## Authors' Addresses

Tatu Ylonen
SSH Communications Security

Email: ylo@ssh.com
URI:   http://www.ssh.com

Greg Kent
SecureIT

Email: gkent@secureit.com

Murugiah Soyppaya
NIST

Email: soyppaya@nist.gov