UNIVERSITY OF HELSINKI

https://helda.helsinki.fi

SSH Key Management Challenges and Requirements

Ylonen, Tatu

2019-06-24

http://hdl.handle.net/10138/307737

Ylonen, T 2019, SSH Key Management Challenges and Requirements. in 2019 10th IFIP International Conference on New Technologies, Mobility and Security : Proceedings of NTMS 2019 Conference and Workshop. International Conference on New Technologies Mobility and Security, IEEE, IFIP International Conference on New Technologies, Mobility and Security, Canary Islands, Spain, 24/06/2019. https://doi.org/10.1109/NTMS.2019.8763773

Downloaded from Helda, University of Helsinki institutional repository. https://helda.helsinki.fi
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.
Please cite the original version.

Challenges in Managing SSH Keys – and a Call for Solutions

Tatu Ylonen
SSH.ORG & University of Helsinki
ylo@ssh.com

Abstract

SSH (Secure Shell) uses public keys for authenticating servers and users. While SSH's key management design was great for grass-roots deployments, it is now causing significant operational and security challenges in large computing environments. This paper summarizes progress in SSH key management so far, highlights outstanding problems, and presents requirements for a long-term solution. Proposals are solicited from the research community to address the issue; so far the industry has been unable to come up with an ideal solution. The problem is of high practical importance, as most of our critical Internet infrastructure, cloud services, and open source software development is protected using these keys.

1 Introduction

SSH (Secure Shell) [28, 29] is an encryption protocol that is used for managing most Unix/Linux servers, Internet routers, and much of cloud computing infrastructure. It is also widely used for committing code to version control repositories and inside various systems management tools.

The SSH protocol provides host authentication using public keys associated with hosts, called *host keys*. These keys are used to prevent man-in-the-middle attacks, such as stealing passwords or command injection after authentication.

The protocol also uses public keys for user authentication. These keys are called *user keys*. While there are several other authentication methods available, public key authentication is the method of choice for automation. It is used, for example, for accessing Linux instances in Amazon AWS, committing source code into github, for automating systems management, and for single sign-on. We have recorded over five million daily logins using public key authentication in some large financial institutions.

We have made several alarming discoveries regarding user keys. First is their sheer number: many large financial institutions have *several million keys* authorized to log in across tens of thousands, sometimes over a hundred thousand servers. In some cases a single key can access tens of thousands of servers, such as for keys originally installed for emergency response, auditing, or patch management purposes. Some of these keys are very old.

Second, extensive monitoring of the usage of the keys has revealed that in several of the large financial institutions that team has worked with, about 90% of the authorized keys have been unused. That is, they are access credentials that were provisioned years ago, the need for which ceased to exist or the person having the private key left, and the authorized key was never deprovisioned. Thus, the access was not terminated when the need for it ceased to exist. This signifies a failure of identify and access management (IAM) practices.

In many organizations – even very security-conscious organizations – there are many times more obsolete authorized keys than they have employees. Worse, authorized keys generally grant command-line shell access, which in itself is often considered privileged. We have found that in the organizations we have worked with, about 10% of the authorized keys grant root or administrator access. SSH keys never expire.

Third, it is rare for organizations to have the capability to quickly replace SSH user keys even when they know there has been a breach and that private keys may have been compromised. In fact, of the dozens of large enterprises we have knowledge of, we estimate that only one or two might have been able to replace their SSH keys in reasonable time.

The final issue relates to attitudes and operations. Many organizations do not want to know. If they acknowledged the issue, they would have to do something about it, because all the major security regulations require controlling who has access to what systems and data. Bringing SSH keys under management is a substantial effort, and many companies kick the can down the road – even when there are literally millions of unaccounted-for credentials granting access to their systems.

There is an established process for managing SSH keys [30]. However, the current process is based on managing user keys in authorized keys files and only addresses access internal to an organization. One challenge is that SSH key

based access is often used by automated processes, while most IAM implementations only deal with access by people. Host key management is also not really addressed by the current process.

There has been extensive research with hundreds of papers on PKI (Public Key Infrastructure), but very little on SSH key and access management, even though it is arguably at least as critical for the core infrastructure.

The contributions of this paper include:

- detailing the SSH key management problem and its importance
- reviewing attempts to solve it so far, and
- providing requirements and design criteria for fully solving it.

2 Host keys

The SSH client uses the server's host key for authenticating the server. In OpenSSH, for example, host keys of known servers are stored in a *known hosts file*. When the user first connects to a server, he/she is shown the fingerprint of the server's host key and asked whether to accept it. On later connections, OpenSSH verifies that the key has not changed, and asks the user to accept the new key if it has changed. Both personal experience and available research [9] indicate that users generally accept unknown host keys without verifying them.

2.1 Problems with TOFU

SSH's original and still dominant approach to host key management is called TOFU (Trust on First Use) [31]. While this approach was great for grass-roots deployment, it has become a problem.

TOFU works well for small static environments with tech-savvy users, but offers little security in large dynamic environments. This has effectively led to a breakdown of SSH's man-in-the-middle attack protection and opened it up for various active network-level attacks, including password stealing, data theft, and command injection. Numerous free tools are available for performing these attacks.

According to research by Gutmann [9], users do not check host key fingerprints in practice. Our anecdotal experience supports this. Users do not understand the warnings about changed host keys and even for experts, verifying the keys is too cumbersome to do reliably. Furthermore, in a large enterprise hundreds if not thousands of new hosts are installed or reinstalled every day, leading to significant key churn.

The uniqueness, security, and validation of host keys is critical for man-in-the-middle attack prevention. We thus find ourselves in a situation where the host authentication in SSH is not reliably serving its function of preventing man-in-the-middle attacks. Thus the mechanism needs to be augmented or replaced by a mechanism that provides better security and smoother operations in large environments.

2.2 Attempts to improve host key management

The SSH protocol was originally designed to work without any centralized key infrastructure, as none existed for ordinary servers in 1995, and even today one really only exists for HTTPS servers. Also, the size of large IT environments has grown since then by orders of magnitude.

Automatically collecting SSH keys and distributing them to system-wide known host files within an enterprise was attempted in the early 2000s with the Tectia Manager product. The approach works reasonably well for up to a few thousand servers, but thereafter raw copying of host keys becomes unwieldy. First, the known hosts files are simple text files that are linearly scanned for the matching host key. With tens of thousands of keys, the files grow to many megabytes, and the cost of scanning through them on every login becomes prohibitively expensive. Furthermore, in a large environment of, say, 100,000 servers, there are easily 2000 servers reinstalled every day, typically with their host keys changing. The cost and complexity of distributing the keys becomes rather high, especially if regular key rotation is to be implemented (key rotation meaning changing the keys). Finally, many clients are laptops or mobile devices that are only sporadically connected to the company network and whose client implementations are not designed to work with very large numbers of known host keys.

Peng and Zhao [21] describes a method for initializing trust for network devices using SSH, where a central system manages and generates a list of known hosts and distributes it to the managed devices. However, in their system a human is assumed to verify the fingerprint of the central device.

Another attempted approach has been to store SSH host keys in DNS (Domain Name System) using SSHFP records [24]. Even though this is an IETF standards-track specification that has been around for over a decade, it has not seen extensive use. The approach relies on DNSSEC [3] for security, but DNSSEC is not yet very widely deployed in most countries. Also, due to the way DNS update permissions are structured in most organizations and lack of automation, updating fingerprints in DNS is often too cumbersome.

Jones [13] presents a layered approach that uses DNSSEC to store the location and public key of a key distribution server, but keys for individual entities are stored in the key distribution server(s). This securely moves individual host keys out from DNS, which is administratively much simpler to implement than SSHFP. Similar mechanisms could also enable automatic discovery of key management servers. It might be possible to extend this system for distributing authorized keys as well.

Redhat Linux supports using the `ProxyCommand` feature in

the OpenSSH client to fetch the host key for the host being connected to from a central registry into a known hosts file, and reading it from that file. However, this solution does not work with other SSH client implementations and only works with RedHat Linux servers.

Perspectives [31] was a proposal to use notary servers to track the history of host keys used by servers and return signed statements indicating which keys have been used by the server and when. This information could then be used for deciding whether to trust the key. We have not encountered deployments of this system.

Alicherry and Keromytis [2] discusses shortcomings of TOFU and possible attacks. They propose fetching the host key using a second means, such as using `tor` to fetch the host key using a different network path. One challenge with this approach is that it relies on heuristics and it is hard to analyze how much security is improved.

Ali and Smith [1] have proposed using a keyed MAC for host authentication. We have not seen this scheme used in practice with SSH.

Napier [20] suggests NFS (Network File System) mounting a centrally maintained known hosts file. We've not seen that used in practice but it is a possible approach to managing keys within an organization. Windows clients, however, generally do not support that approach, and it relies on the security of NFS.

Schechter et al [23] discusses harvesting IP addresses and names of hosts being connected to from known hosts files and ways to make it harder (e.g., for trojans and other attackers trying to spread to other machines using SSH keys). Their appendix also lists various potential host authentication attacks against SSH. They propose hashing host names and IP addresses in known hosts files, which has been incorporated into several SSH implementations. However, this hashing makes automatic management of known hosts files even more difficult.

Tectia SSH has long supported X.509 certificates for host authentication. This approach has been successfully deployed in various financial institutions; however, some organizations have found their X.509 management tools prohibitively expensive for such large numbers of certificates. Better tools for distributing configurations and managing the certificate life cycle would thus be desirable.

OpenSSH supports its own proprietary certificate format for host keys. We have not seen it used widely, mostly due to lack of good tools for issuing and distributing the certificates, even though some open source tools are available. The approach is also not interoperable with other SSH implementations.

Certificates are a possible practical solution for host key management. However, in practice it requires running an internal CA and proper tools to distribute and rotate the certificates. Using certificates for user keys is discussed separately and involves a different set of tradeoffs.

Many SSH implementations offer the option of preventing connections when the host key does not match a known key. However, strictly enforcing host keys can prevent access to servers and break business processes when a host key changes due to a reinstallation, and forcing key acceptance on users has not proven effective.

Based on working with dozens of of large enterprises, the vast majority of organizations today do not seem to have a system in place for managing SSH host keys. This significantly weakens the man-in-the-middle attack protections offered by SSH in practice.

To illustrate the problem, we have seen a very tech-savvy organization with 30,000 general-purpose hosts sharing the same host key. In Heninger et al [11], 65% of all host keys found on the public Internet were used on more than one host – many of them network devices such as routers, firewalls, and server management cards.

2.3 Promising Approaches

There does not seem to be a way to fully solve SSH host key management without adding new code in several widely used SSH implementations. OpenSSH certificates, use of X.509 certificates, something similar to how Let's encrypt handles X.509 certificates for SSL, and the ProxyCommand approach in RedHat are all potentially promising avenues. However, one approach needs to be selected, standardized, server side implemented, and support added to several common clients. Interoperability between the new system and existing implementations is going to be critically important during the transition period.

3 User authentication keys

User keys are a mechanism for authenticating a user to an SSH server. It is an alternative to password-based authentication, and is commonly used by automated scripts, automated file transfers, single sign-on by system administrators and software developers, and source code version control systems.

In public key authentication, the client first establishes an encrypted connection with the server (authenticating the server using host keys) and then sends a digital signature of the user's login name and various session-specific information to the server. The user's private user key is used for generating the signature. The public user key is configured on the server, typically in an *authorized keys file* in the user's home directory on the server.

Originally, public key authentication was envisioned to replace the insecure `.rhosts` authentication, which relied on easily spoofable IP addresses, and to provide a practical, secure single sign-on solution on Unix/Linux. In 1995, environments were small and this was quite manageable. Today, some organizations have hundreds of sysadmins, millions of daily logins using public key authentication, and millions of

user keys listed in tens of thousands of authorized keys files in their environments.

Public key authentication allows users to configure single sign-on for themselves, even across organizational boundaries, or to grant access for colleagues. This is handy in universities and for collaboration, but undesirable in high-security environments. It also makes it easy for sysadmins to configure automatic scripts that transfer data between systems, even between organizations, but lacks the approval processes and auditability required in regulated enterprises.

In analyzing SSH keys for dozens of large enterprises, it has turned out that in many environments 90% of all authorized keys are no longer used. They represent access that was provisioned, but never terminated when the person left or the need for access ceased to exist. Some of the authorized keys are 10-20 years old, and in our experience, scans of large enterprises often find about 10% of them granting root access or other privileged access. The vast majority of private user keys found in most enviroments do not have passphrases.

SSH keys are typically installed by system administrators and power users. Typical reasons for installing them are convenience (e.g., direct single sign-on to all Oracle servers by a database administrator), integrating applications, and deploying automated systems management tools, such as patch management, monitoring, auditing, and file transfers. Some sysadmins also install them to go around cumbersome privileged access management systems.

Balduzzi et al [6] analyzed the use of SSH user keys in newly created EC2 computing instances in the Amazon AWS cloud service, and found about a thousand distinct AMI images (operating system images from which virtual machines can be installed) with authorized keys preinstalled and over two hundred images with also private keys recoverable from the image. They were also able to match running instances to images with SSH keys with fairly good success rates to facilitate effective attacks. This shows that even freshly installed machines are not immune to SSH key based attacks unless care is taken in the management of keys and proper audits are in place.

Foster et al [8] examined a popular aftermarket telematics control unit (TCU) for cars, and found both an authorized key for the root user and the corresponding private key on the device. They then found about 3000 of these devices in an Internet scan, while many times more likely existed behind NATs, as the device is commonly used for car insurance purposes. The access the keys provided was sufficient to remotely control safety-critical automobile features, including breaks. They also found that the same key granted root access for various other models of devices from the same manufacturer.

Kent and Shrestha [14] is one of the earliest comprehensive descriptions of the problems caused by SSH user key proliferation. It is still relevant and a good description of how SSH keys work.

3.1 Current best practice for user keys

The best current practice for managing SSH user keys is described in in NIST IR 7966 [30]. It involves discovering existing keys by scanning installed systems, monitoring which keys are actually used, removing unused and policy-violating keys, assigning ownership to remaining keys, and implementing a proper provisioning and termination process for key-based access.

Dozens of major enterprises are in various stages of implementing the NIST process. Products assisting in the implementation include, e.g., Universal SSH Key Manager (SSH.COM, MA) and Encryption Director (Venafi, Inc., UT). The NIST process is good at dealing with keys in existing legacy environments, but is not well suited for dynamic cloud environments. It also does not address access between organizations, access to version control repositories, or integration into IAM systems.

3.2 Other attempts to address user keys

Tectia SSH supports X.509v3 certificates for user keys. These are widely used with smartcards, including the PIV and CAC cards used by the US government. However, other than use with smartcards, we have not seen much use of X.509v3 certificates for user keys. Their main advantage is that they expire, but they need proper management and rotation to prevent outages due to expired keys.

OpenSSH supports its own proprietary certificate format also for user keys and grants access to users using certificates. However, we don't know of any hardened CA (Certificate Authority) solutions for these proprietary certificates, and thus managing their life cycle in any security-conscious organization is a problem. The use of OpenSSH certificates for user keys makes it difficult or impossible to audit by examining a server which keys have access to the server. Instead, one has to rely on auditing the CA, and when the CA is not hardened and designed for audit, a trustworthy audit of issued certificates is not possible. Revoking OpenSSH certificates is also very cumbersome and must be done separately on every server. We have seen that even hardened public CAs for SSL certificates have been compromised or have otherwise issued tens of thousands of unauthorized certificates recently [7, 17, 22], so having a proper auditable CA is very important. Something similar to certificate transparency [16, 19] might also be help.

The OpenSSH LDAP Public Key Patch [5] enabled fetching authorized keys from an LDAP (Lightweight Directory Access Protocol) directory. Today OpenSSH offers the `AuthorizedKeysCommand` option, which can be used to specify a program to fetch authorized keys for a user from a directory or from another source. This is a very powerful feature that can be used to build centralized access provisioning solutions.

Kolano [15] describes a system where a library preloaded

into an OpenSSH server intercepts the system calls that access the authorized keys file. The library then calls an external command (similar to the modern `AuthorizedKeysCommand` option) that fetches authorized keys from a local Mesh Authentication Point (MAP). The MAP is essentially a key management server that returns authorized keys for the particular user on demand.

The STAR key management system [4] is perhaps the earliest published general-purpose SSH user key management system. It manages access to shared accounts, and allows users to upload their public key to the system to provision access to shared systems. It implements an approval workflow where users request access to an account, and after the administrator approves the request, the access is automatically provisioned. It supports identifying the user logging in to a shared account based on the user's key fingerprint. Administrators can revoke access through the central system if a user's key is compromised.

RedHat Linux IPA (Identity, Policy, Audit) supports storing SSH user keys in an LDAP directory. It also supports SSHFP for storing key fingerprints in DNS. It uses the OpenSSH `AuthorizedKeysCommand` option to run a helper program that fetches the keys from the directory. This can work beautifully in a pure RedHat environment, but doesn't solve the problem for mixed environments, external access, or existing legacy keys.

Netflix BLESS [18] is an approach to controlling SSH access using ephemeral certificates that are issued on-demand based on information in an external directory. This appoach shows promise for new environments and is employed in, e.g., PrivX (SSH.COM, MA). The approach is particularly well-suited for dynamic cloud environments and several commercial vendors have adopted this approach for implementing cloud access control or light-weight privileged access management.

However, the BLESS approach is not a solution for legacy environments, and transitioning legacy SSH keys into this model is a major undertaking. It also doesn't easily lend itself to automated access use cases. For legacy environments, something like the NIST process is still likely to be needed to bring them under control.

Various earlier uses of short-lived certificates in other authentication applications are presented in Sharma [25] and its references. Sheridan [26] has also spoken about SSH certificates.

Napier [20] discusses implementing least privilege using SSH keys, sudo, and setuid, and gives recommendations for using SSH keys and command restrictions. It also discusses use of source restrictions, which can limit abuse of leaked keys, and emphasizes the dangers of null-password root SSH keys. The paper further discusses the dangers of storing authorized keys in NFS home directories.

Thorpe [27] describes a system that implements a secure `su` for executing privileged operations using SSH keys. The idea is to generate a separate user key pair for each privileged command, and set a command restriction forcing that command for each such key. A master host holds a key granting root access to all other systems, and can distribute authorized keys to all other machines. Storing keys on NFS volumes, risks of storing keys on NFS volumes, and storing them on the server's local disk are also discussed.

Harchol, Abraham and Pinkas [10] presents an interesting approach using *k*-out-of-*n* threshold RSA signatures and dividing private keys among multiple devices, so that no device ever holds the entire key, the compromise of a any single device does not compromise the private key, and no single node failure prevents the system from operating.

We don't know of any practical solutions for managing SSH user keys used for accessing external organizations or github and related cloud applications. Management of router and IoT (Internet of Things) keys is also very limited in current solutions.

3.3 Promising Approaches

It seems that centralizing information about user-to-user access grants is desirable, for both automated and interactive access. The ultimate solution might be a new authentication method in SSH, or it might be built on top of existing public key authentication using user keys, with the AuthorizedKeysCommand used for fetching authorizations on the server side, and some new mechanism (e.g., a new authentication agent, possibly with extensions to the agent protocol) used for fetching the proper keys on the client side. This could be combined with caching information on the server side to reduce delays, allow operation even when the centralized server is temporarily down, and minimize man-in-the-middle attack possibilities.

The BLESS approach also looks very promising. It can be easily integrated with Active Directory or LDAP and be used to provision access on demand. It works particularly well for interactive access by users in environments where access can be forced to go through a jump server. Alternatively, an agent integrated to the SSH client or SSH agent on the client side could be used to obtain a certificate for each connection.

In our opinion, a good solution might have the following elements:

- Centralize information about user-to-user access grants, for both interactive and automated access. For interactive access, this is easy, but for automated machine-to-machine access the repositories for this information don't readily exist.
- Use short-lived certificates for actually granting access on-demand for interactive users, or introduce a new authentication method for this.
- Implement a solution for machine-to-machine access, for example using a client-side agent or modified client

that fetches a short-lived certificate for automated connections. Here a challenge is how to make the transition as easy and robust as possible.

4 Idiosyncrasies of current SSH implementations

Besides protocol issues, there are important limitations in current SSH key based authentication implementations that need to be taken into account when designing key management solutions.

The OpenSSH server counts checking whether the server would accept an available key as a login attempt even if no signature is sent and thus no real login attempt is made. This was not the intention of the original protocol design nor suggested anywhere in the protocol standard. The original idea was that the client can offer a key to the server without generating a signature, so that it does not need to ask for a passphrase or for a PIN for a smartcard if the server is not going to accept the key anyway. This is important for a smooth user experience. Such checks were not intended to be counted as login attempts. OpenSSH's behavior can break existing connections when an additional private key is added to a user account. Keys made available through the authentication agent, by inserting a smartcard, or generating a new key can break existing connections. This behavior can be counter-intuitive and needs to be taken into account when designing SSH key management solutions and key rotation implementations that must work with existing OpenSSH servers.

Current SSH implementations do not generally allow specifying more than one command restriction for the same key. Even if a key appears multiple times in the known hosts file, some versions only use the first instance. This may require use of different keys for different commands, which may cause further complications for key rotation with OpenSSH's restriction on the number of keys offered.

The granularity of permissions that can be specified for authorized keys is rather limited in many implementations, especially for file transfers. For example, it would be desirable to configure that a key can only be used for reading files from a particular directory using SFTP. Currently it is very easy to misconfigure systems so that someone granted SFTP access can also write files, possibly outside the intended directory. Allowing better configuration of permissions associated with keys would be desirable.

Many companies would like to log which client-side user is logging in using an SSH key. This information is not currently sent in the SSH protocol (nor could the server necessarily trust the information sent by a client). Only the key fingerprint can be logged. Existing SSH implementations do not uniquely associate private keys with users, and thus mapping keys used for login to client-side users is challenging. Having a reliable, centralized mapping from keys to users would help monitoring and intrusion detection.

Each SSH implementation generally uses its own fingerprint format for logging which SSH keys were used for logins. Even the same implementation can support multiple formats, and the formats have historically changed over time in different versions (e.g., in OpenSSH). It is usually not possible to convert from one fingerprint format to another because they use different hash functions. Therefore, it is difficult to associate log records with keys and thus users. Key management systems may need to save key fingerprints in all formats to enable associating log records reliably with keys and users.

5 SSH in large enterprises

Large enterprise IT environments are substantially different from what most independent software developers and researchers think or have experience about. Many industries are highly regulated, at least for their payment and financial data systems (e.g., by PCI-DSS and Sarbanes-Oxley). Finance, healthcare, energy, and government systems have their own sets of regulations (e.g., COBIT, NYDFS, HIPAA, NERC CIP, FISMA/NIST SP 800-53, CSC 20). Privacy regulations (e.g., GDPR) add their own set of requirements and penalties.

Enterprise information systems tend to be highly regulated, especially for core production systems. Privileged access is tightly controlled. Key elements of most regulations include controlling who has access to what systems and data, only granting access on a need basis, terminating access when it is no longer needed, the principle of least privilege, changing access credentials and security keys on a regular basis, and regular audit and improvement.

Large enterprises have thousands of servers. Information-intensive enterprises typically have tens of thousands, sometimes hundreds of thousands of servers. Additionally, they could have hundreds of thousands of workstations. This implies that hundreds to thousands of systems are replaced every day.

Most users in large enterprises are restricted to using a limited set of applications, and their credentials only grant them access to those applications. Operating system level access is generally reserved to IT specialists – such as system administrators, database administrators, and software engineers. However, some organizations have thousands of such IT specialists, many of them contractors.

Large enterprises usually employ some identity and access management system (IAM) for provisioning and terminating access for their users. Active Directory is widely used. However, these systems almost never cover machine-to-machine access using SSH keys. Service accounts running databases and application processes are often managed locally or using various legacy directories, including LDAP and NIS (Network Information Service), and are usually not part of their normal identity and access management. It is not uncommon for large organizations to have multiple Active Directory forests,

sometimes complemented by multiple NIS and/or LDAP directories and other legacy systems.

Many organizations also use a networked file system on some of their systems. NFS is common in Unix/Linux environments, and often the same home directories are shared by thousands of computers, all of which can be accessed using any authorized key in any user's home directory. NFSv3, which is still very widely used, doesn't protect the file system cryptographically and may allow network-level attackers to write any files – including authorized keys files.

SSH keys are typically provisioned by system administrators and IT specialists, often without any central control, auditing, or termination process. The keys typically continue to grant access to systems even after the person has left the organization. Many SSH keys also facilitate automatic processes, data transfers, and automate systems management. Such keys are often business critical, potentially causing major outages if such automation fails to operate. Many organizations have thousands or tens of thousands of scripts using SSH keys to facilitate the automation, and SSH keys are built into many enterprise file transfer tools and systems management tools. It is also common for software to do "agentless management", which usually means they use SSH keys to login to the system.

We have found that most enterprises are shocked to find how many keys they have in their authorized keys files. The number is sometimes orders of magnitude more than they expect. Several times management has told us they don't use SSH keys, only to later find thousands of authorized keys on their systems.

SSH keys tend to accumulate not only from scripts and management tools, but also from system administrators and database administrators implementing their private single sign-on from their personal desktop or laptop to various systems and database accounts that they administer – often circumventing cumbersome privileged access management systems.

Enterprise IT systems tend to be business critical. Changes are rarely made without proper cause. Software is often not upgraded for years, and we commonly see operating system versions that are more than ten years old, even in major financial institutions. It is not uncommon to see them running applications that aren't available for newer versions of operating systems. Most organizations use a wide mix of operating systems, including Linux, Windows, Unix, z/OS, and/or others. It is very difficult for them to replace SSH client and server software, and doing so would be a multi-year project that in itself could cost them hundreds of thousands of dollars in labor.

Based on our evaluation, enterprise IT systems are currently quite vulnerable to an attack spreading to other systems using SSH keys once a server or system administrator laptop/desktop has been compromised. It is common for attackers and malware to collect SSH keys. Unterminated SSH access also makes organizations vulnerable to penetration using keys held by employees and consultants who have left the organization. The keys could leak even after the person has left.

Enterprise IT systems are also fragmented. Most large enterprises are results of multiple mergers, and IT systems have been only partially integrated. There are often multiple independently administered subdivisions or compartments that are ostensibly isolated from the rest of the network. It is common, though, to see SSH keys granting access across compartment boundaries, such as from development into production, or from other systems into financial data or payment environments.

Routers and server hardware (including IPMI ports, which also often support SSH key based login) are often managed by different teams than logical servers. However, they are a major and business-critical use case for SSH. Server management ports are a particularly effective way to gain and hide covert access by attackers.

Hosts on an enterprise network may or may not have connectivity to the external Internet. It is rather common in large organizations that they do not.

Most enterprises currently store their existing SSH keys on servers and clients in varying locations. Sometimes they are in the default location and sometimes a non-standard location is specified in an SSH configuration file. Some organizations also have custom-built SSH servers with non-standard paths for configuration files or authorized keys files. It is somewhat common to move authorized keys to root-owned locations, such as `/var` or `/etc`. This prevents ordinary users from granting permanent access to others at will.

6 Shortcomings of existing approaches

While there are several open source packages for generating and distributing OpenSSH keys, the proprietary OpenSSH certificates are not supported by most clients and are are not standardized. Use of X.509v3 certificates with SSH is an IETF standard [12], but it is not currently supported by OpenSSH. Use of OpenSSH certificates for user authentication further suffers from the difficulty of auditing from a server who has access to it and lack of hardened certificate authorities. There is also no centralized revocation functionality. Overall, multi-platform tools and standards for distributing certificates to servers and clients and updating them are missing.

No automated deployment mechanism for deploying a host into SSH key management during installation is known to exist. No existing system for determining whether to just use TOFU or to require stronger authentication for a particular host is known.

For user authentication, BLESS and PrivX provide good solutions for many use cases. However, they can't yet replace SSH keys for automated machine-to-machine access, nor can they transparently replace public key authentication in existing scripts and applications. Combining them with the NIST

process helps, but they still don't fully address all of the use cases.

The ability to audit SSH key access and monitor which authorizations are in use is still largely an open problem for hybrid environments and for external connections.

Implementing transparent compatibility while giving incremental benefits during the transition period from current user authentication keys to whatever will replace them long term is still an open question that nobody seems to have addressed. Given that the transition and hybrid phase is likely to last years, easily over a decade, good transition solutions are needed.

Many proposed solutions rely on specific, non-standardized features of a single implementation (OpenSSH). Enterprises use SSH on all sorts of platforms - Linux, Unix, Windows, Mac, z/OS, AS/400, routers, printers, modems, server management ports, storage boxes, IoT devices, libraries in applications, and more. There are dozens of implementations in wide use. Getting a solution universally deployed will require automation and eventual standardization.

7 Requirements for future host key management

Protection from man-in-the-middle attacks is important. It is particularly important for preventing spread of attacks within an organization via password theft or command injection and for preventing compromise of connections over the Internet by state actors.

Making host authentication secure and practical implies the following requirements.

- Host key checking should be automatic and enforced by default.
  - *Rationale:* Given that users don't check fingerprints [9] but the host key check is critical for man-in-the-middle attack protection, it must be somehow automated and automatically enforced. Users should not be relied on to accept keys.
  - It has also become common for attackers to establish presence in routers, printers, and other devices on the local network, making man-in-the-attacks more common. Protecting even the first connection is increasingly important.
- Initial deployment of a host into the system should be easy.
  - *Rationale:* Ease of deployment is necessary for wide adoption for a tool that is installed on almost every system. Ideally, enrollment into the system should be automatic after simple organization-level configuration.
  - Laptops may move between organizations or organizational units, and should automatically verify host keys using the local infrastructure when possible.
- Server upgrades and host key rotation should be automatically supported (once deployed).
  - *Rationale:* Rotation support regularly changing keys. It is particularly important if there has been a compromise and may be required for compliance.
- The approach must work also for clients connecting from behind NAT (Network Address Translation) and inside organizations where computers generally do not have direct Internet access.
  - *Rationale:* Most organizations implement NAT at their firewall/external connection. Some translate between IPv4 and IPv6. Some organizations perform NAT internally. Many cloud servers are behind NAT.
  - Many large enterprises do not offer Internet access to computers on their internal networks unless special arrangements are made (approvals, proxies, etc).

It is not a requirement that public keys be used in the same way as today; however, maintaining compatibility during a transition period is probably easier if keys are used.

7.1 Requirements for future user key management

7.1.1 User key use cases

SSH keys are used to solve various access needs:

- Access by scripts, such as backup scripts, audit scripts, scripts for integrating applications, and log data collection tools
- Agentless management by various enterprise applications, such as systems management and file transfers
- Access to cloud instances
- Access to version control repositories, such as git and github
- Authorizing external file transfers
- Access from jump servers to end hosts, and
- Grass-roots single sign-on by system administrators, researchers, students, etc. – often crossing organizational boundaries.

All of these use cases are valid and the need for them is not likely to go away any time soon, even though many organizations only use or only want to allow some of them.

7.1.2 User key management requirements

Each use case should be considered in view of requirements for SSH access management, including the following:

- Ensure that access is terminated when a user leaves the organization or when automated access between systems is no longer needed.
  - *Rationale:* Currently about 90% of all authorized keys are no longer used in the large organizations view have scanned, and represent unterminated access. This is a major security vulnerability, failure of access management process, and a violation of most security regulations.
- Implement and enforce proper approvals and accountability for machine-to-machine authentication, especially for access to privileged accounts.
  - *Rationale:* Currently system administrators in many organizations install authorized keys at will, whenever needed for the task at hand, and no reliable record of keys exists. This leads to a lack of approvals and termination, non-enforcement of policy, and impossibility of auditing proper termination of machine-to-machine keys.
- Access to servers should be fully and easily auditable.
  - *Rationale:* Security starts from knowing and controlling who has access to a system or data and at which level. It is important to be able to ensure that there are no backdoors and that processes are functioning throughout the access life cycle.
  - Audit should include enforcing proper approvals, proper termination, assignment of ownership to access grants, and enforcement of internal security boundaries.
  - Auditability is also necessary to fully comply with most security regulations and industry best practice.
- It should be possible to delegate who can grant access to which accounts.
  - *Rationale:* In large enterprises, usually only application teams know who should transfer data with each application. Central security teams don't know each application intimately. Large enterprises run thousands of applications.
  - For universities and smaller organizations, flexibility and ease of use are very important and many systems are self-administered.
  - Granting access to privileged accounts or across security boundaries may require additional reviews or approvals.
- Rotating user credentials should be supported.
  - *Rationale:* If user keys have been compromised, changing them is imperative. Most large enterprises today do not seem to have that capability.
  - Periodically changing user keys limits risk of leaked keys and ensures any unaccounted-for authorized keys eventially become unusable.
- Use of access credentials should be monitored and it should be possible to centrally identify and revoke access rights that are not used or that violate policy.
  - *Rationale:* Application teams and individuals often fail to properly terminate access when it is no longer needed. This requirement establishes a second line of defense, beyond the normal life cycle process, to ensure that unused access eventually gets terminated.
  - This is also needed for auditing that the normal access life cycle process is functioning properly and unused access is properly terminated.

7.2 General requirements

The following requirements apply to an SSH key management system in general, for both host keys and user keys.

- Mixing clients and servers that support the new mechanism and clients and servers that do not support it should be supported. While the new system is gradually being deployed, new clients used with new servers should provide full security benefits while allowing interoperability with legacy clients and servers. The determination of whether a host supports the new system should be as reliable as possible.
  - *Rationale:* There is no way to upgrade large enterprises in a single go. The transition is likely to last many years, possibly over a decade.
  - Interoperability is absolutely critical. There are dozens of SSH client and server implementations in wide use on different platforms. It may be a long time before they are all upgraded.
  - It is not realistic to assume enterprises immediately update all the thousands of scripts, third-party applications, and legacy servers that use SSH keys.
- The approach should work across organizational boundaries for collaboration, file transfers, cloud services, external version control repositories, and remote working.
  - *Rationale:* Many organizations will have a hybrid on-premise and cloud setup for the next decade or more.
  - Remote working, distributed organizations, and use of off-site contractors are common in IT.
  - Countless software developers use cloud-based version control systems.
- The system should not create a single point of failure. It should be possible to continue to connect securely,

at least for a time and to servers recently used by the connecting client, even if key management servers or parts of the network are down.

  - *Rationale:* Systems sometimes go down but the enterprise should not. This might be achieved, e.g., via replication or caching.

- The system should not introduce denial-of-service opportunities.
  - *Rationale:* If the SSH server communicates with external servers before authentication is complete, that introduces additional delay to the login process and may make it easy to implement resource exhaustion attacks agains the server.

- The system should not add excessive latency (several seconds is acceptable for a first connection, but regular connections should be reasonably fast).
  - *Rationale:* We don't want to degrade user experience, introduce opportunities for denial-of-service, or unduely load servers.
  - We have seen servers in enterprises that get many key-based logins per second.

8 Conclusion

Given the prevalence of the SSH protocol in the Internet and cloud computing infrastructure and the alarming state of managing SSH keys in many enterprises, it is obvious that implementing processes and solutions is a priority.

SSH host key management and user key management are complicated and solutions must balance different requirements. It does not look like the ideal solution has been invented yet. Help is thus needed from the research community to address these issues and to introduce solutions to secure the Internet infrastructure and, indeed, the underlying infrastructure on which economies and our everyday life and work increasingly depend on.

This paper aims to help the research community better understand what the problems are, what large IT environments really are like, and what must be taken into account in designing a solution.

Improving SSH's access management and man-in-the-middle attack protection is a high priority for improving the resilience of our information infrastructure and society.

References

[1] Yasir Ali and Sean Smith: Flexible and Scalable Public Key Security for SSH. In Proceedings of the European Public Key Workshop (EuroPKI 2004), Lecture Notes in Computer Science, vol. 3093, pp. 43-56, Springer, 2004.

[2] Mansoor Alicherry and Angelos D. Keromytis: DoubleCheck: Multi-path verification against man-in-the-middle attacks. IEEE Symposium on Computers and Communications, IEEE, 2009.

[3] Roy Arends et al: DNS Security Introduction and Requirements, RFC 4033, IETF, 2005.

[4] D. Arkhipkin, W. Betts, J. Lauret and A. Shiryaev: An SSH key management system: easing the pain of managing key/user/account associations, International Conference on Computing in High Energy and Nuclear Physics (CEPH'07), Journal of Physics: Conference Series, vol. 119, IOP Publishing, 2008.

[5] Eric Auge: OpenSSH LDAP Public Key Patch, presented at the 6th Free and Open Source Software Developers' European Meeting (FOSDEM), 2006.

[6] Marco Balduzzi, Jonas Zaddach, Davide Balzarotti, Engin Kirda and Sergio Loureiro: A security analysis of Amazon's elastic compute cloud service. Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC'12), pp. 1427-1434, ACM, 2012.

[7] Peter Bright: Another fraudulent certificate raises the same old questions about certificate authorities, Ars Technica, August 30, 2011.

[8] Ian Foster, Andrew Prudhomme, Karl Koscher and Stefan Savage: Fast and Vulnerable: A Story of Telematic Failures. In Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT'15), USENIX, 2015.

[9] Peter Gutmann: Do Users Verify SSH Keys? ;login:, 36(4):35-36, USENIX, August 2011.

[10] Yotam Harchol, Ittai Abraham and Benny Pinkas: Distributed SSH Key Management with Proactive RSA Threshold Signatures. In Proceedings of the International Conference on Applied Cryptography and Network Security (ANCS 2018), pp. 22-43, 2018.

[11] Nadia Heninger, Zakir Durumeric, Eric Wustrow and J. Alex Halderman: Mining your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In proceedings of the 21st USENIX Security Symposium, USENIX, 2012.

[12] Kevin M. Igoe and Douglas Stebila: X.509v3 Certificates for Secure Shell Authentication. RFC 6187 (Standards Track), IETF, 2011.

[13] John P. Jones, Daniel F. Berger and Chinya V. Ravishankar: Layering public key distribution over secure DNS using authenticated delegation. In 21st Annual Computer Security Applications Conference (ACSAC'05), IEEE, 2005.</p>

[14] Greg Kent and Bhavna Shrestha: Unsecured SSH – the Challenge of Managing SSH Keys and Associations. SecureIT whitepaper, 2010, 2012; from https://www.secureit.com/resources/SSH_Key_Associations_Article%20Final.pdf, accessed 2019-01-30.

[15] Paul Z. Kolano: Mesh: secure, lightweight grid middleware using existing SSH infrastructure. In Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT'07), pp. 111-120, ACM, 2007.

[16] Ben Laurie, Adam Langley and Emilia Kasper: Certificate Transparency. RFC 6962, IETF, 2013.

[17] Neal Leavitt: Internet Security under Attack: The Undermining of Digital Certificates. Computer, 44(12):17-20, IEEE, 2011.

[18] Russell Lewis: How Neflix Gives All Its Engineers SSH Access to Instances Running in Production. Presentation at Oscon 2016. From https://www.youtube.com/watch?v=JwLGsWYVjqU, accessed 2019-01-30.

[19] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten and Michael J. Freedman: CONICKS: Bringing Key Transparency to End Users. In Proceedings of the 24th USENIX Security Symposium, pp. 383-398, USENIX, 2015.

[20] Robert A. Napier: Secure Automation: Achieving Least Privilege with SSH, Sudo and Setuid. In Proceedings of the 18th Large Installation System Administration Conference (LISA'04), pp. 203-212, USENIX, 2004.

[21] Jin Peng and Xin Zhao: SSH-Based Device Identity and Trust Initialization. Information Security Journal: A Global Perspective, 19(5):237-242, 2010.

[22] Fahmida Y. Rashid: Google to Symantec: We don't trust you anymore. Infoworld Tech Watch, Mar 24, 2017.

[23] Stuart E. Schechter, Jaeyeon Jung, Will Stockwell and Cynthia McLain: Inoculating SSH Against Address Harvesting. In Network and Distributed System Security Symposium (NDSS 2006), Internet Society, 2006.

[24] Jakob Schlyter and Wesley Griffin: Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. RFC 4255, IETF, 2006.

[25] Pranav Kumar Sharma: Short-Lived Certificates as Mobile Authentication Method. Master's Thesis, Helsinki University of Technology, 2009.

[26] Niall Sheridan: Managing SSH Access without Managing SSH Keys. Presentation at USENIX Large Installation System Administration Conference (LISA'17). Slides at https://www.usenix.org/sites/default/files/conference/protected-files/srecon17emea_slides_niall_sheridan.pdf, USENIX, 2017.

[27] Christopher Thorpe: SSU – Extending SSH for Secure Root Administration. In Proceedings of the 12th Systems Administration Conference (LISA'98), pp. 27-36, USENIX, 1998.

[28] Tatu Ylonen: SSH – Secure Login Connections over the Internet. In Proceedings of the 6th USENIX Security Symposium, pp. 37-42, USENIX, 1996.

[29] Tatu Ylonen and Chris Longvick (ed.): The Secure Shell (SSH) Protocol Architecture, RFC 4251, IETF, 2006.

[30] Tatu Ylonen, Paul Turner, Karen Scarfone and Murugiah Souppaya: Security of Interactive and Automated Access Management Using Secure Shell (SSH). NISTIR 7966, National Institute of Standards and Technology, 2015.

[31] Dan Wendlandt, David G. Andersen and Adrian Perrig: Perspectives: Improving SSH-Style Host Authentication with Multi-Path Probing. In Proceedings of the 2008 USENIX Annual Technical Conference, 2008.