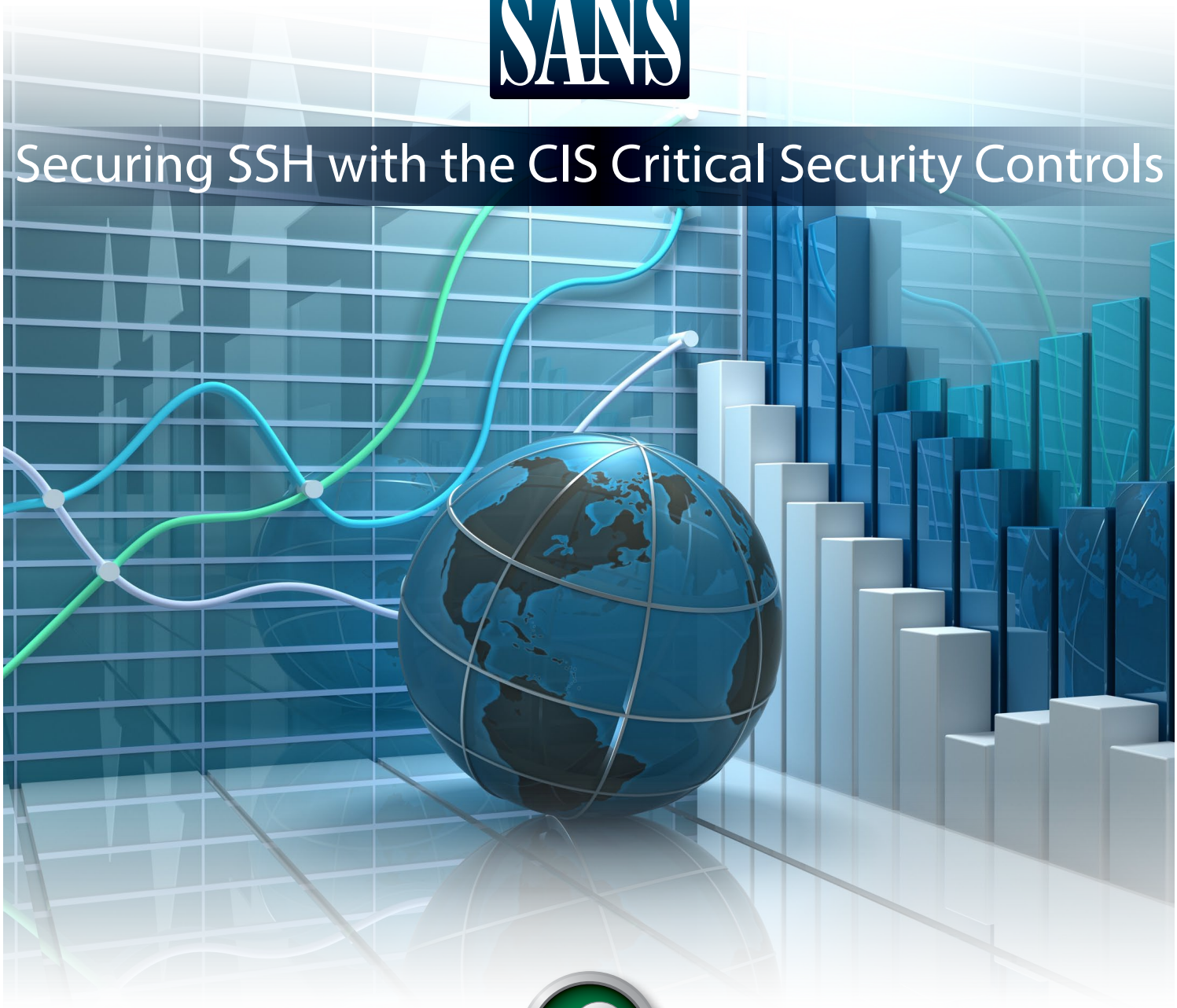
Interested in learning more about security?

SANS Institute InfoSec Reading Room

Securing SSH with the CIS Critical Security Controls

A SANS Analyst Program whitepaper by Barb Filkins. It discusses how the Critical Security Controls coupled with good configuration management processes can support the effort required to avoid the risks inherent to SSH.

Securing SSH with the CIS Critical Security Controls



A SANS Whitepaper

Written by Barbara Filkins

December 2015

Sponsored by
Venafi

Introduction

Secure Shell (SSH) is a de facto Internet standard that provides an encrypted, authenticated alternative to other networking protocols, enabling its users to enter keyboard commands, transfer information or access a remote desktop securely. SSH's support for public key authentication has resulted in its wide adoption for automated, privileged access to mission-critical systems and sensitive data.

But what happens when organizations leave SSH itself unsecured? What inherent vulnerabilities exist in the protocol and its implementations? Today, large-scale SSH deployments involve hundreds of thousands of keys and millions of associations spread across thousands of systems, potentially inviting exposure of sensitive information or an attack against critical infrastructure. Organizations often fail to follow best practices related to SSH configuration or establish adequate controls over SSH keys and key associations. According to a 2014 Ponemon Institute study on SSH vulnerabilities, most organizations (74 percent) don't enforce SSH key policies, and 47 percent of the IT security professionals surveyed said they had to deal with incidents due to compromised or misused SSH keys in the prior 24 months.[1] So the question is, how can an enterprise defend against risks resulting from poor implementation and management of an otherwise secure protocol?

Most SSH vulnerabilities can be avoided through proper configuration and implementation. Remediating issues with existing SSH key and access management deployments can be time-consuming and labor-intensive, even with automated tools. It can be worse for a Fortune 500 company that may have more than a million keys—small files of only a few kilobytes—granting access and elevated privileges to sensitive information or systems. Gaining the necessary knowledge of an existing environment can be especially daunting. To meet this challenge, in this paper we explore how the Center for Internet Security (CIS) Critical Security Controls—coupled with good configuration management processes—can support the investment that organizations, especially larger ones, must undertake to avoid risk associated with their SSH deployments.

[1] "Ponemon 2014 SSH Security Vulnerability Report," www.energycollection.us/Energy-Security/Ponemon-2014-SSH.pdf

How SSH Works

Critical/Common Uses of SSH

1. "Interactive" use (which can be automated as well for host-to-host access with no human intervention):
   - Login to a shell on a remote host (replacing Telnet and `rlogin`)
   - Execution of a single command on a remote host (replacing `rsh`)
   - Automatic (password-less) login to a remote server
   - Automated remote monitoring and management of servers
2. File transfers:
   - Secure file transfer (Secure File Transfer Protocol, or SFTP, as an encrypted alternative to FTP)
   - Combined with `rsync` to back up, copy and mirror files efficiently and securely
3. Point-to-point tunneling:
   - Forwarding or tunneling a port (not to be confused with a VPN, which routes packets between different networks or bridges two broadcast domains into one)
   - Use as a full-fledged encrypted VPN if using an OpenSSH server and client
   - Browsing the web through an encrypted proxy connection with SSH client software that supports use of a Socket Secure (SOCKS) proxy

SSH uses a client-server model of computing, allowing users to establish secure communications between two hosts, whether UNIX, Windows or virtually anything else. SSH is also increasingly important as the use of cloud computing grows, helping organizations limit exposure while connecting to a cloud-based virtual machine over the Internet by providing a local gateway-style system via an SSH endpoint.

Figure 1 provides a high-level view of the SSH architecture.

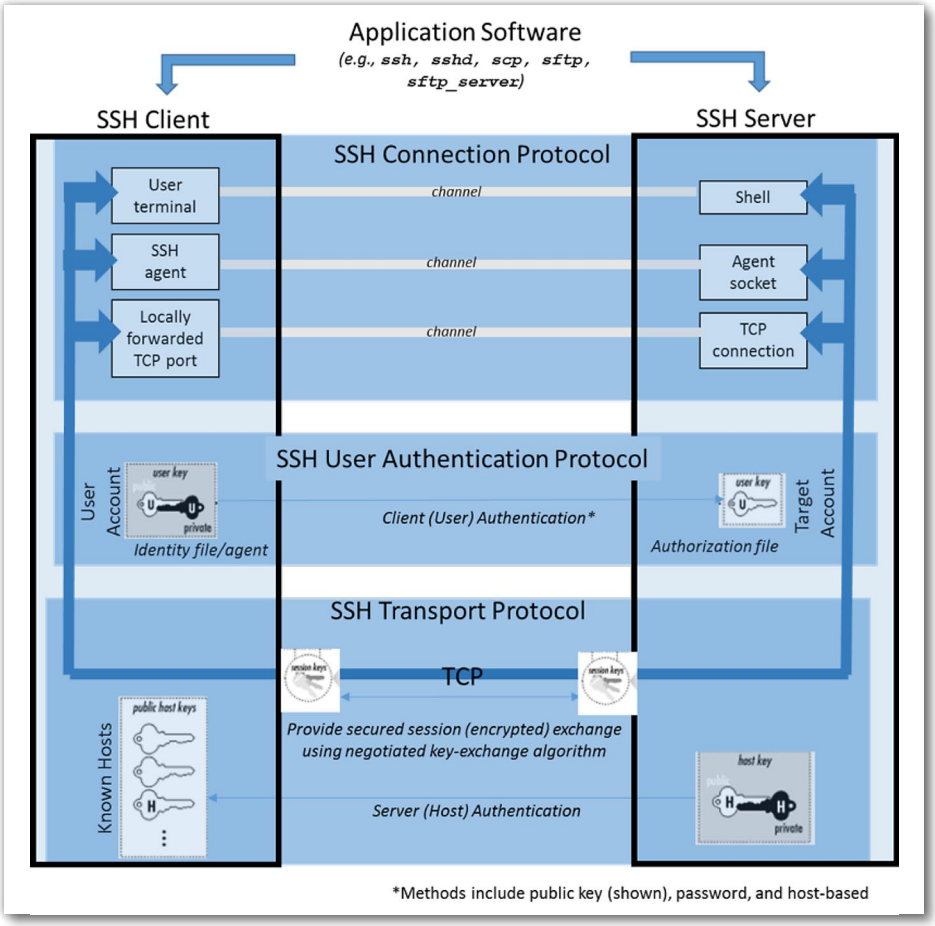


Figure 1. SSH System Architecture[2]

[2] SSH architecture is flexible, and it is defined in RFC 4251 (defines internal layers for SSH v2), www.rfc-editor.org/info/rfc4251
transport layer (RFC 4253), www.rfc-editor.org/info/rfc4253
user authentication layer (RFC 4252), www.rfc-editor.org/info/rfc4252
and connection layer (RFC 4254), www.rfc-editor.org/info/rfc4254

SSH comes in two flavors: SSH v1 and SSH v2, the latter being a total rewrite of the former. The two are incompatible in their implementation. SSH v1 should be deprecated in favor of SSH v2, as the second version is considered a stronger, more comprehensive product with strong cryptographic integrity check, full negotiation of modular cryptographic and compression algorithms, bulk encryption, message authentication code (MAC), public keys and support for public key certificates.[3]

The server enables incoming SSH connections to a host and handles user authentication, authorization and related tasks. The client connects to an SSH server and makes requests after authenticating to the server. An SSH session—the ongoing connection between a client and a server—begins after the client successfully authenticates to a server and ends when the connection terminates. Once the system authenticates a user, one or more channels may be opened within the connection, with each channel acting as an individual data link and separate pathway for information.

