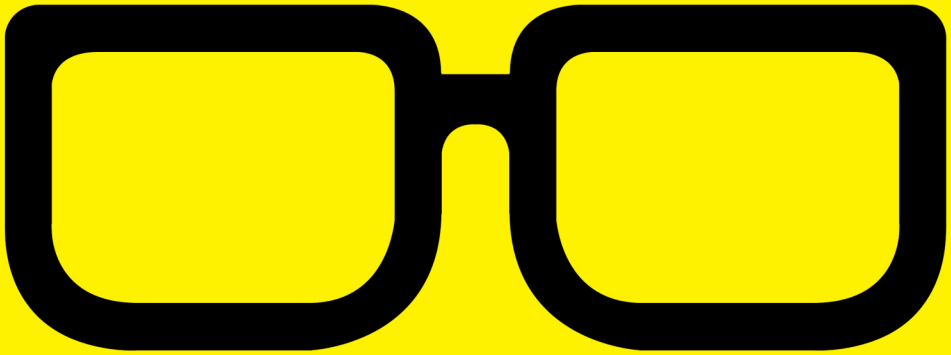
SPONSORED BY

foxtechnologies.

GEEK GUIDE



SSH: a Modern Lock for Your Server?

LINUX™ JOURNAL

Table of Contents

FEDERICO KEREKI is a Uruguayan systems engineer with more than 25 years of experience doing consulting work, developing systems and teaching at universities. He is currently working as a UI Architect at Globant, using a good mixture of development frameworks, programming tools and operating systems—and **FLOSS**, whenever possible! He has written several articles on security, software development and other subjects for *Linux Journal*, IBM developerWorks and other Web sites and publications. He also wrote the *Essential GWT* book. You can reach Federico at fkereki@gmail.com.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About the Sponsor

Fox Technologies

Fox Technologies is a global security company that helps organizations centralize Linux and Unix identity and access management across hybrid IT environments. Enterprises worldwide rely on FoxT's security solutions to enforce granular security controls, simplify compliance, and increase overall IT department efficiencies. By empowering IT and security teams with control over security—they can proactively prevent internal and external critical system attacks—before they start. Fox Technologies has been a leader in the data security industry for more than 30 years. They are trusted by some of the world's top fortune 500 companies, and protect over 20 trillion dollars in assets.

Fox Technologies' BoKS® ServerControl solution (http://www.foxt.com/boks) transforms your multi-vendor Linux and Unix server environment into one centrally managed security domain. It simplifies your organization's ability to enforce security policies and control access to critical systems and information. With full control over accounts, access and privilege—IT and security teams can proactively prevent internal and external critical system attacks—before they start.

To learn more, please visit: http://www.foxt.com.

SSH: a Modern Lock for Your Server?

FEDERICO KEREKI

Introduction

In modern computer systems, privacy and security are mandatory. However, connections from the outside over public networks automatically imply risks. One easily available solution to avoid eavesdroppers' attempts is SSH. But, its wide adoption during the past 21 years has made it a target for attackers, so hardening your system properly is a must.

Additionally, in highly regulated markets, you must comply with specific operational requirements, proving that you conform to standards and even that you have

If you find SSH-1 in your infrastructure, stop using it now!

included new mandatory authentication methods, such as two-factor authentication.

In this ebook, I discuss SSH and how to configure and manage it to guarantee that your network is safe, your data is secure and that you comply with relevant regulations.

What Is SSH?

SSH stands for Secure Shell, an encrypted protocol to provide secure connections over unsecured networks. It was created to send data back and forth between clients and servers, using encryption so attackers can't examine the exchanges to access private data.

SSH is a proposed standard, produced by the Internet Engineering Task Force (IETF: http://www.ietf.org). It was developed by Tatu Ylönen, a Finnish researcher, to replace earlier protocols that didn't ensure confidentiality. SSH can be used for user-to-machine sessions and for machine-to-machine (M2M) connections (the latter is common in multi-server environments).

SSH has two versions: SSH-1 (from 1995) and SSH-2 (a standard since 2006). SSH-1 is now deprecated, because it's vulnerable to certain attacks. If you find SSH-1 in your infrastructure, stop using it now!

The most typical implementation of SSH is OpenSSH, available not only for UNIX, Linux and Mac OS X, but also for Windows.

Windows Does SSH?

SSH is classically a Linux/UNIX tool, so you might think it doesn't apply to Windows users. But, if people want to use SSH in Windows, there are a couple well known options: Cygwin, which is a Linux-like console (http://cygwin.com), and PuTTY, which is a terminal emulator (http://www.chiark.greenend.org.uk/~sgtatham/putty). You also could add WinSCP (Windows Secure Copy: http://winscp.net) and other programs to the mix.

SSH usage may become even more popular in the future, as Microsoft is including OpenSSH in Windows (see http://blogs.msdn.com/b/powershell/archive/2015/10/19/openssh-for-windows-update.aspx), and check GitHub at https://github.com/PowerShell/Win32-OpenSSH). Note: OpenSSH for Windows will not just be a powershell extension. It has its own ANSI-compliant terminal emulator separate from cmd.exe or powershell.exe.

Is SSH Unbreakable?

SSH security is based on math with very large numbers, so systematically trying all possibilities becomes practically impossible. Of course, the ever-increasing power of computers (and possible algorithmic breakthroughs) means that what's unreadable today may be readable tomorrow.

This discussion becomes moot if there are defects in applications or configurations, so a determined eavesdropper with plenty of computer power eventually may be able to break in to your communications. According

to WikiLeaks, the NSA (National Security Agency) may decode, at least sometimes, SSH communications. (Edward Snowden's attack on the NSA was helped by unmanaged SSH keys that provided undetected access; it wasn't a protocol problem, but rather a management one.) Read the *Der Spiegel* article "Prying Eyes: Inside the NSA's War on Internet Security" at http://www.spiegel.de/international/germany/inside-the-nsa-s-war-on-internet-security-a-1010361.html or "How the NSA can break trillions of encrypted Web and VPN connections" at http://arstechnica.com/security/2015/10/how-the-nsa-can-break-trillions-of-encrypted-web-and-vpn-connections for more information.

Tips for Hardening SSH

Once you've installed SSH, you need to make sure it is configured safely and does not provide entrances for intruders. Let's look at a few methods that can make things more difficult for would-be attackers—just remember, there's no "silver bullet" for security; you never can take too few precautions.

Change the Standard SSH Port: The standard SSH port is 22, so a server with that port open becomes a target for attackers. You might get an extra bit of protection by applying the well known (and well derided) concept of "security through obscurity" by changing the port number to a nonstandard number. Most ports above 1000 are safe, but be sure to avoid reserved ones (see http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt for more details).

Changing the port number requires changing the `Port` parameter in the SSH configuration file. Users then will have to specify the new port when connecting. Script kiddies, upon finding port 22 closed, will skip your server—although this change probably won't really make a difference against serious attacks.

One negative with this approach is that the extra configuration details may confuse users. Also, some monitoring tools may detect traffic on the alternate port as suspect and return false positives. Take all of these considerations into account when making security decisions.

Make Users Knock for Access: When running a public server, if you go overboard with protection measures, you likely will hinder valid users. However, when providing access to just a few people, you can hide the SSH port, which won't even be visible unless users know how to make it appear. (For *Lord of the Rings* fans: this is like the "Doors of Durin" at Moria, visible only under moonlight after speaking certain words.) Port knocking is simple to implement, requires little resources and works by keeping ports invisible until users provide a secret "knock", which then (and only then) will open the port for their exclusive use.

The idea is to close all ports, but monitor connection attempts. Whenever a specific sequence of attempts is recognized, the system can open a port and provide access to your server. The "knock sequence" can be simple (like first trying TCP port 7060, then UDP port 7009 and finally TCP port 7022), or it can be a "use-only-once" sequence from a list, as with cryptographically highly secure "one-time pads".

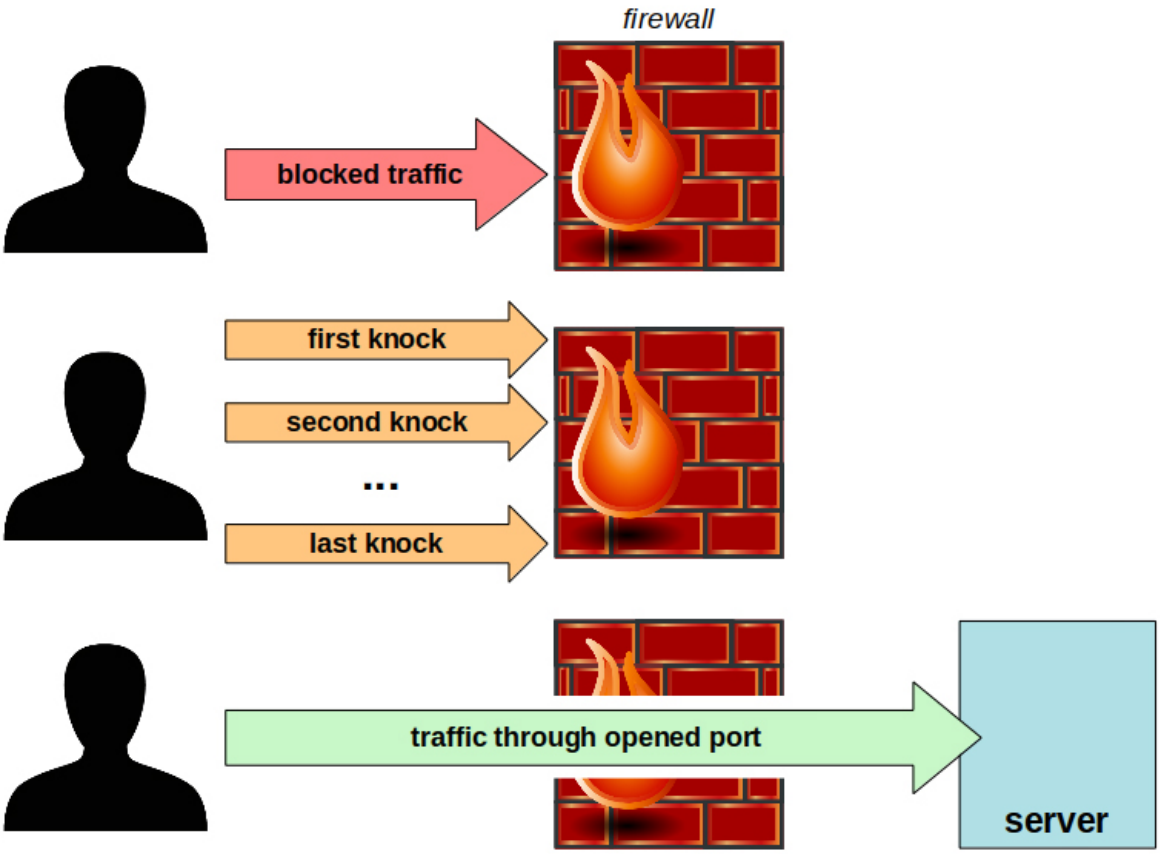


FIGURE 1. With port knocking, a port won't even be visible unless a secret sequence of "knocks" is provided, which will cause the port to open.

Another option is to provide a timeout, so that after the port is opened, the user has to connect quickly. And even then, the port will be opened for that user and no one else, so others can't take advantage of the access window. Why does this work? There are more than 64,000 available ports, and after discarding pre-assigned ones, you are left with more than 50,000 possibilities. Guessing a five-knock sequence would mean trying millions and millions of possible combinations.

From a management point of view, think of each port-knocking sequence as a password and treat it the same way (with expiration times, forced changes and so on). Port knocking can't be your only security protection ("hope is not a strategy", as the saying goes), but it can make it more difficult for others to attack your computer.

Avoid Configuration Weaknesses: The first SSH protocol (SSH-1) was vulnerable to man-in-the-middle attacks, so eavesdroppers could intercept your communications and read your (supposedly secure) traffic. Most distributions' SSH setups allow only SSH-2, but it's a good idea to confirm that Protocol 2 is included in your configuration file. Also do the following:

- Although not common today, rhosts sometimes was used to authenticate systems. Disable that by adding `IgnoreRhosts yes` to SSH's configuration file.
- If you won't be doing X11 forwarding, set `X11Forwarding no` to impede possible attacks.
- Set `DSAAuthentication no` to disable weak DSA authentication.
- Don't ever allow users to work without passwords: set `PermitEmptyPasswords no`.
- Set `PermitRootLogin no` so nobody can log in as root. Users who need to connect and work as root should log in as common users (that is, unprivileged and as restricted as possible) and then use sudo.

Weaknesses don't just come from bad parameters; users also are a risk, and unattended SSH sessions can provide unauthorized access in a ridiculous way. To minimize this particular problem, do the following:

- Set `ClientAliveInterval 300`, so after five minutes (the timeout must be given in seconds) with no activity from the SSH client, a message will be sent requiring an answer. Upon receiving no answer, and after `ClientAliveCountMax` attempts (see below), the user will be logged out. The default value is 0, implying no automatic logouts.

- Set `ClientAliveCountMax 1`, so as soon as there is no response from the SSH client, the logout process will be executed. The default value is 3, so the user will have three options to answer (in this case, 15 minutes) before being forced out.

Prefer Keys over Passwords: Passwords are widely used, but they are not very secure due to the following reasons:

- Social engineering attacks are quite successful at getting passwords from users.

- Users are not very good at selecting hard-to-guess passwords (see http://www.teamsid.com/worst-passwords-2015).

- Some users write passwords down, removing all pretenses of security.

- Keyboard sniffers or viruses can fish out passwords easily.

With public key cryptography, you can do without passwords. After creating a public/private pair of keys, install the public part on all the servers you want to access, but keep the private key safe on your local machine or stored in a USB keystore.

In your server's SSH configuration file, set `PubkeyAuthentication yes` to enable the new method, and if/when you feel confident that everything is right, add `PasswordAuthentication no` and `ChallengeResponseAuthentication no` to disable all other methods.

Creating a key pair requires `ssh-keygen` plus `ssh-copy-id` to send the public part to the server. Use a good passphrase to protect your key. This means you will be able to take it anywhere (like on a USB stick) to log in to remote servers. Otherwise, losing your stick would compromise every server to which you connect.

Try logging in to the machine and check to make sure that only the key(s) you wanted were added. After this, try to log in to the remote host; you should succeed without entering a password. If you used a passphrase when generating the key, you'll be asked for it. To avoid re-entering it every time, use `ssh-agent` (do `man ssh-agent` or read http://linux.die.net/man/1/ssh-agent for more information).

If you have access to several servers, your public key will be replicated on them all. You can use LDAP as a common store, and SSH can access it to get keys. I won't go into

implementation details here, but it's a good combination for centralized passwordless access.

Limit Password-Based Logins: If you can't do just with public keys and need password-based authentication after all, do the following:

- Set a maximum time limit to log in with `LoginGraceTime 30`, so if a user hasn't managed to enter the password in 30 seconds, the login process will have to be restarted.

- Set a maximum number of retries with `MaxAuthTries 3`, so after three wrong attempts at entering a password, the connection will be broken.

- For extra security, you can implement host denying techniques to block users for longer periods of time.

Enable Access Rules: SSH includes rules (`AllowUsers`, `DenyUsers`, `AllowGroups` and `DenyGroups`) that can provide or deny access. Patterns, using * and ? wild cards, also are possible. SSH first applies `DenyUsers` (so users blocked thereby won't be allowed in, no matter what other rules may indicate) followed in sequence by `AllowUsers`, `DenyGroups` and `AllowGroups`.

Rules can take users' source IPs into account, so for example, you could allow access to anybody from the internal network, but only a few people from the outside. SSH accepts connections over all network interfaces, but if your server has "trusted" and "untrusted" interfaces, you can accept connections only from trusted parts of the

network with the `ListenAddress` option.

You can add even more specific rules, but if things start getting out of hand, use PAM (see below) so rules can be changed on the fly, without needing to restart SSH.

Use PAM (Pluggable Authentication Modules) for Checks: If every program had to implement its own authentication logic, things would be a mess, because nobody could be certain that every application included the same tests, made the same checks and correctly implemented the same code. And, if extra controls were needed, everything would have to be reprogrammed!

With Linux servers, whenever a program needs some authentication task, it can call the PAM API, which will run all the required checks in its configuration files. If you modify such files on the fly, all PAM-aware programs instantly will apply the new rules. And, to include new devices (such as USB tokens or iris scanners), you simply need to add the corresponding PAM modules.

PAM deals with four security areas, identified by specific keywords:

- Account limitations: what is a valid user allowed to do, and when?
- Auth (authentication) details: how is a valid user recognized?
- Password-related functions, including password changing, for example.
- Session management, including connection and logging.

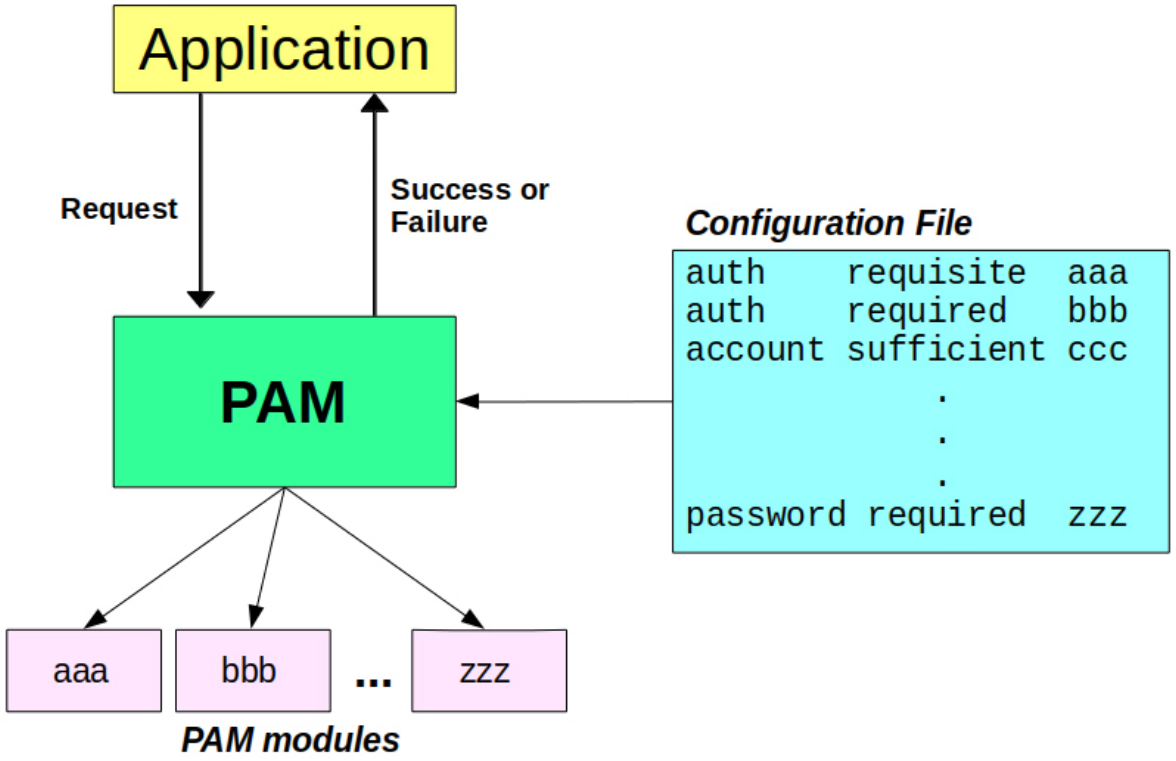


FIGURE 2. Whenever an application makes a security request, PAM executes modules as specified in its configuration file to approve or reject said request.

You can specify a series of modules to run in sequence, implying checks to be performed. Depending on those checks, user requests will be approved or denied.

A check can be required (it must succeed to approve a request, and if it fails, the request will be denied, but all other modules will run anyway) or optional (at least one must succeed to get approval). Modules also can be requisite (their failure instantly produces a rejection, without more modules) or sufficient (their

success automatically implies an approval).

The list of all available PAM modules is quite long; see http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html for an incomplete list. There are modules for user/password authentication (and to enforce good practices for passwords), for LDAP, for certificates and for other methods for identifying a user. You can use modules for specific restrictions (as mentioned previously), and you won't have to restart SSH in order for the new rules to be incorporated, so you can specify your own stack of checks dynamically.

Block Brute-Force Attacks: Publicly available servers always are targets for attack. Security logs are full of "authentication failure" lines, derived from brute-force attacks that try one possible password after another to attempt to get into your system—and given enough time, such attacks eventually will succeed. A three-pronged way to deal with them requires first detecting entry attempts, then deciding whether they should be considered real attacks and finally denying access for a sufficiently long while to stop would-be intruders on their tracks.

DenyHosts, available at http://www.denyhosts.sourceforge.net, can help, but also consider other tools like BlockHosts (http://www.aczoom.com/tools/blockhosts) and Fail2Ban (http://www.fail2ban.org). DenyHosts usually runs in the background, checking access logs, analyzing access attempts and locking out attacker IPs when the number of consecutive failures reaches a threshold.

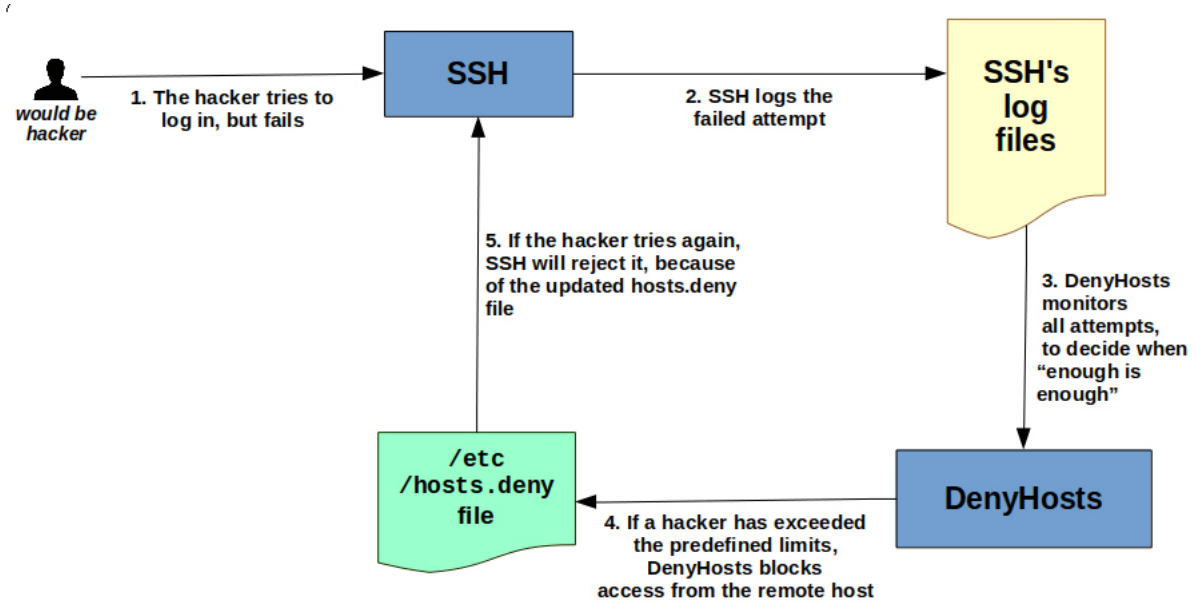


FIGURE 3. **DenyHosts continuously monitors access logs to detect and lock out attackers' hosts.**

DenyHosts instances can share information with other instances to build "IP blacklists". Whenever an IP is banned, it is sent to a central server. If the IP has been detected by enough distinct servers, it will be blacklisted and broadcast to all participating servers for their protection.

Host denying won't hinder valid users, but it will stave off malicious attempts and even some Denial of Service (DoS) attacks. It doesn't require any specific management (it works automatically, after all), so all you need to do is verify that it's up and running—a "fire-and-forget" weapon against attackers.

SSH Roots

There are more than 20 official IETF RFC (request for comments) documents on SSH; the full list is at https://datatracker.ietf.org/doc/search/?name=ssh&sort=&rfcs=on, but the principal ones are RFC 4250 to RFC 4256 (https://tools.ietf.org/html/rfc4250 through https://tools.ietf.org/html/rfc4256). For official OpenSSH documentation, check the manual pages on ssh, sshd, ssh-keygen and all the rest of the SSH suite (http://www.openssh.com/manual.html), or use the `man` command directly.

Upgrading the Lock: SSH Management

So far, I've shown several ways to harden SSH for normal use. However, in highly regulated businesses, having provably safe methods and automated tools in place is a must, so let's consider SSH management more in depth.

Standards, Policies and Compliance Requirements: In the modern world, compliance is increasingly important, and many security-related standards exist, such as the following:

- Army Regulation AR 25-2 for the military.
- Basel II and Basel III Accords for banking institutions.
- FISMA (Federal Information Security Management Act) for US federal agencies.

All of these standards and laws include security considerations and minimum configuration requirements, and failure to comply even can lead to jail time, typically for sysadmins and CTO/CISOs.

- GLBA (Gramm-Leach-Bliley Act) for banking and financial services companies.
- HIPPA (Health Insurance Portability and Accountability Act) for health-care organizations.
- PCI-DSS (Payment Card Industry—Data Security Standard) for organizations that work with credit cards.
- Sarbanes Oxley Act (SOX) for public company boards, management or public accounting firms.
- And, of course, ISO 27001, for all kinds of companies.

All of these standards and laws include security considerations and minimum configuration requirements, and failure to comply even can lead to jail time, typically for sysadmins and CTO/CISOs. SSH usage can provide a way to access (without permission) a server and then hack into sensitive information, so let's consider

how it should be managed and what characteristics should be required for tools to accomplish that.

SSH Management Risks: SSH is used in most network environments, but not all companies manage access adequately, possibly providing pathways for intruders who then can exploit unapproved rights. Passwords aren't the safest way to get access, and SSH keys do not expire and may not be under the overview of security teams, so they both pose security risks.

Here are some common problems:

- Visibility: do you know all current possible accesses using SSH? Who can access what? How is SSH used and secured? Without centralized control, private SSH keys could end up in the hands of unauthorized users, without being noticed by security teams.

- Level: are users restricted to their least-possible access level? Are there users with high-level access, even though they do not (or no longer) need it?

- Rogue keys: can you detect rogue keys? How would you do it? With audits? Log analysis? The presence of such keys may be due to a simple operations error or a full-scale attack that adds its own keys to provide back-door access.

- Rotation: are keys changed on a periodic basis? Is it done automatically? Or, does it depend on manual processes, which implies higher costs, more time and probability of errors? Shorter rotation times do not imply higher security automatically (attackers may have introduced

back doors or new keys unaffected by rotations), but long periods certainly are worrisome.

- Termination: are keys revoked when users leave the company? Is the process controlled to make sure access is revoked everywhere? If a user has new, different access requirements, are old keys revoked before issuing new ones?

- Human mistakes: what happens if a configuration is edited incorrectly and access is granted to unauthorized users? Or what if keys that grant access to development environments also are used mistakenly for production environments? How do you guard against such errors?

- Updating: are security patches applied regularly in a controlled manner? Many security agencies report on software vulnerabilities or produce security recommendations that require software or process updates. Are those applied?

- Auditing: are audits performed regularly? Are their results acted upon? Several of the standards mentioned previously mandate that companies protect, control and track all access to privileged accounts, implying the need to secure and monitor the usage of SSH keys. The lack of centralized control may result in failed audits.

The typical results of audits are that large numbers of (valid, current, usable) SSH keys are no longer necessary and even may correspond to people no longer authorized to use

Doing SSH key management by hand adds several problems of its own.

them. Even worse, some of those keys may allow the key granting process itself to be compromised, in which case the potential security problem would escalate exponentially.

SSH by itself is not the problem. Rather, the issues stem from the way companies manage it in the face of an ever-more severe compliance environment, the growth of the threats landscape (including intruders and malware), and the many problems that might be caused by the high-level access that SSH users can have. Doing SSH key management by hand adds several problems of its own. Beyond the possibility of missed or misapplied processes, there is a hidden cost involved by having IT staff manually doing tasks that could be automated, with safer and more consistent results.

A 2014 Forrester Research report shows some disturbing results (http://www.venafi.com/assets/pdf/wp/Gaps_In_SSH_Security_Create_An_Open_Door_For_Attackers.pdf):

- 36% of companies do not perform regular system scans to detect unauthorized added keys.

- 19% of companies do not rotate SSH keys, and 54% of companies do it on a less than yearly basis, so keys may end up being used for more than a year in practically three out of four companies.

- 40% of companies depend on sysadmins for detection of rogue keys instead of systematic methods.

- In 34% of studied companies, SSH keys are secured by operations or sysadmin people, rather than by IT security.

Another 2014 study by the Ponemon Institute shows similar negative results (http://www.venafi.com/assets/pdf/Ponemon_2014_SSH_Security_Vulnerability_Report.pdf):

- About 50% of companies had been impacted by SSH key-related compromises in the last two years.

- 82% of companies do not rotate keys or do it on a less than once a year basis. (The Forrester report number in this category was 73%.)

- 60% of companies said that they couldn't detect new SSH keys that were introduced into their networks.

- 68% of companies had no automated processes for SSH key policy enforcement; usually, sysadmins are responsible for control and management of keys, possibly giving would-be attackers unrestricted access to servers.

What is behind these problems? It's not a matter of simple oversights, but rather a combination of several complicated factors. SSH keys typically are not regenerated, as expiring and renewing an SSH host key on a single system requires a major outage. Here are some problems related to this:

- Checking all operational and data processes that utilize the key can take days, since later you also will need to check that all processes still work after the change.

- Deleting and updating a key itself is practically immediate; however, testing that all processes still work can take hours—on average two hours per OS.

- If you have 50 production servers that need keys to be renewed during a specific six-hour change window on a Saturday, manual processes just will not hack it; you will need an automation framework.

- And worse, financial services and Critical National Infrastructure (CNI) are moving to a six- to one-month rolling renewal cycle for SSH host keys. So with such mandated shorter renewal cycles for SSH host keys, a "business-as-usual" automated process will be required; manual methods will no longer be feasible.

Software Management Requirements: An appropriate process for managing SSH should include:

- Centralized control and storage: keys for all servers (both physical and cloud-based) should be managed centrally. This allows for globally enabling or disabling keys. Key management should not depend on independent sysadmins, but rather should come under the purview of a centralized security group.

- Key rotation: keys must be rotated periodically, just as with common passwords. Rotation periods should not be overly long, and there should be a way to assign one-time-access keys, if required for business reasons.

- Usage policies: there should be policies to limit usage of SSH keys and reduce the possibility of expanded access. A baseline for normal, valid, key usage should be defined.

- Usage logging: there should be reports on who used what keys, when and for how long. These reports will let you know what systems were accessed and by whom.

- Move quickly to SSH two-factor authentication: financial services, government and CNI sectors already have moved quickly (in 2014/2015) to requiring that all servers have mandatory SSH 2FA configured. For example, PCI/DSS requirement 8.3 requires its incorporation, and various technologies (tokens, cards and even mobile phones, despite some disadvantages) are available for this.

- Continuous monitoring: automated procedures must scan for rogue key usage. Given correct levels of usage logging, it becomes feasible to provide a way to detect anomalies, such as unexpected or unusual accesses. These controls shouldn't depend on sysadmin users, the same way that malware detection shouldn't depend on individual users.

- SSH configuration control: an even worse case of

wrongful key usage is if the configuration files themselves for your SSH setup are accessed and modified. After such a change, security policies for the involved server would be defeated, and that machine would become a point for further attacks on your infrastructure. Access to SSH configuration should be closely limited and monitored, and all changes should be reported and reviewed instantly

- Commercial SSH products that centralize SSH configuration and control: standard OpenSSH depends on changes to many config and key files on each and every server, as I already mentioned. If your root account has been pwned, your SSH config on that server also is compromised, and there is no central register to notice a change has been made. Commercial OpenSSH-based products move the active SSH config to secure networked databases, accessed via PAM. If the local files are compromised, your SSH security keeps working.

- Vulnerability fixing: server and SSH configurations should be updated according to current best practices. This includes the need for software updates, especially in the case of discovered security "holes". Specific allowed cryptographic methods also should be updated as needed once details of possible weaknesses are revealed or current methods become prone to attacks.

- Software patching: due to its worldwide deployment and mind numbingly detailed analysis by security researchers

and sysadmins, OpenSSH typically has somewhere between 10–20 CVEs (Common Vulnerabilities and Exposures) reported annually, many with critical or major classifications. If you are not patching SSH at least two to six times per year, your organization could be a) non-compliant and b) you as an individual sysadmin or your DevOps team *and* your organization together may be legally liable in the event of a breach.

- Require minimum key sizes: from OpenSSH 6.X onward, you can define minimum acceptable key sizes and report on existing keys that do not meet those criteria, making them candidates for a quick renewal.

- Compliance verification: by managing and controlling access to SSH configuration and SSH keys, companies comply with standards and policies, and the produced reports can show their compliance. All types of audit processes should be made possible by the software itself.

Conclusion

In this ebook, I have described various methods for hardening SSH and have considered desirable aspects of an SSH management systems in terms of security and of compliance with standards.

SSH isn't likely to go away, but it can be managed in provably secure ways if the right processes and tools are used. All servers are liable to be attacked, but sane, modern policies can diminish those perils, making SSH a valuable, quite usable, tool for your environment.■

Resources

Books that cover some SSH fundamentals:

- Daniel Barrett and Richard Silverman, *SSH: The Secure Shell*, O'Reilly, 2001.
- Himanshu Dwivedi, *Implementing SSH: Strategies for Optimizing the Secure Shell*, Wiley Publishing, Inc., 2004.
- Michael Stahnke, *Pro OpenSSH*, Apress, 2006.
- Brad Sibley, *OpenSSH—A Survival Guide for Secure Shell Handling*, Sans Press, 2003.

The following are articles I have written on SSH security:

- "PAM—Securing Linux Boxes Everywhere", *Linux Journal*, January 2009: http://www.linuxjournal.com/magazine/pammdashsecuring-linux-boxes-everywhere
- "The rsync family", IBM developerWorks, April 2009: http://www.ibm.com/developerworks/aix/library/au-rsyncfamily
- "Implement Port-Knocking Security with knockd", *Linux Journal*, January 2010: http://www.linuxjournal.com/magazine/implement-port-knocking-security-knockd
- "Three locks for your SSH door", IBM developerWorks, August 2010: http://www.ibm.com/developerworks/aix/library/au-sshlocks

- "More locks for your SSH door", IBM developerWorks, September 2011: http://www.ibm.com/developerworks/aix/library/au-moresshlocks

- "More Secure SSH Connections", *Linux Journal*, January 2014: http://www.linuxjournal.com/content/more-secure-ssh-connections

- "Security in Three Ds: Detect, Decide, and Deny", *Linux Journal*, January 2015: http://www.linuxjournal.com/content/security-three-ds-detect-decide-and-deny

Since SSH is pretty mature, even though these texts or articles may be some years old, they are actually still valid. However, there have been several advances on operational requirements and configuration settings since then, and these management aspects are being constantly updated; see just a few examples:

- Payment Card Industry (PCI) Data Security Standard version 3.1, dated April 2015, at http://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf

- National Institute of Standards and Technology (NIST) at http://csrc.nist.gov/publications, specifically "NIST Internal Report 7966 — Security of Interactive and Automated Access Management using SSH", dated October 2015, at http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf