Securing SSH in Commercial Environments

by Robert Moller

Version	Date	Information
1.0	27 July 2021	First release

Table of Contents

Introduction

Installation and installation examples were made on Rocky Linux 8.4 (Green Obsidian), a binary compatible release of Red Hat Enterprise Linux (RHEL) operating system.

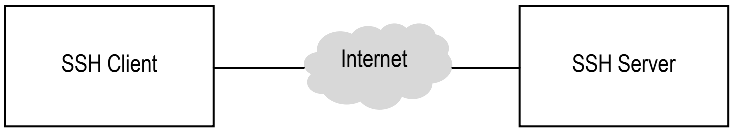
SSH Connection Methods

Connections to SSH servers can be performed in a number of ways, it can be direct to the public IP address of the SSH server, or via a firewall which directs the packets to the relevant server. In more robust environments the server is not directly exposed to the internet but is accessible via a proxy agent which directs packets to the relevant server. The later also has the advantage of allowing a level of redundancy known in the industry jargon as high availability.

Below is a list of connection methods outlining the benefits and limitations of each one.

Direct Connection

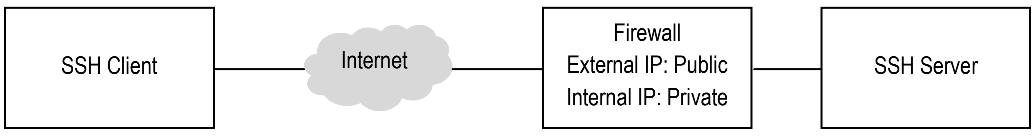
Direct connect involves connecting directly to the SSH server. The connection can be IP or domain name based. Security is provided by the servers SSH configuration and server firewall hardening methods.



Advantages	Disadvantages
• No firewall to configure • Easy to setup	• Public IP of server exposed to internet providing multiple attack vectors. • Robust security needed via SSH service hardening in sshd_config and PAM modules. • Server firewall (iptables) is first point of contact and needs to be well configured to prevent different attack vectors.

Firewalled Connection

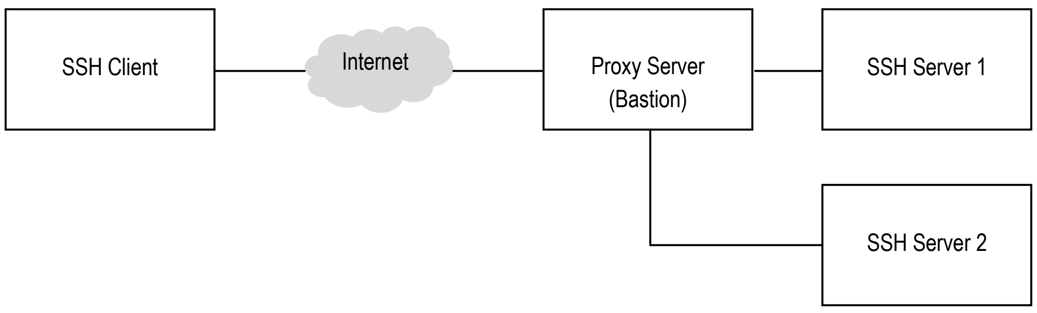
Firewalled connection involves placing a firewall between the SSH client and server. This method provides a degree of protection as the public IP of the SSH server is not exposed (where the connection to the server uses NAT). This set up involves the additional cost of a hardware or software firewall appliance or service.



Advantages	Disadvantages
• IP address of SSH server not exposed to internet • Only SSH port forwarded to server reducing attack vectors • Additional configuration on firewall can limit traffic based on source IP address (note that source address filtering can also be done on the server firewall as well)	• Harder to reconfigure – addition of NAT and port forwarding rules • Additional cost of a firewall appliance

Proxy Connection

In a proxy type connection a proxy server is placed between SSH client and server. The proxy server acts as a bastion host, that is, a public IP facing hardened server which acts a gateway for communication to other servers. The advantage of such setup is that only this server is exposed to the internet. As proxy servers can also load balance traffic can be spread over multiple bastion servers, this also provides redundancy; as this allows a server to be taken offline for maintenance. Data is relayed directly the relevant SSH servers using SSH tunnelling.



Advantages	Disadvantages
• Exposes only one server to the internet reducing attack scope. • Proxy may also provide high-availability via automatic failover to couple proxy servers. • Proxy server can provide access to several bastion servers (SSH gateways), this achieves load balancing and high-availability as bastion servers can be placed offline for maintenance.	• Extra cost for proxy server • More complicated configuration

Direct Connection to Server via Proxy Jump Command

This uses the –J flag in the ssh connect command. The format of this command is:

```
$ ssh –J <bastion_hostname> <proxy hostname>
```

If the user name of the bastion host and/or port number is different these values can be supplied as well

```
$ ssh –J <user1>@<bastion_hostname:bastion_port> <user2>@<ssh_hostname>:ssh_port>
```

Direct Connection to Server via Forwarding StdIn StdOut

This uses the –o ProxyCommand option. The –W flag instructs ssh to forward standard in and out traffic to the SSH server hostname and port. The format of this command is:

```
$ ssh –o ProxyCommand = “ssh –W <ssh_hostname>:<ssh_port> <bastion_host>” <bastion_host>
```

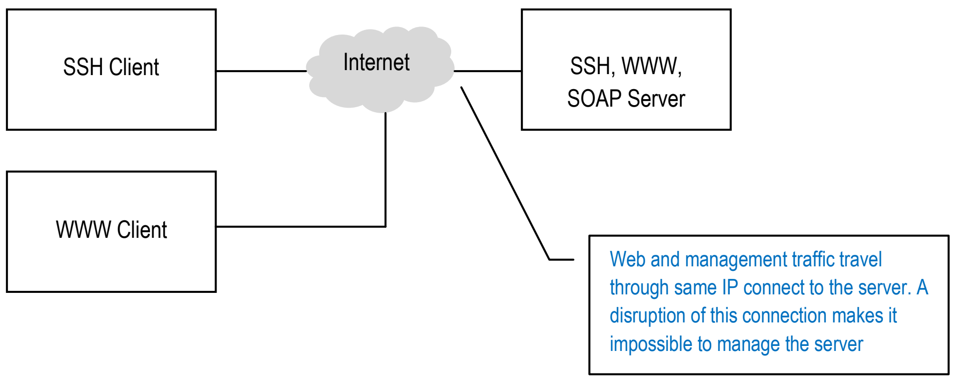
In-Band and Out-of-Band Connections

SSH traffic can travel through the same IP address as user data, such as web traffic, this is known as in-band management system. Out-of-band connection occurs when SSH and other management protocols travel through a separate IP network, involving separate connections.

From a security perspective it is better to have SSH data travel through its own network and switches (or on separate VLANS). The benefit in this approach is a reduced attack vectors against SSH servers. It is recommended more than one of out-of-band management channel be available and at least one involving cellular network, ie. 4G LTE.

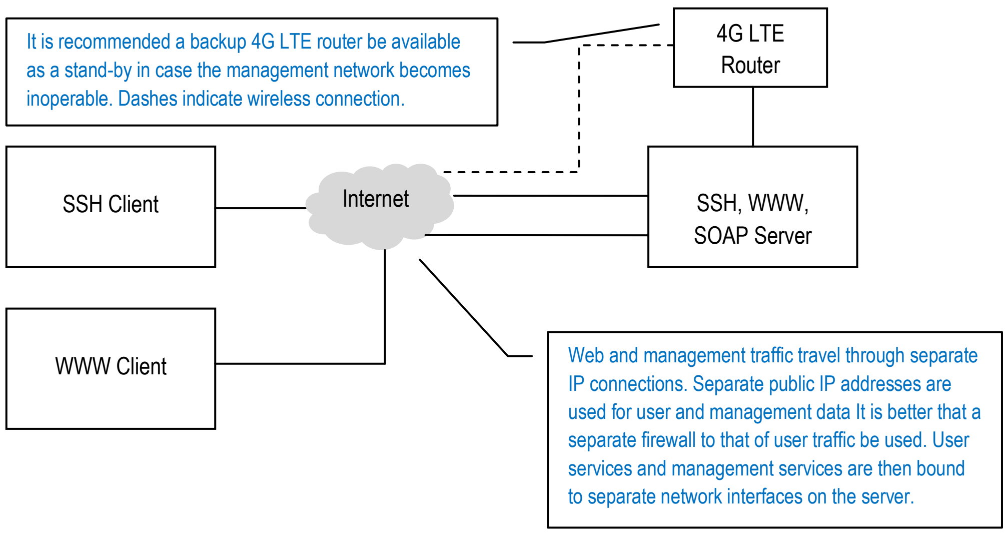
In-Band Management

In this management style all user and management traffic travel to the exposed port of the SSH server. This server may host multiple services, ie. http, soap, dns, etc. through the same IP port. This greatly increases attack vectors. Firewall configuration needs to be resilient and weaknesses in the web server may allow inserting backdoors for SSH protocol.


SSH Client
Internet
SSH, WWW, SOAP Server
WWW Client
Web and management traffic travel through same IP connect to the server. A disruption of this connection makes it impossible to manage the server

Out-of-Band Management

Out-of-band management places management data in separate networks to the server compared to user data. Management staff won't be locked out of servers in case of cyber attacks. If available a separate cellular (4G LTE) network should also be installed in case the IP network near the server becomes inoperable. The wireless network becomes the last resort method for accessing servers via SSH.


It is recommended a backup 4G LTE router be available as a stand-by in case the management network becomes inoperable. Dashes indicate wireless connection.
4G LTE Router
SSH Client
Internet
SSH, WWW, SOAP Server
WWW Client
Web and management traffic travel through separate IP connections. Separate public IP addresses are used for user and management data It is better that a separate firewall to that of user traffic be used. User services and management services are then bound to separate network interfaces on the server.

4GE LTE Connectivity

Connecting via 4G LTE requires a 4GE LTE capable modem with a publicly addressable IP address. Most cell based networks use Carrier-Grade NAT (CG-NAT) at least for IPv4. These networks use middle box network translators to translate private IP addresses into public, this function is similar to that of NAT. CG-NAT IP addresses are not reachable for services without the addition of a NAT-to-NAT third party data transfer service.

A public IP address can be obtained from most telecommunications companies. In Australia, Optus provides an easy method to obtain a public IP address capable SIM card at a low additional cost to that using CG-NAT. Customers applying for this type of SIM need an Australian Business Number (ABN). For more information see Optus Mobile Business Broadband (Contact 133 343).

Using SSH with HAProxy

HAProxy is free, open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications. It has the reputation for being fast and efficient in terms of memory usage. It is available under the GNU General Public License Version 2 distribution and modification rights.

The configuration below is a typical scenario where there is 1 HAProxy service connected to 3 SSH Servers. Connections are made tunnelled using SSH tunnelling method, ProxyCommand.

The installation will first use the Apache web server for testing, later on SSH will be enabled.

Installation and Http Proxy Configuration

1. Update server to latest packages

```
$ sudo dnf update -y
```

2. Install Apache web server on the SSH servers

```
$ sudo dnf install httpd -y
```

3. Enable and start web server on the SSH servers

```
$ sudo systemctl enable httpd
$ sudo systemctl start httpd
```

4. Install telnet client

```
$ sudo dnf install telnet -y
```

5. Check web server is running by telnetting into it. If working the message 'Connected to localhost' should appear.

```
$ telnet localhost 80
```

6. Install HAProxy

   Check if HAproxy is available

```
$ sudo dnf info haproxy
```

   Install HAProxy

```
$ sudo dnf install haproxy -y
```

7. Enable and start HAProxy

```
$ sudo systemctl enable haproxy
$ sudo systemctl start haproxy
```

8. Check HAProxy is running

```
$ systemctl status haproxy
```

9. Do a HAProxy test configuration to load balance web server.

   Copy current configuration file to backup

```
$ cd /etc/haproxy
$ sudo cp haproxy.cfg haproxy.cfg.bak
```

Edit the haproxy.cfg file.

```
$ sudo nano haproxy.cfg
```

Modify and add new entries and enter below

```
frontend http_front
    bind *:80
    stats uri /stats
    default_backend http_back

backend http_back
    balance roundrobin
    server ssh1 192.168.108.10:80 check
```

10. Test configuration changes. If configuration is valid the message 'Configuration file is valid appears'

```
$ haproxy –c –f /etc/haproxy/haproxy.cfg
```

11. Restart HAProxy to load new configuration

```
$ sudo systemctl restart haproxy
```

HAProxy SSH Configuration – SSH Tunnelling

This configuration uses the Jump or ProxyCommand. It is harder to configure but offers the benefit of only exposing 1 or 2 ports to the Internet.

1. Do a HAProxy SSH configuration to load balancer web server.

   Copy current configuration file to backup

```
$ cd /etc/haproxy

$ sudo cp haproxy.cfg haproxy.cfg.bak
```

Edit the haproxy.cfg file.

```
$ sudo nano haproxy.cfg
```

Modify and add new entries and enter below. ssl_fc_sni variable is used to select the backend. It is substituted with the server wishing to be connected as part of the ProxyCommand option. SSH is set to port 2222 on load balancer. On SSH server it is set 31605.

```
frontend ssh_front
    bind *:2222
    mode tcp
   log-format "%ci:%cp [%t] %ft %b/%s %Tw/%Tc/%Tt %B %ts %ac/%fc/%bc/%sc/%rc %sq/%bq
dst:%[var(sess.dst)] "
    tcp-request content set-var(sess.dst) ssl_fc_sni use_backend %[ssl_fc_sni]
    use_backend %[ssl_fc_sni]

backend ssh_server1
    mode tcp
    server ssh1 192.168.108.80:22 check

backend ssh_server2
```

```
    mode tcp
    server ssh2 192.168.108.81:22 check
```

2. Test configuration changes. If configuration is valid the message 'Configuration file is valid appears'

```
$ haproxy –c –f /etc/haproxy/haproxy.cfg
```

3. Restart HAProxy to load new configuration

```
$ sudo systemctl restart haproxy
```

4. Connect from another host using the ProxyCommand option

```
$ ssh -o ProxyCommand="openssl s_client –quiet –connect 192.168.16.97:2222 –servername
ssh_server1 –p31605" dummyName1
```

Creating x509 Certificate

```
$ sudo openssl req –x509 –sha256 –nodes –newkey rsa:4096 –subj "/C=AU/CN=localhost" –keyout
private.pem –out cert.pem
```

HAProxy SSH Configuration – SSH Proxying

This configuration uses the standard proxy configuration as is done for http traffic. An external port is required for each ssh service as there is no tunnelling. This method exposes more external ports for attack.

1. Do a HAProxy SSH configuration to load balancer web server.

   Copy current configuration file to backup

   ```
   $ cd /etc/haproxy

   $ sudo cp haproxy.cfg haproxy.cfg.bak
   ```

   Edit the haproxy.cfg file.

   ```
   $ sudo nano haproxy.cfg

   frontend ssh_front
       bind *:2222
       mode tcp
       timeout client 60s
      log-format "%ci:%cp [%t] %ft %b/%s %Tw/%Tc/%Tt %B %ts %ac/%fc/%bc/%sc/%rc %sq/%bq
   dst:%[var(sess.dst)] "
       use_backend ssh_back

   backend ssh_back
       mode tcp
       timeout server 60s
       timeout connect 60s
       server ssh1 192.168.108.10:22 check
   ```

2. Test configuration changes. If configuration is valid the message ‘Configuration file is valid appears’

   ```
   $ haproxy -c -f /etc/haproxy/haproxy.cfg
   ```

3. Restart HAProxy to load new configuration

   ```
   $ sudo systemctl restart haproxy
   ```

4. Connect to the public facing host using the proxy port

   ```
   $ ssh user@192.168.16.97 -p 2222
   ```

Troubleshooting HAProxy Port Binding Issues

A common reason for HAProxy not starting is the port already is used by another process. The sample case below shows identifying the process owner and stopping it before reattempting to start HAProxy. We identifying who owns the process for port 80

1. Restart HAProxy

   ```
   $ sudo systemctl restart haproxy
   ```

   An error is returned and asking to check 'systemctl status haproxy.service' and 'journalctl –xe'

2. Verify journalctl –xe for any information

   ```
   $ journalctl -xe
   ```

3. Check if port 80 is bound to an IPv4 interface

   ```
   $ sudo ss -4 –tlnp | grep 80
   ```

4. Check if port 80 is bound to an IPv6 interface

   ```
   $ sudo ss -6 –tlnp | grep 80
   ```

   If the port is bound an output identifying the owner process is returned. The process number is identified in the 'pid' parameter.

   ```
   LISTEN 0      128                *:80               *:*
   users:(("httpd",pid=1146,fd=4),("httpd",pid=1145,fd=4),("httpd",pid=1143,fd=4),("httpd",pid=959,f
   d=4))
   ```

5. Check owner of process 959 (pid=959)

   ```
   $ sudo ps –p 959
   ```

   Output of owner process appears to be the httpd process

   ```
      PID TTY          TIME CMD
       959 ?        00:00:00 httpd
   ```

6. Checking the status of the web server we see that in fact is running

   ```
   $ systemctl status httpd
   ```

7. Stop and disable the web server

   ```
   $ sudo systemctl stop httpd
   $ sudo systemctl disable httpd
   ```

8. Try and restart HAProxy service

   ```
   $ sudo systemctl restart haproxy
   ```

9. Connect via telnet to HAproxy server, the web page should be displayed

   ```
   $ telnet localhost 80
   ```

   If connected to host type in uppercase and press <Enter> key twice

   ```
   GET / HTTP/1.1
   HOST: HOSTNAME
   ```

10. An alternate way to verify web site is reachable is to use curl.

    ```
    $ curl localhost
    ```

    Output should display default web site running on the SSH server

    ```
    <!doctype html>
    <html>
      <head>
        <meta charset='utf-8'>
        <meta name='viewport' content='width=device-width, initial-scale=1'>
        <title>HTTP Server Test Page powered by: Rocky Linux</title>
        <style type="text/css">
    (...)
    ```

Checking SSH Service and Port Status

1. Check SSH is running

```
    $ ps aux | grep sshd
```

Results...

```
    root         928  0.0  0.1  92320  6960 ?        Ss   07:27   0:00 /usr/sbin/sshd
```

Alternatively, the RedHat sytle 'service' and 'systemctl' command can be used

```
    $ service sshd status
```

or

```
    $ systemctl status sshd
```

Results...

```
    [user@localhost ~]$ systemctl status sshd
    ● sshd.service - OpenSSH server daemon
       Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
       Active: active (running) since Mon 2021-07-12 07:27:02 EDT; 7min ago
         Docs: man:sshd(8)
               man:sshd_config(5)
     Main PID: 928 (sshd)
        Tasks: 1 (limit: 23551)
       Memory: 2.2M
       CGroup: /system.slice/sshd.service
               └─928 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-
    poly1305@openssh.com,aes256-ctr,>

    Jul 12 07:27:01 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...
    Jul 12 07:27:02 localhost.localdomain sshd[928]: Server listening on 0.0.0.0 port 22.
    Jul 12 07:27:02 localhost.localdomain sshd[928]: Server listening on :: port 22.
    Jul 12 07:27:02 localhost.localdomain systemd[1]: Started OpenSSH server daemon.
```

2 . Check port sshd is running

```
    $ netstat -plant | grep :22
```

Result...

```
[user@localhost ~]$ netstat -plant | grep :22
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

```
3. Check if can initiate connection using telnet (in Putty) externally

4. Check can connect via loopback (to itself)

Format: ssh <user>@<host>

$ ssh user@127.0.0.1

5. Check can connect outside (port forwarding required if VM running in NAT)

6. Check firewall status (not iptables). If firewall running it may be blocking ssh connection

    $ systemctl status firewalld
```

```
If running result would be

  firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2021-07-12 07:27:01 EDT; 44min ago
     Docs: man:firewalld(1)
 Main PID: 867 (firewalld)
    Tasks: 2 (limit: 23551)
   Memory: 31.4M
   CGroup: /system.slice/firewalld.service
           └─867 /usr/libexec/platform-python -s /usr/sbin/firewalld --nofork --nopid
```

To stop and disable firewalld type

```
    $ systemctl stop firewalld
    $ systemctl disable firewalld
```

```
Check again if running

    $ systemctl status firewalld

    [user@localhost ~]$ systemctl status firewalld
       firewalld.service - firewalld - dynamic firewall daemon
       Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor preset: enabled)
       Active: inactive (dead)
         Docs: man:firewalld(1)
```

6. Check if SSH service is running on port 22 (or desired port)

```
$ netstat –plant | grep :22
```

SSH Authentication Methods

SSH connections can be authorised using password or cryptographic keys. A password may also be used with the cryptographic key method of authentication though in practice this is seldom done.

Below is a breakdown of strengths and weaknesses using each type of authentication method

Password Authentication

Advantages	Disadvantages
• Relatively easy to set up • No key files necessary	• Subject to brute force attacks. Keys need high level of entropy (not vulnerable to dictionary attacks) • Temptation to create easy to remember passwords which makes logins vulnerable to attack • Username and passwords need to be directly transmitted to server making it prone to hacking. • Hackers can clone popular sites to trick users to giving their usernames and passwords. • Recommend use of long passphrases as passwords, Example: bluegums34makesfor#funkyhaze$

Recommendations for Password Authentication

1 – Create login banner. The banner should identify the company and provide warning on unauthorized access.

```
$ cd /etc/ssh
$ sudo nano login_banner
```

2 – Create a Message of the Day (motd) Bash file. This is better than the Printmotd as it allows Bash commands to be executed where Printmotd does not

```
$ cd /etc/ssh
$ sudo nano motd.sh
```

Add sample content displaying hostname and system up time

```
#! /usr/bin/env bash
HOSTNAME=`uname -n`
SYSTEMUP=`uptime -p`
echo "
Hostname: $HOSTNAME - Uptime: $SYSTEMUP
"
```

Set execute rights

```
chmod +x motd.sh
```

Copy the motd file to profile folder so it can be executed at login

```
$ sudo cp motd.sh /etc/profile.d/motd.sh
```

3 – Change ssh port. SSH uses TCP port 22 by default Many Linux distros use ports in the 32768-60999 range for dynamic assignment, these should not be used

Check if port is available – no response is returned if port not used. Example below uses port 31605

```
$ sudo lsof -i:31605
```

4- Add delay via the Pluggable Authentication Modules (PAM) shared object pam_faildelay.so. This module allows configuration of delay time between each connection attempts. This has the benefit of slowing down brute force attempts. The example below will add a 10 second delay between each login attempt.

Check pam_faildelay.so shared object is installed

```
$ sudo find / -name pam_faildelay.so
```

If not found it can be installed using dnf (dandified yum) package manager.
Check which package contains the search file

```
$ dnf provides '*pam_faildelay.so'
```

Once we found the package we can install

```
$ sudo dnf install pam
```

SSH type login delay are configuredin the sshd file located at /etc/pam.d

```
$ cd /etc/pam.d
$ sudo nano sshd
```

Add the following line to add login delay of 10 seconds. The delay parameter is entered in microseconds.
Sample values and second equivalents: 1s = 1000000us, 10s = 10000000us, 60s = 60000000us

auth include pam_faildelay.so delay=10000000

For a 10 second delay the file should look similar to...

```
#%PAM-1.0
auth       substack     password-auth
auth       include      postlogin
auth       optional     pam_faildelay.so     delay=10000000
account    required     pam_sepermit.so
account    required     pam_nologin.so
account    include      password-auth
password   include      password-auth
# pam_selinux.so close should be the first session rule
session    required     pam_selinux.so close
session    required     pam_loginuid.so
# pam_selinux.so open should only be followed by sessions to be executed in the$
session    required     pam_selinux.so open env_params
session    required     pam_namespace.so
session    optional     pam_keyinit.so force revoke
session    optional     pam_motd.so
session    include      password-auth
session    include      postlogin
```

4 – Edit sshd_config file

```
$ sudo nano /etc/ssh/sshd_config
```

Disable SSH protocol version 1

```
Protocol 2
```

Delay compression until after authentication

```
Compression delayed
```

Banner – Add warning banner. This may be used as deterrent for unauthorised logins.

```
Banner /etc/ssh/login_banner
```

Motd – Set PrintMotd to off – this is handled by a Bash script

```
PrintMotd no
```

Port Number – Change port number from 22 to desired port from step 1

```
Port 31605
```

Allowed Users – Restrict which users can login via ssh. Users login names separate by space.

```
AllowUsers John Mary Smith
```

Root Login – Disable root login

```
PermitRootLogin no
```

Authentication Methods – Edit/uncomment the following lines to allow for host-based authentication.

```
PasswordAuthentication yes
RSAAuthentication no
PubkeyAuthentication no
```

Empty passwords – Disabling empty passwords will ensure an attacker cannot access SSH on a misconfigured user account.

```
PermitEmptyPasswords no
```

Login Times – Minimises risk of successful brute force attacks. Intruder only has 60 seconds to attempt logins. It also limits the number of concurrent unauthenticated connections

```
LoginGraceTime 60  (or LoginGraceTIme 1m)
```

Login Attempts – Slow down bots trying to brute force, default is 6.

```
MaxAuthTries 3
```

Idle logout – Set idle logout to avoid unattended SSH session. The command below will exit client if they haven't entered a command after 5 minutes

```
ClientAliveInterval 300
ClientAliveCountMax 0
```

Maximum startups – This parameter configures the ability of the SSH server to handle unauthorized access at a given instance. Reducing this value will hinder coordinated attacks from different clients.

Parameters are in the form

MaxStartups <Param1>:<Param2>:<Param3>

Param1 - Number of unauthenticated connections before server starts dropping connections
Param2 – Percentage chance of dropping connections once server reaches value in Param1
Param3 – Maximum number of connections before server starts dropping everything

```
MaxStartups 4:10:8
```

Strict Mode – Enforces checks on privilege and ownership in the users' home direction. If these cheks fail the user is not allowed to connect via SSH

```
StrictModes yes
```

.rhosts file – SSH can be configured to emulate the behaviour of the obsolete rsh command used in .rhosts which is historically unsafe

```
IgnoreRhosts yes
```

Gateway ports – SSH binds local port forwarding to the loopback address only. GatewayPorts is used to specify if this is the expected behaviour.

```
GatewayPorts no
```

X11 Forwarding – Disable X11 GUI support if not needed – is used when forwarding GUI interface via ssh

```
X11Forwarding no
```

~~Roaming – Roaming was an experimental and undocumented feature where clients could disconnect from one location and reconnect from another.~~

~~Host *~~
~~UseRoaming no~~

Known Host Hashing – If client machine is compromised, the amount of useful information obtained can be minimised by obfuscating the known_hosts file. This is option is only relevant to public-key type SSH authentication. This file holds public key data.

```
HashKnownHosts yes
```

SSH Forwarding – Prevent SSH clients from creating VPN tunnels via the SSH connection

```
AllowTCPForwarding no
```

5 – SELinux does not allow sshd to run on another port on a default install. We need to Inform SELinux about change of default port from 22 to 31605.

```
$ sudo semanage port -a -t ssh_port_t -p tcp 31605
```

6 – Restart ssh service

```
$ sudo systemctl restart sshd
```

7 – Check sshd service is running

```
$ sudo systemctl status sshd
```

Output example...

```
[user@localhost ~]$ systemctl status sshd
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-07-13 08:19:51 EDT; 5s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 9973 (sshd)
    Tasks: 1 (limit: 23551)
   Memory: 1.1M
   CGroup: /system.slice/sshd.service
```

```
            └─9973 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-
poly1305@openssh.com,aes256-ctr
```

8 – Check sshd service using new port

```
$ sudo netstat -plant | grep :31605
```

Output example...

```
tcp        0      0 0.0.0.0:31605           0.0.0.0:*               LISTEN      9973/sshd
tcp6       0      0 :::31605                :::*                    LISTEN      9973/sshd
```

9 – Connect from other host

The –vvv flag shows debugging information in case connection fails.

```
$ ssh -vvv testuser@192.168.1.1 -p 31605
```

Public Key Authentication

Advantages	Disadvantages
• Keys can be 4096 bits in length or more, making it difficult to brute-force attack. • SSH keys are not human generated so not subject to human weakness of easy to guess values. • Connection can only be initiated from machine hosting the private key. • A password can be added to SSH key authentication (multi-factor) increasing security	• Extra work required to set up keys. • Private key needs to be stored on client machine, these keys may be stolen. • Longer key sizes impose encryption penalty on devices. This increases per user connected. • Burden of copying keys to new client devices if connected from different machines.

Ciphers Used in Public Key Communication

DSA - Digital Signature Algorthim – Developed by National Security Agency (DSA). Discovered to contain a subliminal channel that allows an implementor to leak information, ie. secret bits, with every signature. Some versions contain security flaws. **Not recommended.**

```
$ ssh-keygen –t dsa
```

RSA - Rivest-Shamir-Adleman – Most widely used asymmetric ciphers. Draws its security from the difficulty in factoring large integers that are a product of two large primes of similar size. Maybe susceptible to attack if bad number generators used. This can be mitigated by using the correct options during key generators. **Recommended.**

```
$ ssh-keygen –t rsa –b 4096
```

Types of SHA (hash) can be directly selected in via –t parameter

```
$ ssh-keygen –t ssh-rsa –b 4096          (Not recommended)
$ ssh-keygen –t rsa-sha2-256 –b 4096
$ ssh-keygen –t rsa-sha2-512 –b 4096
```

Passphrases should be at least 20 characters. The –a option specifies the number of key derivation function (KDF) rounds to be used when check the passphrase. A high number will slow down key verification and frustrate brute-force or dictionary type attacks on a server.

```
$ ssh-keygen –t rsa-sha2-512 –b 4096 –a 200 –N “greendogseatmoreducks43than&redemus”
```

Key length have gradually increased due to more powerful processors available.

Key Length	Symmetric Equivalent	Secure Until	Information
1024	80	2018	Insecure. Do not use.
2048	112	2030	Recommended by NIST. Acceptable usage up to 2030.
3072	128	2030+	Acceptable usage beyond 2030.
4096	140	2030+	Used by GitHub. Acceptable usage beyond 2030.
7680	192	2030+	Acceptable usage beyond 2030.
15360	256	2030+	Acceptable usage beyond 2030.

Note: Doubling the key length, ie. 2048 to 4096, increases speed time to verify a message by 7x, and time to verify a signature by 3x. This could become significant on VPS servers running many SSH connected clients. Ed25519 is a newer cipher which places fewer burdens on CPU for equivalent security.

ECSDA – Elliptic Curve Digital Signature Algorithm – This algorithm is a variant of DSA which uses elliptic curve cryptography. Like DSA, it was purported to have a kleptographic backdoor advantageous to those who know about, such as the NSA. This backdoor may be in relation to the Dual_EC_DRBG (Dual Elliptic Curve Deterministic Random Bit Generator) design. The backdoor mechanism has been published by Dan Shumow and Niels Ferguson. **Not recommended.**

```
$ ssh-keygen –t ecdsa –b 521
```

Ed25519 – EdDSA signature scheme using SHA-512 and Curve25519 elliptic curve offering 128 bits of security. Introduced in OpenSSH 6.5. It offers the advantage of being compact, ie. only 68 characters compared to 544 characters (RSA 3072). It offers a rast signing process. The EdDSA signature scheme is vulnerable to single fault attacks. This was demonstrated by Kudelski Security Research running an attack on an Arduino Nano micro PC using the Ed25519 scheme. Voltage glitches were introduced to cause single byte random errors and the end of the computation. That allowed testers to brute force the error location and value and recover half of the secret key. **Recommended.**

```
$ ssh-keygen –t ed25519
```

Verifying SSH Public Key Algorithms

Password – Connections made using a password

Pro	Easy to implement Does not require private/public key files
Con	Weakest method

Public Key

Algorithms

DSA	Unsafe and not recommended
RSA	Safe if key lengths 4096 or greater used
ECDSA	Possible weakness in the Dual_EC_DRBG algorithm
Ed25519	Better security and faster performance than DSA or ECDSA

```
           Only supported on OpenSSH 6.5 and above
```

Appendix A – Setting Hostnames in Test Environment

```
$ sudo hostnamectl --transient set-hostname ssh1.private.com
$ sudo hostnamectl --static set-hostname ssh1.private.com
$ sudo hostnamectl --pretty set-hostnmame ssh1.private.com
```