The balance of this paper assumes the use of SSH v2.

[3] For a full comparison of the differences, see www.snailbook.com/faq/ssh-1-vs-2.auto.html

Public key is the only SSH client authentication method recommended for automated access and interactive logins to accounts with elevated privileges. Initial server authentication is done in one of two ways: either with public/private keys that the SSH client must accept as trusted or with a certificate issued by a recognized certificate authority. Successful authentication results in an encrypted channel between the two hosts. One host acts as an SSH client, and the other acts as the SSH server.[4]

Figure 2 illustrates how this method uses keys or certificates to authenticate an interactive user (or automated process) operating on an SSH client (Host A) to an SSH server (Host B).

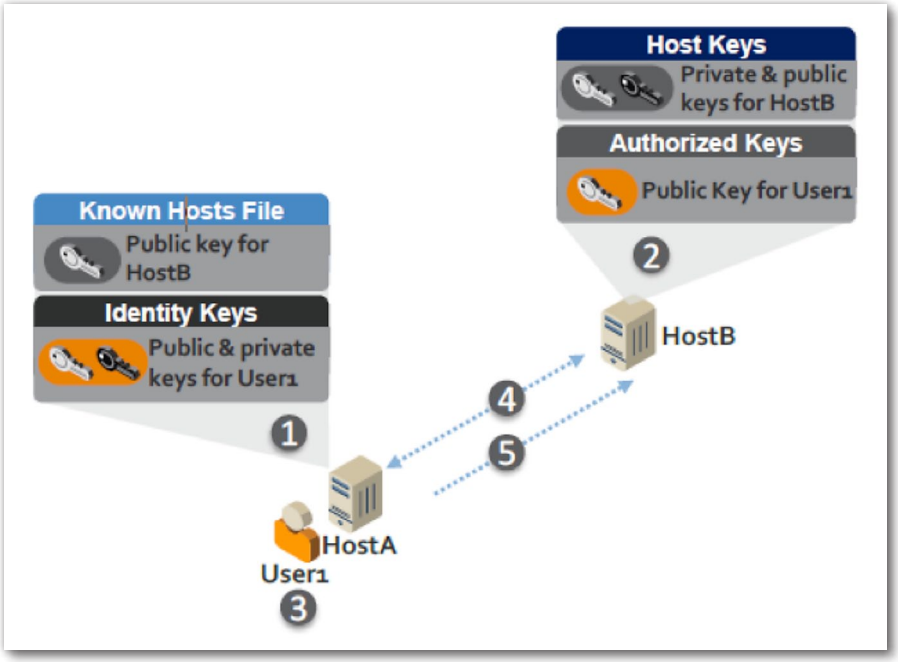


Figure 2. Public Key Client Authentication[5]

Trust Relationship

The access relationship an authorized key grants to an account on one system (SSH server) with the corresponding identity key in an account on another system (SSH client).

These keys authorize the interactive user or automated process to access a user account associated with the SSH host system. An interactive user or automated process on an SSH client (Host A in Figure 2) has a user key called an *identity key*, typically an RSA or DSA private key.

[4] Certificates that comply with the X.509 standard formats for public key certificates used in public key infrastructures are a useful alternative for host keys in large environments in which the number of host keys and the effort to manage their rotation among servers is more complicated than using X.509 certificates.

[5] NISTIR 7966, "Security of Interactive and Automated Access Management Using Secure Shell (SSH)," October 2015, Figure 3-5, http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf

The SSH server (Host B) must have the corresponding public key configured as an *authorized key* for the user account to which access will be granted. Any user or process in possession of the identity key can log into that user account on the server (Host B) and perform actions under the privileges configured for the account.

This method does not create an implicit trust relationship between two hosts, such as those created by Active Directory between parent and child domains. Instead, it creates an expressly defined trust relationship between two hosts that comes with clear limits on what functions a user or process from Host A can launch when authenticated to Host B.

Public key authentication also supports restrictions on commands (what can be done on a server using an authorized key) and restrictions on sources (limits the IP addresses from which logins using the authorized key can be used).

Together, these features support the ability to definitively audit who can access a system and an account, assuming that the authorized keys are controlled and tracked.

Vulnerabilities with SSH

Vulnerabilities with SSH range from weaknesses in the protocol itself to configuration, implementation and management complexities that can lead to dangerous mistakes. Poor SSH key management practices can result in compromise of administrative privileges, expanded breadth of attack surfaces and other issues that could easily be avoided by proper staff training and more effective processes backed by automation and tools. For example, inadequately trained key administrators might deploy authorized keys to root user accounts rather than to normal user accounts, giving the recipients far higher levels of access than originally intended. It's unlikely those administrators would notice other problems, either—for example, the organization's failure to upgrade from a flawed version of the SSH protocol to the most recent version. Although SSH v2 debuted in 2006, some organizations have yet to replace tools reliant on SSH v1, which is known for being vulnerable to man-in-the-middle attacks.

Configuration Issue: Development Awareness Around Critical Keys

Incidents involving GitHub, the popular Web-based hosting service for software development projects, reveal that many developers overlook the risks of including critical key files as part of a managed code base. On January 23, 2013, GitHub unveiled improvements to its code search engine, one of which turned out to be a little *too* efficient. On the following day, users noticed that a GitHub search easily found SSH private keys that programmers had mistakenly uploaded.

CIS Control 18.9. For in-house developed applications, ensure that development artifacts (sample data and scripts; unused libraries, components, debug code; or tools) are not included in the deployed software, or accessible in the production environment.[6]

This highlights again the general lack of understanding by developers of best practices for SSH key security, as well as the lack of oversight by organizations that delegate the responsibility for key management to developers who may not be trained in the subject's finer points. Some keys were reportedly associated with the production server for a major Chinese website, and others with Chromium, the source code repository for Google Chrome.[7] Also revealed were encryption keys in live configuration files from cloud services such as Amazon Web Services (AWS) and Azure.[8] (Ultimately, GitHub modified its indexing to ignore such files.)

[6] *CIS Critical Security Controls for Effective Cyber Defense Version 6.0*, Center for Internet Security, www.cisecurity.org/critical-controls.cfm

[7] "GitHub Search Makes Easy Discovery of Encryption Keys, Passwords In Source Code," SecurityWeek, January 24, 2013, www.securityweek.com/github-search-makes-easy-discovery-encryption-keys-passwords-source-code
"GitHub Forced to Disable Search After Exposing Private SSH Keys," Softpedia, January 25, 2013, http://news.softpedia.com/news/GitHub-Forced-to-Disable-Search-After-Exposing-Private-SSH-Keys-324200.shtml

[8] "GitHub Search Makes Easy Discovery of Encryption Keys, Passwords In Source Code," SecurityWeek, January 24, 2013.

Application developers must learn how to keep both configuration and private key files secure. With these files, a malicious individual can easily impersonate an authorized user (or host) and easily connect to a remote host and application to wreak havoc, as developer Ryan Hellyer found out.[9]

One evening in 2014, Ryan, a developer of plug-ins, uploaded his WordPress site to GitHub. Less than four hours after the upload, Amazon sent him an email, informing him to check his EC2 instances, which he found odd because he didn't use EC2. The problem was that he had no EC2 instances; his AWS access keys for GitHub were compromised. The net result was that around 600 EC2 instances were running within approximately 12 hours of upload to GitHub, and he was billed $6,000 for unauthorized usage.

There are several lessons here. First, Ryan thought he had taken all the necessary security precautions, removing the **`wp-config.php`** file containing his AWS access keys from his GitHub upload. However, he overlooked a backup file—**`wp-config.php.save`**—that contained his AWS access keys. Second, attackers are always actively scouring for and can quickly exploit stolen or leaked AWS (and other) keys. Finally, security must be part of application design, meaning find a way to store AWS credentials other than in **`wp-config.php`**.

Unfortunately, insecure practices continue. Subsequent analysis in the spring of 2015 revealed the inclusion of weak encryption in SSH keys in several high-profile source-code repositories could have resulted in compromise of music streaming service Spotify, Russian Internet company Yandex, the U.K. government and the Django web application framework.[10]

[9] "Ryan Hellyer's AWS Nightmare: Leaked Access Keys Result in a $6,000 Bill Overnight," WP Tavern, September 26, 2014, http://wptavern.com/ryan-hellyers-aws-nightmare-leaked-access-keys-result-in-a-6000-bill-overnight

[10] "Users with weak SSH keys had access to GitHub repositories for popular projects," ITworld, June 3, 2015, www.itworld.com/article/2931135/security/users-with-weak-ssh-keys-had-access-to-github-repositories-for-popular-projects.html; "Compromised SSH keys used to access popular GitHub repositories," Security Affairs, June 3, 2015, http://securityaffairs.co/wordpress/37459/cyber-crime/compromised-ssh-keys.html

Watch Default Configurations (CIS Control 5.3)

CIS Control 5.3. Before deploying any new devices in a networked environment, change all default passwords for applications, operating systems, routers, firewalls, wireless access points, and other systems to have values consistent with administration-level accounts.[11]

Few industry specifications define how device manufacturers should implement SSH or present guidance on how to best configure SSH in the field. This lack of guidance helped to cause a 2013 incident in which emergency alert system encoder/decoder devices from Monroe Electronics shipped with privileged keys hardcoded into the device. That unintentional elevation gave the devices sufficient levels of trust that attackers could have used one to log into the U.S. Emergency Broadcast System from anywhere on the Internet and broadcast fake messages across the top-priority network, established during the Cold War to let the U.S. government give citizens a 10-minute warning before a nuclear apocalypse.[12]

Cyber Kill Chain: Gaining Access with SSH

Lack of defined governance for SSH key-based trust relationships can allow an attacker who compromises one system to quickly pivot from one system to another and extend a breach into other parts of an organization. Enough keys may be stolen, leaked or disused—without having terminated their trust relationships—to pose a serious, ongoing threat to an organization. The December 2014 Sony hack included the leak of SSH keys as well as password lists, which led to the compromise of related services and accounts; mishandled SSH keys may have even facilitated the initial compromise.[13]

Attackers can exploit vulnerable implementations of an SSH client or server to gain unauthorized access to critical assets such as applications, systems or communications channels, as demonstrated during attacks in 2014 against Russian banks by a group calling itself Anunak. Press reports estimated Anunak was able to steal more than $25 million.[14]

The process of compromise is not that complicated. An attacker generates a new key pair and adds the new authorized key to the authorized key file, which is often unaudited. The attacker can then use the associated identity key to log into the server or application unnoticed. Once attackers gain access, they can pivot their attention to other systems owned by the victim—a narrowly focused risk that, ironically, becomes greater as organizations improve their overall level of security with widespread adoption of SSH key infrastructures.

[11] *CIS Critical Security Controls for Effective Cyber Defense Version 6.0, Center for Internet Security*, www.cisecurity.org/critical-controls.cfm

[12] "Root SSH Key Compromised in Emergency Alerting Systems," SecurityWeek, July 8, 2013, www.securityweek.com/root-ssh-key-compromised-emergency-broadcast-systems

[13] "Sony Pictures Hacked – Employee Details & Movies Leaked," Darknet.org.uk, December 4, 2013, www.darknet.org.uk/2014/12/sony-pictures-hacked-employee-details-movies-leaked; "More Trouble For Sony? PlayStation Servers 'Used To Spread Stolen Data,'" Forbes, December 3, 2014, www.forbes.com/sites/thomasbrewster/2014/12/03/sony-playstation-serving-hacked-data

[14] "Cybercrime group steals millions from Russian banks, targets U.S. and European retailers," PCWorld, December 22, 2014, www.pcworld.com/article/2862312/cybercrime-group-steals-millions-from-russian-banks-targets-us-and-european-retailers.html

Remediating Vulnerabilities with Best Practices

Configured correctly, SSH keys are harder to crack, steal or guess than passwords—reason enough to consider replacing password-based authentication with SSH keys, despite the complexities. Organizations can avoid many issues involving SSH by tackling the vulnerabilities through best practices. The CIS Controls provide a structured framework for vulnerability remediation, while the U.S. National Institute of Standards and Technology (NIST) interagency report, "Security of Automated Access Management Using Secure Shell (SSH)," outlines SSH's vulnerabilities and recommended best practices for managing the protocol's deployment.[15] The recommendations in the NIST report and related instructions from the CIS Controls map are shown in Table 1.

Table 1. SSH Best Practices

Vulnerability Root Cause	CIS Control	NIST Best-Practice Recommendations
Configuration	**Inventory of Authorized and Unauthorized Devices (CSC 1)**: 1.1, 1.4 and 1.6 **Inventory of Authorized and Unauthorized Software (CSC 2)**: 2.1 through 2.3 **Secure Configurations for Hardware and Software on Mobile Devices, Laptops, Workstations, and Servers (CSC 3):** 3.1 and 3.7	Establish approved baselines, first on hardware inventory and then software. Centrally manage SSH configuration across an organization. Implement configuration and/or change management around SSH keys, trust relationships and privileged accounts. Incorporate trust relationships and privileged access granted via SSH keys into a privileged access management system for consolidated and regular review. Only enable SSH server functionality on systems where absolutely required; associate unique identity keys for specific, well-defined identities.
Implementation	**CSC 1:** 1.6 **CSC 3:** 3.1, 3.4 through 3.7 **Controlled Use of Administrative Privileges (CSC 5):** 5.1, 5.2, 5.6, 5.8 and 5.9 **Limitation and Control of Network Ports, Protocols, and Services (CSC 9):** 9.1 through 9.3 **Data Protection (CSC 13):** 13.1 and 13.7 **Controlled Access Based on the Need to Know (CSC 14):** 14.2 **Application Software Security (CSC 18):** 18.6	Harden SSH implementations and keep updated. Disable SSH v1. Disable unapproved authentication methods. Have a distinct, unique host key for each host using SSH for automated access. Prevent implicit access by limited SSH-accessible accounts and groups (including "root"). Disable port forwarding. Limit access to environment variables. Use only approved ciphers/algorithms. Properly configure supporting subsystems used in conjunction with SSH. Enforce SSH inactivity timeouts. Configure incoming and outgoing trust relations in separate accounts to prevent pivoting. Separate the SSH trust relations that cross security zone boundaries, such as keys used for development and production environments (see also CIS Control 18-6). Define compensating Controls to encapsulate vulnerable SSH traffic where needed (e.g., some appliances and embedded devices) (CIS Control 14.2). Require less-than-privileged access for those SSH-accessible accounts used by automated processes (system) and for remote access (CIS Control 5-1). Establish authorized key command and source restrictions as needed.

[15] *CIS Critical Security Controls for Effective Cyber Defense Version 6.0*, Center for Internet Security, www.cisecurity.org/critical-controls.cfm

Table 1. SSH Best Practices (Continued)

Vulnerability Root Cause	CIS Control	NIST Best-Practice Recommendations	
Keys and Provisioning	**CSC 3:** 3.1, 3.5 and 3.7 **CSC 5:** 5.6 **CSC 13:** 13.4 **CSC 14:** 14.6	Automate key provisioning processes. Keep complete inventory of enabled SSH identity keys in the organization. Terminate identity keys when an employee leaves the organization. Enforce standards for minimum key lengths used with approved algorithms (NIST SP 800-131A).[16] Establish maximum time that an identity key can be used before replacement (NIST SP 800-57).[17]	Use passphrases to protect identity keys and establish approved procedures. Regularly rotate SSH keys, including authorized keys, in similar fashion to other credentials (e.g., passwords).
Access Issues	**CSC 5:** All subcontrols (as applied to keys, rather than passwords) **CSC 14:** 14.4 and 14.6 **Account Monitoring and Control (CSC 16):** All subcontrols	Lock down your keys. Prevent non-superusers from installing new authorized keys for user accounts. Restrict access on identity keys to interactive use, automated process or named admin staff responsible for managing keys. Protect identity keys against theft.	Grant identity keys only the minimum privileges necessary. Minimize number of keys that provide administrator-level access or have privilege escalation capabilities. Allow only authorized administrators to modify authorized keys. Add key management to privileged account management systems.

[16] NIST Special Publication 800-131A, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," January 2011, http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf

[17] NIST Special Publication 800-57, "Recommendation for Key Management – Part 1: General (Revision 3)," July 2012, http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf

Table 1. SSH Best Practices (Continued)

Vulnerability Root Cause	CIS Control	NIST Best-Practice Recommendations
Misuse of SSH	**Continuous Vulnerability Assessment and Remediation (CSC 4):** 4.1 through 4.3, 4.6 through 4.8 **Maintenance, Monitoring, and Analysis of Audit Logs (CSC 6):** 6.6 **CSC 9: All Continuous Vulnerability Assessment and Remediation (CSC 4):** 4.1 through 4.3, 4.6 through 4.8 Maintenance, Monitoring, and Analysis of Audit Logs (CSC 6): 6.6 CSC 9: All subcontrols **Secure Configurations for Network Devices such as Firewalls, Routers, and Switches (CSC 11):** All subcontrols **Boundary Defense (CSC 12):** All subcontrols **CSC 14:** 14.6	Monitor and audit all use of SSH. Watch all changes to authorized key files. Audit SSH connections to ensure their use is only for intended purposes.

Of all the controls listed in Table 1, CIS Controls 1 through 5 should receive priority in developing a strategy around SSH key management. Applying these controls can answer some fundamental questions for an organization:[18]

- What is connecting to our infrastructure using SSH? (CSC 1)
- Where is SSH software running (or trying to run) within our infrastructure? (CSC 2)
- Are we continuously managing our SSH deployment using "known good" configurations? (CSC 3)
- Can we monitor, identify and remediate any "known bad" configurations? (CSC 4)
- Can we limit and track the people who have the administrative privileges to manage our SSH implementation, key management and privileged account provisioning? (CSC 5)

Organizations successfully addressing these first five CIS Control families establish a strong foundation for SSH security with a clear growth path for maturity beyond these basic actions.

[18] *CIS Critical Security Controls for Effective Cyber Defense Version 6.0*, Appendix D: The National Cyber Hygiene Campaign, Center for Internet Security, www.cisecurity.org/critical-controls.cfm

According to NIST IR 7966, the cost of manual labor in an SSH remediation project is often significantly greater than the cost of the automation used. The first step in developing a baseline is for an organization to not only gain visibility into where its SSH hosts reside but also understand SSH-related behaviors associated with these hosts. Organizations should typically monitor each SSH host for at least six months to establish a normal baseline of SSH activity. Security teams need a formalized, documented approach that they can use to achieve desired outcomes, one that can defend against the loss of project momentum, balance project resources (staff, time and money), and provide clear insight into progress in the form of a reduced risk to the organization.

To achieve these objectives, we look to integrate and balance two approaches: the five steps of formal configuration management (CM) and the five steps of a methodology around successful CIS Control implementation. CM provides the project management structure that defines formal baselines, maintains those baselines by formal processes that balance project resources, and provides accounting as to the status of the implemented baselines. The CIS Control approach follows a pattern of steps that have emerged as common to organizations that have made substantial progress in reducing risk using the Critical Security Controls.

Figure 3 diagrams how these two approaches can provide a coordinated set of activities that promote successful SSH remediation.

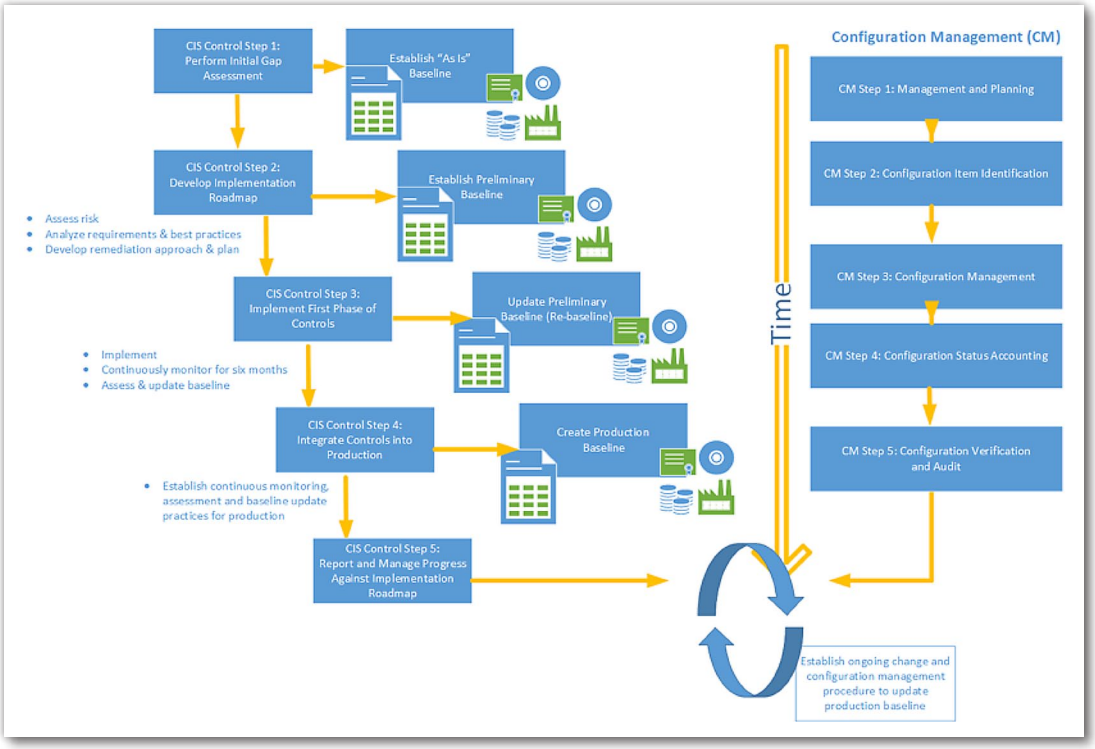


Figure 3. SSH Remediation: An Integrated Approach

Table 2 suggests a set of reliable guidelines on how to avoid most pitfalls of complex remediation projects.

Table 2. Approach for SSH Key and Account Remediation

Phase/Objective	Recommended Activities
Management and Planning (CM Step 1)	Create a formal document and plan to guide the SSH remediation effort following standard project management processes (e.g., as defined by the Project Management Institute). It should define roles, responsibilities and resources; training requirements; configuration procedures and tools; baseline generation processes; and configuration control and configuration status accounting for developing and maintaining the SSH baseline. Perform risk assessment for the current environment with a goal of balancing business and security requirements against threats of compromise or attack. Step back from where the organization has implemented SSH to allow discovery of a wide range of complexities in the infrastructure that might influence SSH remediation: data classification and protection policies, access control model(s) and harmonization, and a detailed map of interfaces and intersystem processes that demonstrates where automation is paramount and where trust relationships will likely exist. Review current identity and access management processes, evaluating the use of SSH key management as it relates to privileged accounts. Examine current policies and procedures, both formal and informal, around general and privileged access as well as the use of automation to understand the impact of securing access management with SSH. ***Outcomes:*** ***1) Project Plan, 2) SSH CM Plan, 3) Initial Risk Assessment, 4) Infrastructure Review***
Configuration Identification (CM Step 2), **Initial Gap Assessment** (CIS Control Step 1), and **Develop Implementation Plan** (CIS Control Step 2)	Establish configuration items and the basis by which changes to any part of the SSH environment are identified and tracked throughout their lifetime. This can include the keys, the trust relationships, the SSH software, and documented policies and procedures. Establish the "as-is" baseline: Create an inventory of all existing SSH keys, including the locations of related hosts and accounts, identity and authorized keys and authorized key restrictions (i.e., source and command). Map all existing trust relationships that have resulted from deployed identity and authorized keys. Include policies and procedures in establishing baseline requirements for SSH usage. Address the best-practice recommendations in Table 1. Don't neglect remote, off-premises systems or any cloud-based applications. Make sure roles and responsibilities are spelled out for key and access management, and don't forget to classify keys as sensitive data. Evaluate the results. Which keys do not have appropriate approval? Do authorized keys have proper source and command restrictions, consistent with what is needed for proper access? Are identity keys protected by a passphrase? Are there authorized keys that need to be retired? How many trust relationships are implied? What authentication methods are in use? What is the potential for using SSH to pivot in the initiation of an attack? How are keys being used for privileged account access? ***Outcomes:*** ***1) "As-Is" Baseline, 2) Remediation Plan***

Table 2. Approach for SSH Key and Account Remediation (Continued)

Phase/Objective	Recommended Activities
Configuration Control (CM Step 3), **Configuration Status Accounting** (CM Step 4), and **Implement First Phase of Controls** (CIS Control Step 3)	Update the "as-is" baseline to reflect the impact of remediation plans, updated processes and new requirements on the SSH configuration items (CIs). Place the initial baseline under formal configuration control as the approved baseline. Control key generation according to the current approved baseline so that configuration, as enforced by policy, is maintained (e.g., keys are generated centrally and per policy to enforce key length and algorithms). Changes to this and all subsequent baselines will require evaluation of change requests and subsequent approval or disapproval. Establish management procedures based on appropriate automation tools, identifying existing processes and tools to repurpose or more fully utilize, as well as new tools to acquire and processes to enhance. Look to automation for the following: • **Inventory of all SSH identity and authorized keys.** Select an inventory tool that can integrate if possible with any existing configuration management database and processes around change and configuration management. • **Management of trust relationships.** Select a privileged access management system, if one is not already available, that can import this data and facilitate an accurate review and tracking of approvals for access granted through SSH keys. • **Management of SSH key life cycles.** Select a tool that interfaces to standard account provisioning and management tools and supports the best practices noted previously, like rotating identity keys. • **Continuous monitoring and audit.** Automation is virtually a necessity for any continuous monitoring process. An audit system that can provide central reporting and analysis tools across the inventory can reduce the time necessary to prepare the data required for audits and make it easier for auditors to verify proper implementation and identify exceptions. To validate initial remediation and reconfiguration efforts, monitor and analyze SSH use for at least six months to confirm the initial baseline—identify the normal activities around SSH keys and the keys that should be removed due to lack of use, expiration date or lack of compliance with policy. Use selected CIS Control effectiveness metrics to evaluate and report on outcomes related to remediation and reconfiguration. Perform configuration status accounting to document all departures from the baseline during this time, allowing for quick approval of modifications in the case of suspected problems. Re-baseline the initial baseline if needed through the CM process to establish a new approved baseline ***Outcomes:*** ***1) Management Procedures, 2) Automation Tools, 3) New Approved Baseline***

