Architecture and Configuration for Hardened SSH Keys

Scott Ross

Architecture and Configuration for Hardened SSH Keys

GIAC (GSEC) Gold Certification

Author: Scott Ross, c11scott.ross@gmail.com
Advisor: David Fletcher, david@blackhillsinfosec.com

Accepted: October 1, 2020

Abstract

The Secure Shell (SSH) protocol is a tool often-used to administer Unix-like computers, transfer files, and forward ports securely and remotely. Security can be quite robust for SSH when implemented correctly, and yet it is also user-friendly for developers familiar with Unix. Asymmetric SSH keys used by the protocol have allowed operations engineers and developers to authenticate to remote machines – supporting increased automation and orchestration across DevOps environments. While the private keys should be password protected, they are often not. The fast pace of DevOps and the focus on delivery has led to many companies not controlling their authentication credentials or understanding the risk they create. Private key files can become scattered around the environment, presenting a tempting target for threat actor exploitation to pivot across a network or access cloud services. This paper will evaluate a simple solution for protecting private keys by storing them on an external cryptographic device (Yubikey) and automating key management/SSH configuration (Ansible). This potential solution will be compared to local key storage and prevalent ad-hoc key management against conventional SSH attack techniques in the MITRE ATT&CK matrix.

1. Introduction

The Secure Shell (SSH) is a mainstay protocol in managing computers and devices at scale. SSH v2 (SSH-2) was proposed in 1995 and is the only currently supported version of the protocol; it is defined in RFC 4251 and consists of three layers: user authentication (RFC 4252), transport (RFC 4253), and connection (RFC 4254). Since its inception, SSH expanded in many ways to implement new authentication modes, newer cryptographic ciphers, and other extensions. The SSH protocol RFCs are listed below.

SSH PROTOCOLS			
Name	Purpose	RFC	Requires
Transport	provides SERVER authentication, confidentiality, integrity, compression (optional)	4253	TCP/IP
Authentication	provides USER authentication for public key, password, and host-based methods	4252	Transport layer established
Connection	provides interactive login sessions, remote execution of commands, forwarded TCP/IP connections, and forwarded X11 connections multiplexed over a single channel	4254	Authentication layer established

Figure 1 IETF RFC library

SSH is robust and straightforward to use, allowing authenticated users to access computing resources that are local or remote behind several authenticated gateway jumps. This access can be automated and built upon with other tools. Git, Ansible, Vagrant, and many other standard DevOps software packages use SSH for secure file transfer (via SFTP, SCP, or rsync), terminal access, port forwarding, and much more. SSH can be used to set up a new server on the cloud, enforce a secure configuration, push files, and run an X Windows application on a server with the display sent to a local machine. SSH authentication can also enable some forms of single sign-on (SSO). Each connection will use symmetric encryption negotiated between the server and the client. The server and the client can authenticate using public-key cryptography. This forward secrecy and authentication protect against eavesdropping on the network and offer some protection against machine-in-the-middle attacks (as long as the configuration enforces host key validation). The overall security, flexibility, and widespread use of SSH make it an indispensable tool for IT Operations, developers, and security teams to understand and manage well. SSH is usually the primary way for administrators to access critical systems, making it even more critical to secure the authentication of users and configuration of SSH servers (NIST 7966 p18).

Scott Ross

Large scale deployments of SSH can have hundreds of thousands of keys with millions of security associations (Filkins 2015). Since SSH can provide authenticated access to many systems, it is an attractive target for network attackers to leverage. MITRE ATT&CK has identified SSH credentials (through either passwords or public keys) as a part of several tactics that adversaries have used to pivot within an organization (MITRE 2020). If an administrator computer is compromised and an attacker discovers unencrypted SSH private keys, they would have instant access to any computers the administrator had connected to with the private keys. Sometimes developers will even hard-code unprotected private keys/passwords into scripts or load them to public repositories, leading to potential compromise. Real-world examples such as the SONY hack[1], Chalubo botnet[2], Ebury malware[3], Chaos malware[4], and even exposed private keys on Github[5] all highlight the importance of proper SSH access management and configuration. Unfortunately, many organizations do not know how many SSH keys they have, what the keys grant access to, or who has the keys (NIST IR 7966 p12).

SSH has several options for authenticating clients to a host machine, with passwords being most common for interactive users and public/private key pairs being most common for systems at scale (NIST IR 7966 p6). The public-key authentication mechanism relies on servers and clients with an asymmetric key pair available to them. Servers authenticate themselves to the client using a public key saved by the client for subsequent connections. This process prevents machine-in-the-middle attacks after the server and client establish initial trust. A client can authenticate if a server saves their key in the proper location. The server uses this public key to encrypt a random challenge to the client. This process allows secure, automated authentication as long as the client has access to the private key – and the private key is not compromised. Configuration options for both the SSH client and server also are critical in securing the connection. Misconfigurations can allow SSH to use weak cryptography, not validate hosts, expose private keys on intermediate jump hosts, and not correctly limit authenticated users' access. While proper configuration is essential, detailed

1 Shukhman 2019
2 Easton 2018
3 MITRE 2020
4 MITRE 2020
5 Matrix 2019

configuration is beyond the scope of this paper. The Appendix contains resources for SSH server and client configuration.

With the usefulness of SSH in automation, cloud, and other DevOps tools only increasing, more organizations are setting up SSH infrastructure for multiple users. However, the speed of change in DevOps makes proper key and certificate management very challenging (Bocek 2019). Lack of proper configuration and key management opens yet another avenue for attackers to compromise a network. An attacker only needs one key to break an enterprise trust model, and authenticated sessions can be difficult to identify as malicious (Webb 2016). Organizations can purchase SSH key management software to help them with these challenges, but smaller organizations may struggle to justify the cost and continue to practice ad-hoc key management. Pairing external cryptography devices with open-source configuration management software mitigates many of the vulnerabilities in SSH. This paper will compare the vulnerabilities inherent in ad-hoc key management with a simplified solution using open-source software and a GPG card compatible cryptographic device.

2. SSH and Key Infrastructure

While SSH was created in the late 1990s as a replacement for Telnet and remote shell, its uses have expanded to include the ability to tunnel other protocols within itself and even run some graphical applications over an SSH session. There are many implementations of the SSH protocol today, including Dropbear (IoT), OpenSSH (BSD/Linux/and recently Windows), Putty (Windows), and Paramiko (Python), among others. Dropbear and OpenSSH are the overwhelming majority of SSH implementations in use today (Albrecht et al. 2016).

2.1 Authentication Methods

The Secure Shell uses host key checking for verifying servers to clients, and several authentication modes for clients including password (keyboard-interactive and PAM), generic security service application program interface (GSSAPI, including Kerberos), and public key (including host-based authentication and certificates). OpenSSH also offers challenge-response authentication and Fast Identity Online/ Universal Second Factor (FIDO/U2F)

authentication since version 8.2 as a part of keyboard-interactive authentication (Cimpanu 2020). Host authentication occurs during the SSH-transport session establishment, after the exchange of session keys. Client authentication occurs in the SSH-authentication protocol after the host server authentication. Each client authentication method has its strengths in certain situations, and SSH can require many combinations of methods or to explicitly block specific methods for individual users. OpenSSH for Windows currently only supports password and public-key authentication (Microsoft 2019). This paper will primarily focus on securing public-key authentication for Linux on OpenSSH, but other methods will be briefly summarized below to present their relative advantages. Challenge-response mode and FIDO/U2F will not be summarized, but they are a useful addition to any primary authentication method and are especially suited to multipurpose devices like the Yubikey, discussed in Section 3.

2.1.0 Host Key Authentication

Host key authentication (not to be confused with Host-based authentication) occurs after the key exchange negotiation step during the establishment of the SSH-transport protocol and is the primary way a client authenticates a server. Authentication of a server is vital to prevent machine-in-the-middle (MiTM) attacks and can use either certificates or public keys. Whenever a client connects to a host, it presents a list of host key types it will accept. The host responds with its public key, part of the session key, and its signature, showing it has possession of the private key corresponding to the public key. The client then verifies the host key based on a certificate or public key in the local known hosts file. If there is only a public key, and the client has not connected to this host before, the fingerprint (key grip) of the key will be displayed, asking the user if they want to trust the targeted host. This process is called trust on first use (TOFU), and the user should verify the key grip via another communication channel. If the host's public key matches the public key stored in the 'known hosts' file, the client will accept the host as verified. Invalid or changed certificates will show an error. Sometimes users will disable strict host key verification in environments where servers are ephemeral and change keys; this increases the risk of a MiTM attack. Valid host certificates do not require TOFU since they are trusted if the certificate authority (CA) is trusted.

```
curly@Linux-Base:/home/ansible$ ssh linux-base-server
The authenticity of host 'linux-base-server (10.8.8.11)' can't be established.
RSA key fingerprint is SHA256:lJdsiMkN96Wg5FYO5JCkWOL5H4KMumcXSfbM5jfktRO.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'linux-base-server,10.8.8.11' (RSA) to the list of known hosts.
```

Figure 2: First connection from a host to an SSH server showing TOFU

```
curly@Linux-Base:/home/ansible$ ssh linux-base-server
Last login: Wed Sep  2 16:41:55 2020 from 10.8.8.10
$ _
```

Figure 3: Subsequent connections from host to a known SSH server

```
curly@Linux-Base:/home/ansible$ ssh linux-base-server
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:iHqOB7PvEuVrbaYglcIGjjPy4wWZyPu1zt42Hj2IVJE.
Please contact your system administrator.
Add correct host key in /home/curly/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/curly/.ssh/known_hosts:1
  remove with:
  ssh-keygen -f "/home/curly/.ssh/known_hosts" -R "linux-base-server"
RSA host key for linux-base-server has changed and you have requested strict checking.
Host key verification failed.
curly@Linux-Base:/home/ansible$
```

Figure 4: A known SSH server with changed keys - a possible indicator of MiTM. Note that default SSH configuration disables this connection.

2.1.1 Password Authentication

Password authentication includes both traditional password mode (for backward compatibility with SSH-1) and keyboard-interactive mode. Keyboard interactive mode uses the operating system for authentication (including options for Pluggable Authentication Module (PAM) in UNIX systems) and can include other modes of challenge-response, allowing for FIDO or RSA tokens. The password authentication method can also reference an LDAP server for user authentication. LDAP can be a very robust authentication method if it enforces multi-factor authentication (MFA) and reasonable password complexity

requirements[6]. An advantage of passwords is that they can easily have an expiration and enforced complexity; users also do not have to worry about moving and securing private keys. As the number of computers managed by this method increases, it becomes more difficult to have unique passwords for each system (if there is no LDAP). Especially in large environments, this may increase password reuse. Passwords are also susceptible to brute-force attacks, especially generically named accounts such as 'root'.

2.1.2 GSSAPI Authentication

GSSAPI authentication includes methods such as Kerberos, where a centralized server authenticates users, and a temporary ticket (Ticket Granting Service or TGS) is passed to the client for authentication to the SSH service. GSSAPI SSH authentication inherits Kerberos' vulnerabilities, such as 'Kerberoasting', golden ticket attacks, etc. An attacker who manages to get a valid ticket-granting ticket (TGT) should also get a TGS that may be accepted by an SSH server within the Kerberos domain if the user has the proper authorization for the service. Kerberos alone is not a recommended authentication method because it creates large implicit trust relationships. Some SSH implementations have no command restriction for Kerberos accounts, and may not work in NAT environments (NIST IR 7966 p14). Interestingly, Kerberos environments with service authentication also make it possible to use SSH without host keys on servers.

2.1.3 Public-Key Authentication

Public-key infrastructure (PKI) is the most commonly used authentication method (especially for automation work) and is usually the recommended solution, according to NIST IR7966. Many different algorithms can generate asymmetric key pairs, including RSA, DSA, ECDSA, and ED25519. The SSH-keygen program creates keys in an ASCII-formatted text file. Existing SSL or GPG keys can also be converted and used for SSH as long as they use a supported algorithm and key size. The gpg-agent can even act as an SSH-agent, using GPG keys stored within the agent, on a card, or within a token (such as a Yubikey) for all SSH

6 NIST no longer recommends certain complexity requirements and expiration (NIST 800-63-3), but other compliance frameworks or organizational policies still do.

authentication without exposing the private key. Private keys can be password protected and are only stored unencrypted in memory for use when authenticating. Since using a private key requires a password (something you know) plus access to the key file (something you have), it can meet MFA requirements as long as a reasonable password policy enforces key passwords.

Public keys are widely shareable, and an SSH server will store the public key of each authorized user in an authorized key file. The client connecting to a server must reply to a challenge encrypted by the public key associated with the account for which it is requesting access. Only the clients private key can decrypt this challenge, and the client responds back to the server to prove the user has access to the authenticating private key. Alternatively, a certificate authority (CA) can be stored and trusted on the server or client that allows any public key signed by the CA key to be trusted. Certificates have the advantage of having potential expiration dates and easier management at scale than standard public keys. Automated systems can even provide certificates daily, dynamically scoped to allow access to only necessary resources for that day; this is similar to the model Google used to make BeyondCorp (Dweyer 2017). Public key methods also include host-based authentication, where any user from a specific host can authenticate to a server based on a global private key stored on the SSH client. Host-based authentication is generally discouraged when used alone because it allows broad access to many users where it becomes harder to audit trust relationships and attribute actions to users. The security of any PKI based system hangs on protecting and limiting access to private keys which will be the focus of Section 3.

2.1.4 Authentication Trade-offs

There are several trade-offs between authentication methods from a usability standpoint. In a DevOps environment, users may need to connect from different clients and access different systems on an irregular basis. LDAP or SSO tools like GSSAPI can be effective, but also require management overhead and added complexity that may not be worth the cost, especially in smaller or temporary environments where PKI and password authentication are common. Passwords can be difficult to manage in these environments because they should be: unique for each system, adequately complex, and must expire. PKI can have public keys shared between systems, allowing a single private key to access many

systems. The key can expire (through using certificates) or never expire depending on requirements. Administrators must manually remove keys not using certificates from the authorized keys files on a host.

Private keys are inherently more complex than passwords and are much harder to steal or brute-force (Filkins 2015, p9). Passwords can be burdensome to enter when authenticating to many systems in succession. PKI makes automated authentication quick and easy, even when keys are password or pin-protected since a private key can be cached in memory for a configurable amount of time. Environments relying on passwords for SSH authentication will often find numerous people using the same accounts and sharing passwords; these passwords may even be hard-coded into scripts for ease of use. It is similarly problematic for PKI when users share private keys among a group, however storing private keys on cryptographic devices mitigates this risk.

Managing a password environment is also a nightmare, requiring constant resets of passwords for systems not regularly accessed. A compromised user password may be valid on many systems and for other enterprise services. Managing PKI for many users can also be a challenge, with most work being done upfront during enrollment or when deactivating accounts. The end-users currently do PKI enrollment in many smaller shops. This ad-hoc management can create a problem when users create many key pairs for themselves that are scattered and shared throughout the environment, with each private key potentially providing access to many different systems. The longer a key pair is in use, the more permissions it likely acquires (Dweyer 2017). Proper controls and management, specifically certificates with an expiration, mitigate this 'key creep'.

	Password	Host-Based	Kerberos	Public Key
Supports Portability (User not required to carry credential from system to system)	Yes	Yes	Yes	Yes/No[8]
Provides Single Sign-On	No	No	Yes	Yes
Enforces Command Restrictions	No	No	No	Yes
Prevents Man-in-the-Middle[9]	No	Yes	Yes	Yes
Minimizes Implicit Access	Yes/No[10]	Yes	No	Yes
Supports NAT Environments	Yes	Yes	No	Yes[11]

Figure 6: Comparison of authentication methods for SSH (NIST pg.17)

With passwords, users can reliably access the resources they need from any connected computer as long as they remember their password. PKI needs access to the private key to work; users need to securely move and store their key to authenticate from a different client. External cryptographic devices that store private keys (such as smart cards or gpg cards) solve this mobility challenge. PKI authentication solves many of the problems of remembering passwords, and it is easy to add users to new systems. Finally, SSH has configuration options that allow users to run only specified commands on a host. This command restriction option is only available with PKI. For these reasons, and due to its widespread use, this paper will focus on securing and hardening PKI.

2.2 Managing and Securing SSH Keys

Since PKI is both the most prevalent and the recommended way to authenticate using SSH, proper key management is vital to its overall security. Specific architecture, configuration, and management options relating to securing PKI authentication will be discussed below in more detail.

Both private and public keys should be protected when stored in configuration files. Private keys need to be protected from disclosure or modification, while public keys need to be protected against modification. Consider the following examples with an ACME employee named Bob and an attacker named Alice:

1. Bob's compromised private key would allow Alice to authenticate as Bob anywhere his public key is used. She could even find valid SSH servers by examining Bob's shell history or SSH known hosts file. This attack is an excellent way for an adversary to pivot through an environment.

2. If Alice tampered with Bob's known hosts file to switch the ACME Inc. public keys, Alice could impersonate ACME Inc. when Bob next authenticates to ACME and possibly steal sensitive information (like sudo password).
3. If an ACME server stores Bob's public key in an authorized keys file in his home directory, and that file is appended by Alice to list her public key in addition to Bob's, Alice could authenticate as Bob to that ACME server even if Bob changes his password.
4. If Bob's laptop is compromised, Alice may authenticate as Bob when he has unlocked his private key by passing SSH connection requests to the SSH-agent. Alternatively, Alice could install a key logger and steal Bob's private key and private key password.
5. Bob could also use an intermediate jump, ACME_gateway, to authenticate to an internal host ACME_internal. If Alice has compromised the intermediate host, she could pass authentication requests to Bob's SSH-agent unbeknownst to him for ACME_internal_2. Alice could then authenticate as Bob to any host reachable from ACME_gateway.

It is crucial for organizations to know where these keys are stored at rest, where the private key is stored unencrypted, agent caching mechanisms, and common attacks against SSH. We will then look at and test potential mitigations against these attack vectors.

2.2.0 Key Storage Locations

SSH can be configured to use key files in various locations on disk or on a separate cryptographic device. While SSH clients and servers can have non-standard key locations specified at runtime, this is not a standard or suggested practice. Understanding where keys are stored allows security teams and system administrators to focus efforts on protecting these crucial locations to prevent adversaries from quickly pivoting through an environment. OpenSSH is the primary SSH server/client software in use for Linux and recently also for Windows. OpenSSH for Linux and Windows stores keys in the following locations by default. All stored keys must have the proper file permissions not to be viewed or modified by unauthorized users.

AUTHORIZED KEYS		
~/.ssh/authorized_keys	Primary	Linux
~/.ssh/authorized_keys2	Secondary	Linux
~/.ssh/authorized_keys	Users	Windows
~/.ssh/authorized_keys2	Users	Windows
%programdata%/ssh/ administrators_authorized_keys	Administrators	Windows
/etc/ssh/sshd_config	Global Cert – use TrustedUserCAKeys cakey.pub	Linux
~/.ssh/authorized_keys	User Cert – use directive @cert-authority cakey.pub	Linux

Figure 7 SSH authorized keys file locations.

The authorized keys file is stored on an SSH host and is used to authenticate clients using the SSH-authentication protocol. A client sends the server a message with an SSH session identifier, user name, service name, public-key algorithm, and a public key signed by the private key. This information is enough to validate that the current client is attempting to log in as a particular user, and they have both the public and private keys of a key pair. The host can then determine if the provided public key is valid for the user attempting to log in. The authentication request will succeed if the public key is listed in the user's authorized keys file (or the Administrator's key file in Windows), or if a CA trusted by the host signs the public key.

KNOWN HOSTS		
~/.ssh/known_hosts	user	Linux
/etc/ssh/ssh_known_hosts	System	Linux
~/.ssh/known_hosts	Primary	Windows
/etc/ssh/ssh_known_hosts	System-cert – use @cert-authority	Linux
~/.ssh/known_hosts	User-cert – use @cert-authority	Linux

Figure 8 SSH Known hosts file locations.

The known hosts file is stored on the SSH client and is used to authenticate hosts when the client initiates a connection. This process takes place in the SSH-transport protocol after the key exchange establishing symmetric session keys. A client will trust a host with a public key already in the client's known hosts file, or if a trusted CA signs the public key. The hosts in the known hosts file can be hashed values of the IP or domain name; this lessens an attacker's ability to know where they can pivot next if they compromise a user's keys.

HOST KEYS		
/etc/ssh/ssh_host_xxx_key	Private	Linux
/etc/ssh/ssh_host_xxx_key.pub	Public	Linux
%programdata%/ssh/ssh_host_xxxx_key	Private	Windows
%programdata%/ssh/ssh_host_xxxx_key.pub	Public	Windows
/etc/ssh/ssh_host_xxx_key-cert.pub	Host-Certiificate – must add HostCertificate {path} to */etc*/ssh/sshd_config	Linux

Figure 9 SSH host key locations

The SSH server stores host private keys, which should be unique for each system. They should not be encrypted with a passphrase because the SSH daemon must automatically use them to sign the host authentication message. The entire selected host public key is a part of the host authentication message. Alternatively, SSH servers can also present a certificate of their public key signed by a trusted CA.

CLIENT KEYS		
~/.ssh/id_xxxx (private)	Primary	LINUX
~/.ssh/id_xxxx.pub (public)	Primary	LINUX
~/.ssh/id_xxxx.-cert.pub (certificate)	Primary	LINUX
/etc/ssh/ssh_config – add IdentityFile {path}	Alternate -Global	LINUX
~/.ssh/config – add IdentityFile {path}	Alternate	LINUX
C:\Users\{username}\.ssh\id_xxxx (private)	Primary	WINDOWS
C:\Users\{username}\.ssh\id_xxxx.pub (public)	Primary	WINDOWS

Figure 10 SSH client (user) key locations

The user computer or external cryptographic device, which will be discussed in later sections, should be the only place that securely stores user keys. These keys should be password protected. An agent stores the private keys in memory for a configurable period after decrypting them for use, such as when signing an authentication request. If an external cryptographic device holds the keys, the agent must have access to the client public keys or certificates since the authentication process provides both parts to a host.

2.2.1 Caching and Encryption

Private keys should be encrypted with a strong passphrase when at rest on disk. When SSH decrypts private keys for use, it can store them in an agent running in memory. SSH sends the key operation requests to a socket where the agent can respond with the desired signature or encryption. The agent does not send the private key anywhere; it only uses the private key to sign/encrypt. OpenSSH includes an SSH-agent by default, but gpg-agent can serve as a drop-in replacement for the SSH-agent to allow using GPG keys for SSH

authentication. Gpg-agent also allows for using a cryptographic device such as a Yubikey for safe key storage[7]. Each agent has different settings for how long it will cache credentials before requiring the user to decrypt the key again. There are options for setting the lifespan of keys in the agent (-t), for asking for confirmation on each use of a key (-c), and for locking/unlocking access to the agent with a password (-x/-X, note this is separate from the private key password). Options allow SSH with GPG to forward the agent socket in the connection to the host. This agent forwarding option allows a user to SSH through an intermediate gateway to a host without transferring their private keys to the gateway computer. Windows official support for gpg-agent SSH use is limited but possible with open-source npiperelay and wsl-ssh-pagent (Pye 2020).

AGENT SOCKETS (SSH_AUTH_SOCK)		
$TMPDIR/ssh-XXXXXX/agent.<ppid>.	Ssh-agent	LINUX
~/.gnupg/S.gpg-agent	Gpg-agent	LINUX
*depend on putty or openssh	Ssh-agent	WINDOWS
\\.\pipe\winssh-pageant	Gpg-agent	WINDOWS

Figure 11: Agent socket locations

2.2.2 Attacking Authentication

There are several ways that an attacker can compromise the confidentiality, integrity, or availability of SSH using public-key authentication. Attackers can directly steal keypairs from a user, use keys without the owner's knowledge (through sockets), add new authorized keys to a host, and impersonate legitimate SSH servers. These techniques encompass the scope of this paper. The next two attack classes are worth consideration, but are out of scope for this paper. Attackers can modify SSH config files for a variety of effects, including running unwanted commands on user login or disabling access. Attackers can compromise SSH cryptography, and the Cipher Block Chain (CBC) mode of SSH symmetric ciphers is especially vulnerable to attacks (CERT 2008). Proper configuration mitigates both of these risks.

Stealing keys is possible if a user unknowingly exposes their private key (including a possible passphrase) to the public. There have been many recorded instances of developers uploading secret keys with code onto Github or other repositories (Filkins 2015). These

7 gpg-agent does not cache the pin when used with a Yubikey, the Yubikey itself will cache it (Dr. Duh 2020)

compromises can be costly if they are keys to a public cloud service, as attackers can use cloud resources to mine cryptocurrency (leaving the victim to pay for the resources). Keys uploaded accidentally by Uber developers allowed an attacker to gain privileged access to Uber's AWS account (Miller 2018). Keys can be read from memory, if not on a hardware token. Hardware tokens/cryptographic devices are also vulnerable to theft, but because attackers cannot feasibly duplicate their content, a user should know if their key is missing[8]. Once attackers steal keys, they can use them to brute-force access to other computers. The Chalubo botnet (Easton 2018) and CHAOS malware used a similar method where they collected a large number of key pairs then randomly brute-forced computers (MITRE 2020). Targeted attacks on an organization can be even more effective if an administrator's key pair is stolen since administrators often have access to more machines at a higher level of authorization. An attacker can also comb through the 'known hosts' file to determine which machines they can pivot to with the user's credentials (Lodge 2016). Old keys from past employees can also count as stolen keys since those keys should no longer be authorized. Because standard SSH keys never expire, a user will never lose access to a system if there is no configuration management to disable accounts and old keys.

Keys can sometimes be used for authentication without an attacker directly compromising the key; this can happen when using an agent socket. An attacker with elevated privileges on a compromised computer can send authentication requests to the agent. If SSH forwards the agent from a remote client, the attacker could still use the agent on the remote host. As long as the key is cached in the agent memory (assuming the agent is not locked) and the attacker sends a valid request to the agent (with access permissions to the socket), the agent will sign the request. An attacker could compromise a jump or gateway host and wait for a user to connect to the compromised host with agent forwarding enabled. The attacker could then send requests to the forwarded user agent and authenticate to any host the user has access to that is reachable from the compromised gateway. This technique can allow an attacker to rapidly pivot through an environment, especially if the user keys they compromised have broad access.

8 Cryptography side-channel attacks (timing/power/etc.) are possible for both hosts and cryptographic devices like the Yubikey, but are out of scope for this paper.

Disabling key validation or asking a user to connect to a new host the attacker has created (TOFU) could create a man-in-the-middle attack (MITRE technique T1557 MiTM). An attacker could impersonate a known server, proxying authentication information to the real server and inspecting/modifying the traffic en route. The attacker could gain access to any sensitive information passed over the session, including sudo passwords or contents of printed files. With traffic tampering, an attacker could even run commands as the user on the server. While this would not allow the attacker to compromise the user's key directly, it opens access to the server and to potentially running code on the server. Even simple attacks without proxying can be effective. An attacker could temporarily redirect a victim trying to SSH to an attacker-controlled machine and prompt them for a password. The user may assume there was some sort of innocuous error and try one of their passwords to connect. The attacker then has a password and username that may be valid on other computers in the environment.

If an attacker compromises a system, they can add their keys to create persistence in the environment. Adding the key allows the attacker to connect back any time they choose as either a targeted user (MITRE technique T1078) or a new user (T1126) using a new key pair the attacker controls. System-required password changes or expiration would not limit this access. These forms of persistence can be easy to miss in large environments with many users or in environments where a system does not manage the 'authorized keys' files since the attacker is logging in much like a regular authorized user would. These attacks correspond with MITRE ATT&CK tactics listed in Appendix B.

2.2.3 Hardening Authentication

Barbara Filkins identified several vulnerability root causes in her 2015 white paper on securing SSH with the critical security controls: Configuration, Implementation, Keys and Provisioning, Access Issues, and Misuse of SSH (Filkins 2017 p. 9-11). While these vulnerabilities are comprehensive, this paper focuses on mitigating vulnerabilities to the authentication mechanisms of public keys in SSH. To simplify the issue, we will evaluate a solution focusing on Architecture (includes aspects of keys and provisioning, access issues, and misuse of SSH) and configuration (includes configuration and implementation). The general attack patterns covered so far can be mitigated with a combination of architecture and configuration.

2.2.3.1 Architecture

Secure design for an SSH system takes the different attack vectors into account, but with any change there is a cost. Organizations looking to harden their SSH configuration need to ensure any solutions are functional for end-users and maintainable in terms of budget, staffing, and technical expertise. Consequently, solutions that use established protocols and tools, well-supported software, and can be easily automated, are generally better. The solution proposed in this paper will use a combination of external crypto devices (GPG cards), SSH certificates, and provisioning software (Ansible).

Storing user private keys on an external crypto device shields them from all but the most dedicated and advanced attacker. Rather than exposing the private key on disk (like an unencrypted key file) or in memory (like a key stored in SSH-agent or GPG-agent), an external cryptographic device keeps private keys off of a system on a separate device. The popular GPG software supports PGP cards; they are generally in either a smart card or USB format. The Yubikey is a USB cryptographic device that supports many different modes, including a PGP card. The Yubikey is rugged and has anti-tamper features[9]. It can require a pin and touch button on the device for every key operation it performs (Fong-Jones 2019). Yubikeys cannot have the private key read or copied directly off the device. They also will lock the device if too many incorrect pin entries are attempted, unlike password-protected private keys stored on disk. These features can mitigate several attack patterns, including abusing sockets (SSH session hijacking T1563.001) and stealing keys (Unsecured credentials T1552.004).

Using key certificates instead of standard public keys is another architecture decision that offers several advantages. Firstly, key certificates offer a better way to authenticate SSH servers to the client on a client's first connection; this bypasses the potential for a MITM (Man-in-the-Middle T1157) attack based on the weakness of TOFU. Secondly, certificates ease the on-boarding of new users, allowing a valid certificate to be used during authentication as long the server trusts the CA and the user is a principal on the certificate. Finally, certificates can expire (fail secure) versus default SSH public keys that do not expire

9 Yubico 2019

(fail open) (Malone 2019). Since key pairs can accumulate access to more systems, the longer they are used, keys used by long-term employees can have quite a bit of access since there is no expiration (Dwyer 2017). The expiration used in certificates can further mitigate key compromise (Unsecured credentials T1552.004) even if an attacker steals the Yubikey and pin. Some companies such as Netflix have new certificates issued every day by user request via a separate SSO authentication (Malone 2019). A major cost of implementing certificates is establishing an automated workflow for issuing certificates and provisioning keys.

Software provisioning and configuration management (CM) systems are another architecture consideration for hardening SSH. Provisioning installs the proper software versions on clients and servers, while configuration management ensures correctly provisioned software. Both are needed to make sure up-to-date software is run with the correct secure configuration options. Old versions of SSH server and client software can have known vulnerabilities. Certain SSH configuration options will offer less protection against attack. While there are many options for provisioning and CM software, this paper will use Ansible for a hardened configuration of OpenSSH. Ansible is a good option because it has no client agents to load and is multi-platform, supporting both Windows (WinRM or SSH) and Linux (SSH). An additional consideration is that Ansible requires an account on each managed system to access via SSH. Because service accounts and their keys can be especially vulnerable to abuse, Ansible accounts will be secured with Yubikey private key storage. Ansible stores configuration tasks in playbooks, which are text files that store all of the provisioning or CM tasks that a control node will remotely run on any managed node. Each time an administrator runs a playbook, Ansible uses SSH to connect to each host and determine if it runs the task. If the task runs, such as if there was a change to a configuration file, Ansible will run the task and report the status back to the control machine. The playbook outputs the status of any changes and the result. Appendix A contains the configurations and playbooks used for this experiment in the GitHub repository link.

2.2.3.2 Configuration

Configuration files for SSH and GPG have many applicable options to the overall security of SSH authentication and authorization. This paper does not provide a comprehensive survey of all of the best security configuration options, but it will present

several settings related to authentication. Additional SSH settings are available in the Github section of Appendix A. Configuration includes known hosts and authorized keys files since they are managed the same way; the files include 'known good' authorized keys and certificates. Administrators should remove anything else. Each organization needs to determine the security trade-offs they are willing to make between security and convenience/compatibility. However, it is especially crucial that once the determinations have been made, the organization reliably enforces their chosen configuration. Several of the important configuration items for SSH are in the Github link of Appendix A.

Administrators must also manage file permissions to improve the security of SSH. Even the best configuration options will be meaningless if they can be easily changed (configuration files) or read (known host, authorized key, and key files) by users or attackers. Part of good configuration management is to enforce least privilege access to configuration and key files. Improper file permissions can lead to key compromise (Unsecured Credentials: Private Keys T1552.004, Remote Services SSH T1021.004) or attackers adding new keys (Application Protocol T1071).

File integrity is closely linked to file permission, but it ensures there have not been any unauthorized changes to the file contents. Even if administrators correctly set file permissions, an attacker could compromise a privileged account and add their public key to the authorized keys file for persistence (Application Protocol T1071). The security team should monitor file integrity, and any changes to configuration files warrant further investigation.

3. Analysis

Knowing a general picture of how SSH works, some of the common attack patterns against it, and possible mitigating controls against those weaknesses; We can test the effectiveness of the proposed solutions to see if there is a real reduction in the vulnerability of SSH. This analysis of SSH authentication hardening will test Debian 10 in both a hardened and baseline (default) configuration against generalized attacks of stealing keys, abusing sockets, adding credentials, and MiTM impersonation. These generalized attacks represent some of the tactics used by attackers in the MITRE ATT&CK matrix in Section 2.2.3. The

tests' results evaluate the effectiveness of the proposed mitigating controls and can propose next steps for further research.

3.1 Test Design

Two different baselines will evaluate the four attack techniques of stealing keys, abusing sockets, adding keys, and impersonation attacks (MiTM):

1. Standard OpenSSH configuration using keys stored on disk
2. Hardened OpenSSH configuration using Yubikey external cryptographic device and configuration managed with Ansible

Multiple different simulations will test the two baselines, generalizing common attack methods against SSH. These simulations will run against Linux (Debian 10). Windows also allows for a similar setup using the Yubikey and OpenSSH as a server or client. Due to complexities in setting up the Windows environment, it was not included in the test. All of the configuration options and tests for this experiment are available on GitHub, as referenced in the Appendix.

TEST DESIGN			
ATTACK CLASS	SIMULATION 1	SIMULATION 2	SIMULATION 3
stealing keys	Access key on disk	Access key in memory	
abusing sockets	Client Compromised – Connect	Client Compromised – Remote	Client Compromised – Jump
adding keys	Add key to user file	Add key to new user	Add attacker cert as trusted
Bad TOFU	MiTM connection to new host	MiTM connection to existing host	

Figure 12: Tested for Baseline-Linux and Hardened-Linux

Accessing keys on disk will be accomplished by copying existing SSH private keys off the target machine remotely. This access could happen through any variety of attacks, including command injection. Once they are off the machine, John The Ripper[10] can crack the key's password. Both the specific compromise leading to copying the key and the actual cracking will not be covered, but are assumed to be trivial once a host is compromised. Accessing the keys in memory involves using a PoC code from NetSPI for Linux[11].

10 SSH key cracking is available in John The Ripper jumbo version

11 Noprop offers a similar tool for Windows keys

Accessing memory assumes root access in both cases. This experiment assumes the initial compromise and privilege escalation.

Abusing sockets will be tested by having a root-level attacker present on several intermediate machines; this attacker will send authentication requests to the SSH-agent to connect to another machine the user has access to. In the first simulation, the attacker will be on the SSH-client computer. Next, the attacker will be on a remote host the user connects to with agent-forwarding enabled. Lastly, the attacker will be on a jump host where the user has agent forwarding enabled.

The adding keys attack will simulate a root-level attacker adding new SSH keys to an existing user and an attacker creating a new account with SSH access to use as a persistent back door. The attacker will attempt to log again to the system using the new keys or the new account. For the hardened baseline test, the re-connection attempt will occur after running Ansible configuration management.

While it is reasonable to expect some security improvements from a hardened configuration, the second part of the test will evaluate if key management and system configuration can be automated using Ansible. In the operational world, users will eventually disregard improvements that require excessive labor, and instead use less secure alternatives. In fast-moving DevOps environments, proper key and certificate management can be a challenge that is 'worked around' (Bocek 2019).

3.2 Test Results

The test results table below summarizes the different trials' outcomes with the base and hardened Linux systems. In every case, the hardened configuration offered improved or similar defenses when compared to the base system.

		Does the ATTACK succeed?	
ATTACK CLASS	SIMULATION	Base system	Hardened System
Stealing keys	Keys read from disk	SUCCESS	FAILED
Stealing keys	Keys read from memory	SUCCESS	FAILED
Abusing Sockets	Client agent	SUCCESS	FAILED
Abusing Sockets	Remote agent	SUCCESS	FAILED
Abusing Sockets	Jump agent	SUCCESS	FAILED
Adding Keys	Existing user	SUCCESS	FAILED
Adding Keys	New user	SUCCESS	FAILED
MiTM	New host	SUCCESS * (TOFU)	FAILED
MiTM	Existing host	FAILED*	FAILED

Figure 13: Test Results - note that 'success' means that the attack succeeded

Detailed test steps are available in the Github section referenced in the Appendix.

3.3 Further Research

Several areas would benefit from more analysis linked with this research. Firstly, the analysis of SSH vulnerabilities was limited to authentication using public keys/certificates and the associated configurations; more analysis of alternative authentication techniques, especially combinations of authentication including keyboard interactive, FIDO, or PAM U2F would be beneficial. Secondly, there is much more room for research of configuration options and SSH security, especially in multi-hop connections with agent forwarding. Thirdly, more research into logging best practices for SSH to ensure user action traceability and the number of SSH users on each system and mapping possible indicators and artifacts to each attack technique would also be useful. Finally, there is room for evaluation of different certificate enrollment and automated user management solutions. Originally, this experiment intended to include OpenSSH for Windows and Linux but did not due to limitations with automating and testing the Windows environment. There continues to be a gap in capabilities between mature Linux OpenSSH and the newer Windows implementation of OpenSSH; Microsoft has stated that this gap will narrow with time. However, inconsistencies or differences between the two operating systems will continue to be of interest. Windows also currently requires special

software to communicate between gpg-agent and OpenSSH (Pye 2020). The uses of WSL and SSH are also potentially interesting topics for further research.

4. Synthesis

Yubikeys and other cryptographic devices can be a great option for securing SSH with MFA, but it is important to communicate the benefits to the system users. Some research has suggested focusing on the personal risk of insecure authentication, identifying a secure alternative, and demonstrating the ease of use is an effective way to change user behavior (Ackerman 2017 p.9). Using configuration management software to enforce the desired state also helps to secure SSH. Ansible is capable of enforcing configuration files, file permissions, and file integrity.

4.1 Recommendations

The tests above show clear advantages in lowering the attack surface SSH presents in an organization. There are several other factors to consider when implementing the solution.

1. Gaining user acceptance for using hardware devices and loss of self-management of SSH keys. This includes an accepted plan for transitioning away from any old key in active use and disabling old keys not actively used.
2. Implementing a key management solution based on company, compliance, or threat modeling requirements (i.e., How often are certificates rotated, is host keys rotation required, how a user is authorized to use SSH on a machine).
3. Determining how to automatically provision new computers to the standard company OpenSSH baseline and how often to enforce existing computer configuration.
4. Implementing SSH logging and monitoring.

4.1 Limitations of Experiment

Readers should consider several limiting factors or assumptions made in this investigation. Firstly, this investigation only looked at OpenSSH, Yubikey, and Ansible; other combinations of SSH software, cryptography device, or configuration management

software may have different results. Secondly, this paper did not cover several classes of vulnerability, including cryptographic attacks, exploiting SSH itself, and attacker C2/exfiltration over SSH. Finally, difficulties with user or organizational acceptance, process implementation, or monitoring were not covered. Organizations need to establish a process for user key provisioning where all keys are tracked and rotated as needed (Kolodgy 2013). While the processes and the protections used in this experiment can mitigate some of the risks many companies face in key management for SSH, it needs to be a part of a broader effort to inventory and track users and their permissions.

5. Conclusion

The Secure Shell (SSH) is an essential part of many networks. It excels in allowing users a relatively secure way to interact with remote resources, traditionally for UNIX systems, but recently for Windows. OpenSSH is one of SSH's most common implementations, and it offers several authentication methods, including passwords, host-based authentication, private keys, and Kerberos. Private key and password authentication are the most common methods, with private key being generally used for most automated SSH tasks common in DevOps. Private key and password authentication are the most common methods, with automated SSH tasks common to DevOps generally using private keys. The ease of use and flexibility of SSH has led to the Lassie-Fare management of keys, leading many organizations not to have an inventory of keys or an understanding of trust relationships in their environment. This key mismanagement is especially a problem when coupled with the known adversary tactics that use or abuse SSH. Many of these attacks can be abstracted to the following categories: stealing SSH keys, abusing SSH sockets, adding attacker keys, and impersonation attacks. This paper presented a potential solution using architecture (external cryptography device, certificates, and configuration management software) and configuration (SSH configuration options, file permissions, and file integrity). This solution was evaluated against a baseline ‘standard’ configuration against the abstracted attack categories for Linux. The test results showed that these changes greatly reduced the attack surface against SSH authentication.

References

Ackerman, P. (2017). *Impediments to Adoption of Two-factor Authentication by Home End-Users*. https://www.giac.org/paper/gsec/35160/impediments-adoption-two-factor-authentication-home-end-users/139464.

Ajaz, Z. (2015, July 29). *Applied Informatics Inc. Blog*. How To Manage SSH Keys Using Ansible. http://blog.appliedinformaticsinc.com/how-to-manage-ssh-keys-using-ansible/

Albrecht, M. R., Degabriele, J., Hansen, T. B., & Paterson, K. G. (2016, October/November). *A Surfeit of SSH Cipher Suites* [Scholarly project]. In *Royal Holloway University London - Information Security Group*. Retrieved August 04, 2020, from https://www.isg.rhul.ac.uk/~kp/surfeit.pdf

Bocek, K. (2019, January 22). Security at DevOps speed: Mind your keys and certificates. https://techbeacon.com/app-dev-testing/security-devops-speed-mind-your-keys-certificates

Burns, K. (2014, July 31). Stealing unencrypted SSH-agent keys from memory. Retrieved July 28, 2020, from https://blog.netspi.com/stealing-unencrypted-ssh-agent-keys-from-memory/

CERT Vulnerability Note VU#958563. (2008, November). Retrieved August 14, 2020, from https://www.kb.cert.org/vuls/id/958563

Cimpanu, C. (2020, February 14). OpenSSH adds support for FIDO/U2F security keys. Retrieved May 14, 2020, from https://www.zdnet.com/article/openssh-adds-support-for-fidou2f-security-keys

Drduh. (2016). *YubiKey-Guide*. GitHub. https://github.com/drduh/YubiKey-Guide

Dwyer, I. (2017, December 5). Moving Security Beyond SSH and PKI. https://devops.com/moving-security-beyond-ssh-pki/

Easton, T. (2018, October 22). *Chalubo botnet wants to DDoS from your server or IoT device*. Sophos News. https://news.sophos.com/en-us/2018/10/22/chalubo-botnet-wants-to-ddos-from-your-server-or-iot-device/

Flathers, R. 'ropnop'. (2018, May 20). Extracting SSH Private Keys From Windows 10 ssh-agent. Retrieved July 28, 2020, from https://blog.ropnop.com/extracting-ssh-private-keys-from-windows-10-ssh-agent/

Filkins, B. (2015, December). Securing SSH Itself with the Critical Security Controls - SANS Institute. Retrieved May 14, 2020, from https://www.sans.org/reading-room/whitepapers/analyst/securing-ssh-cis-critical-security-controls-36462

Fong-Jones, L. (2019, July 8). Hardening SSH with 2fa. Retrieved May 15, 2020, from https://gist.github.com/lizthegrey/9c21673f33186a9cc775464afbdce820

Kolodgy, C. J. (2013, April). A G a p i n g H o l e i n Y o u r I d e n t i t y a n d A c c e s s M a n a g e m e n t S t r a t e g y : S e c u r e S h e l l A c c e s s C o n t r o l s. IDC. https://info.ssh.com/a-gaping-hole-in-your-identity-and-access-management-strategy-secure-shell-access-controls

Lodge, D. (2016, August 31). *How to abuse SSH keys*. How to Abuse SSH keys. https://www.pentestpartners.com/security-blog/how-to-abuse-ssh-keys/

Malone, M. (2020, June 18). *If you're not using SSH certificates you're doing SSH wrong*. If you're not using SSH certificates you're doing SSH wrong. https://smallstep.com/blog/use-ssh-certificates/

Matrix.org. (2019, April 11). We have discovered and addressed a security breach. (Updated 2019-04-12). Retrieved from https://matrix.org/blog/2019/04/11/we-have-discovered-and-addressed-a-security-breach-updated-2019-04-12

Microsoft. (2019). *Overview about OpenSSH for Windows*. Overview about OpenSSH for Windows | Microsoft Docs. https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview

Miller, M. (2019, December 9). DevOps Security Best Practices. https://www.beyondtrust.com/blog/entry/devops-security-best-practices

MITRE ATT&CK. (2017, May 31). Retrieved from https://attack.mitre.org/techniques/T1184, https://attack.mitre.org/techniques/T1145, https://attack.mitre.org/techniques/T1071, https://attack.mitre.org/techniques/T1021, https://attack.mitre.org/software/S0220/

Moyle, E. (2017). *SSH: Practitioner Considerations*. ISACA. https://www.isaca.org/bookstore/bookstore-wht_papers-digital/whpssh

National Institute of Standards and Technology, Burr, W. E., Perlner, R. A., Grassi, P. A., Newton, E. M., Fenton, J. L., … Richer, J. P., NIST SP 800-63B: Digital Identity Guidelines (2017). National Institute of Standards and Technology. https://pages.nist.gov/800-63-3/sp800-63b.html#sec8

National Institute of Standards and Technology, Ylonen, T., Turner, P., Scarfone, K., & Souppaya, M.(2015, October),NIST IR7966: Security of Interactive and Automated Access Management Using Secure Shell (SSH). Retrieved from http://dx.doi.org/10.6028/NIST.IR.7966

Oswald, D., Richter, B., & Paar, C. (2014, February). *Side-Channel Attacks on the Yubikey 2 One-Time Password Generator* (Rep.). Retrieved July 15, 2020, from Horst Görtz Institute for IT Security website: https://www.emsec.ruhr-uni-bochum.de/media/crypto/veroeffentlichungen/2014/02/04/paper_yubikey_sca.pdf

Pye, B. (2020, March). *benpye/wsl-ssh-pageant*. GitHub. https://github.com/benpye/wsl-ssh-pageant

Shukhman, P. (2019, August). *Securing SSH Access*. *OWASP Ottowa meetup*. Ottowa, Canada.

Shukhman, P. (2020, February 13). *YubiKey for SSH on Windows: Complete Walkthrough*. Work & Life Notes. https://worklifenotes.com/2019/07/05/yubikey-for-ssh-on-windows-complete-walkthrough/

Webb, G. (2016, February). Infographic: Crumbling Cybersecurity-CIOs Are Wasting Millions. Retrieved July 16, 2020, from https://www.venafi.com/blog/infographic-crumbling-cybersecurity-cios-are-wasting-millions

Yubico. (2019). Product Briefs. Retrieved September 27, 2020, from https://www.yubico.com/resources/product-briefs/

Appendix A

Please see the Github page for all Appendix information.
https://github.com/Dou10s/Hardened_SSH

Scott Ross

Configuration for hardened SSH and SSHD
https://github.com/Dou10s/Hardened_SSH/tree/master/KeyServer/provisioning/hardenedconfig

Ansible playbooks
https://github.com/Dou10s/Hardened_SSH/blob/master/KeyServer/provisioning/enforce-config.yml
https://github.com/Dou10s/Hardened_SSH/blob/master/KeyServer/provisioning/provision-keys.yml

Detailed test steps
https://github.com/Dou10s/Hardened_SSH/tree/master/how-to/tests

Scott Ross

Appendix B

TACTIC	TECHNIQUE	CODE	DESCRIPTION	ARCHITECTURE	CONFIGURATION
Initial Access	Valid Accounts	T1078	Use a valid account for SSH	ECD, CERT	
Initial Access	Exploit Public-Facing Application	T1190	Exploiting SSH service (out of scope)		
Execution					
Persistence	Account Manipulation	T1098.004	Add attacker cert/keys as trusted	PROV	PERM, INTG
Persistence	Create Account	T1136	Add attacker account	PROV	INTG
Persistence	Valid Accounts	T1078	Use existing account for SSH	ECD, CERT	
Privilege Escalation	Valid Accounts	T1078	Use existing privileged account	PROV	CFO
Privilege Escalation	Exploitation for privilege escalation	T1068	Exploiting SSH service (out of scope)		
Defense Evasion	File and Directory Permissions Modification	T1222	Attacker can modify SSH config files	PROV	PERM, INTG
Defense Evasion	Valid Accounts	T1078	Existing account use harder to detect	ECD, CERT	
Credential Access	Brute Force	T1110	Brute force password ssh / key passwords	ECD, CERT	CFO
Credential Access	Unsecured Credentials: Private Keys	T1552.004	Attackers steal private keys on a system, can be on disk (simple) or in memory (more advanced)	ECD, CERT	PERM
Credential Access	Input Capture: keylogging	T1056.001	Attackers capture private key password/ pin	ECD, CERT	
Credential Access	Man-in-the-Middle	T1557	attacker disguises as server to client and proxies connection to server to steals information	CERT	CFO
Credential Access	Network Sniffing	T1040	Attacker can record and crack weak SSH implementations/cryptography		
Discovery	Remote System Discovery	T1018	Attacker looks through shell history, known hosts file to find new targets		CFO
Discovery	Network Sniffing	T1040	Attacker can record and crack weak SSH implementations/cryptography		
Lateral Movement	Remote Service Session Hijacking	T1563.001	Attacker abuses existing socket for authentication	ECD	CFO
Lateral Movement	Remote Services	T1021.004	attacker steals ssh credentials	ECD, CERT	CFO, PERM
Lateral Movement	Exploitation of remote Services	T1210	Exploiting SSH service (out of scope)		
Collection					
Command and Control	Application Layer Protocol	T1071	Attacker can run C2 over SSH		
Command and Control	Protocol Tunneling	T1572	Attacker can tunnel C2 in SSH		
Exfiltration	Exfiltration over Alternative Protocol	T1048	Attacker can exfiltrate data using SSH		
Exfiltration	Exfiltration over C2 Channel	T1041	Attacker can exfiltrate data using SSH		
Impact					

NOT APPLICABLE	ARCHITECTURE		CONFIGURATION	
	External Cryptograhic device (Yubikey)	ECD	Configuration file options	CFO
OUT OF SCOPE	Certificates	CERT	File Permissions	PERM
	Provisioning Software (Ansible)	PROV	File Integrity	INTG

Figure 14: MITRE ATT&CK mapping for SSH

Scott Ross

Last Updated: November 28th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Essentials Australia 2021	**Melbourne, AU**	**Feb 15, 2021 - Feb 20, 2021**	**Live Event**
SANS OnDemand	**OnlineUS**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**