Table 2. Approach for SSH Key and Account Remediation (Continued)

Phase/Objective	Recommended Activities
CM Verification and Audit (CM Step 5), and **Integrate Controls into Production** (CIS Control Step 4)	Audit (i.e., independent review) of the resulting SSH environment to assess compliance with established performance requirements, best practices and documentation before acceptance into the production baseline. At this point, the entire process is in production use and subject to the configuration/change management processes that the organization establishes for SSH. ***Outcomes:*** ***1) Production Baseline, 2) Production Change Management***

Following the steps in Table 2 can help an organization remediate its SSH infrastructure, but to continue the investment made, it must establish ongoing configuration and change management practices lest it return to a compromised infrastructure. Plan to transition procedures developed during remediation into the change management and configuration management processes that have been established for the organizational infrastructure as a whole.

Conclusion

Widely used, SSH is an important protocol that provides encrypted, authenticated communications in a variety of configurations. Public key authentication is especially attractive, as it automates access and can easily provide ease of login to accounts with elevated privileges. However, SSH is also especially vulnerable to complexities that involve configuration, implementation and management. Reducing risk requires consistent attention to and implementation of best-practice recommendations, both during implementation and afterward. The CIS Controls provide an excellent framework for guiding the implementation (and, in some cases, automation) of SSH and evaluating outcomes with effectiveness metrics.

In a Fortune 500 organization that may have upwards of a million keys, remediation of an environment where SSH keys and trust relationships have never been fully managed is complex, labor-intensive and time-consuming. Here the CIS Control implementation methodology provides a practical approach to control implementation. However, integration with more formal configuration management processes can truly ensure success of an important project that, given its demands on enterprise resources over a substantial time, might otherwise stall.

About the Author

Barb Filkins, a senior SANS analyst who holds the CISSP and SANS GSEC (Gold), GCH (Gold), GLSC (Gold), and GCPM (Silver) certifications, has done extensive work in system procurement, vendor selection and vendor negotiations as a systems engineering and infrastructure design consultant. She is deeply involved with HIPAA security issues in the health and human services industry, with clients ranging from federal agencies (Department of Defense and Department of Veterans Affairs) to municipalities and commercial businesses. She focuses on issues related to automation—privacy, identity theft and exposure to fraud, as well as the legal aspects of enforcing information security in today's mobile and cloud environments.

Sponsor

SANS would like to thank its sponsor:

Last Updated: January 15th, 2018

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Las Vegas 2018	Las Vegas, NVUS	Jan 28, 2018 - Feb 02, 2018	Live Event
Cyber Threat Intelligence Summit & Training 2018	Bethesda, MDUS	Jan 29, 2018 - Feb 05, 2018	Live Event
SANS Miami 2018	Miami, FLUS	Jan 29, 2018 - Feb 03, 2018	Live Event
SANS London February 2018	London, GB	Feb 05, 2018 - Feb 10, 2018	Live Event
SANS Scottsdale 2018	Scottsdale, AZUS	Feb 05, 2018 - Feb 10, 2018	Live Event
SANS SEC455: SIEM Design Beta One 2018	Arlington, VAUS	Feb 12, 2018 - Feb 13, 2018	Live Event
SANS Southern California- Anaheim 2018	Anaheim, CAUS	Feb 12, 2018 - Feb 17, 2018	Live Event
SANS Secure India 2018	Bangalore, IN	Feb 12, 2018 - Feb 17, 2018	Live Event
SANS Brussels February 2018	Brussels, BE	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Secure Japan 2018	Tokyo, JP	Feb 19, 2018 - Mar 03, 2018	Live Event
Cloud Security Summit & Training 2018	San Diego, CAUS	Feb 19, 2018 - Feb 26, 2018	Live Event
SANS Dallas 2018	Dallas, TXUS	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS New York City Winter 2018	New York, NYUS	Feb 26, 2018 - Mar 03, 2018	Live Event
CyberThreat Summit 2018	London, GB	Feb 27, 2018 - Feb 28, 2018	Live Event
SANS London March 2018	London, GB	Mar 05, 2018 - Mar 10, 2018	Live Event
SANS Secure Singapore 2018	Singapore, SG	Mar 12, 2018 - Mar 24, 2018	Live Event
SANS San Francisco Spring 2018	San Francisco, CAUS	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS Secure Osaka 2018	Osaka, JP	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS Paris March 2018	Paris, FR	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS Northern VA Spring - Tysons 2018	McLean, VAUS	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS Munich March 2018	Munich, DE	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Pen Test Austin 2018	Austin, TXUS	Mar 19, 2018 - Mar 24, 2018	Live Event
ICS Security Summit & Training 2018	Orlando, FLUS	Mar 19, 2018 - Mar 26, 2018	Live Event
SANS Secure Canberra 2018	Canberra, AU	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Boston Spring 2018	Boston, MAUS	Mar 25, 2018 - Mar 30, 2018	Live Event
SANS 2018	Orlando, FLUS	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS Abu Dhabi 2018	Abu Dhabi, AE	Apr 07, 2018 - Apr 12, 2018	Live Event
Pre-RSA® Conference Training	San Francisco, CAUS	Apr 11, 2018 - Apr 16, 2018	Live Event
SANS Dubai 2018	OnlineAE	Jan 27, 2018 - Feb 01, 2018	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced