Linux server security best practices

* Last updated on: 2017-04-11

* Authored by: Christoph Champ and Reese McJunkin

The first step after you create a Linux cloud server should be to set the security on it. This crucial step must be performed on every server to prevent hackers from obtaining unwanted access. The result is a more secure environment that helps prevent you and your business from being hacked. Performing these basic steps and hardening the security on your server *should* make hackers give up and move on to a new target.

**User management**

By default on every Linux system, the root user is created as the first user. The root user should be used only for the initial configuration of the system and should then be disabled via Secure Shell (SSH). Disabling this root user via SSH makes it harder for a hacker to gain access to the system. Because the root user is created by default on every Linux server, hackers already have half the information they need to log in to your server if the root user is enabled via SSH. All they need to do is run any number of brute-force SSH attacks until the password hash is broken.

To avoid this situation, you must create a secondary user to use when you need to log in and administer the system. Each end user on the system should have their own login credentials for logging purposes. Depending on the actions that the end user will perform, they might need to have sudo permission to perform administrative actions. This section provides examples on how to add a user with sudo permission on both Debian- and Red Hat Enterprise Linux-based systems.

**Password strength guidelines**

Before you create any users, ensure that you use strong passwords that require a minimum length (and maybe even include expiration dates). Here are common guidelines advocated by proponents of software system security:

* Use a minimum password length of 12 to 14 characters, if permitted.
* Include lowercase and uppercase alphabetic characters, numbers, and symbols, if permitted.
* Generate passwords randomly where feasible.
* Avoid using the same password twice (for example, across multiple user accounts or software systems).
* Avoid character repetition, keyboard patterns, dictionary words, letter or number sequences, user names, relative or pet names, romantic links (current or past), and biographical information (for example, ID numbers, ancestors’ names, or dates).
* Avoid using information that is or might become publicly associated with the user or the account.
* Avoid using information that the user’s colleagues or acquaintances might know to be associated with the user.
* Do not use passwords that consist wholly of any simple combination of the aforementioned weak components.

**Add a user (Debian and Ubuntu)**

1. Create a new user and set the user’s password:
2. adduser {username}
3. Give the new user permission to use sudo to perform privileged operations on the system. This new user will be your main user when logging in remotely and making changes to the server.

a. Run the following command as root to edit the list of user permissions:

visudo

b. Add the following line directly after the line containing root ALL=(ALL:ALL) ALL:

{username} ALL=(ALL:ALL) ALL

c. Save and quit.

1. Switch to the new user and test its permissions by using sudo to run a command that normally requires root access:
2. su {username}
3. sudo iptables -L

You are asked to enter the new user’s password for verification before the command is executed.

If several lines about INPUT and OUTPUT appeared, the new user has sudo permissions and you can skip to the next section. You should log in with user instead of root whenever possible. Using sudo will help you avoid making inadvertent system changes, and your changes will be logged for future reference.

**Add a user (Red Hat and CentOS)**

1. Create a new user with adduser and set the user’s password with passwd:
2. adduser {username}
3. passwd {username}
4. Give the new user permission to use sudo to perform privileged operations on the system. This new user will be your main user when log in remotely and make changes to the server.

a. Run the following command:

visudo

**Note:** On some distributions, the text editor that system uses for visudo is vi. It’s not a user-friendly editor, so you may need to consult a [**vi tutorial**](http://bignosebird.com/docs/vi.shtml) for help.

b. Add the following line directly after the line containing root ALL=(ALL:ALL) ALL:

{username} ALL=(ALL) ALL

c. Save and quit.

1. Switch to the new user and test its permissions by using sudo to run a command that normally requires root access:
2. su {username}
3. sudo iptables -L

You are asked to enter the new user’s password for verification before the command is executed.

If several lines about INPUT and OUTPUT appeared, the new user has sudo permissions and you can proceed to the next section. You should log in with this user instead of root whenever possible. Using sudo will help you avoid making inadvertent system changes, and your changes will be logged for future reference.

**Generate an SSH key pair**

For a login method that is more secure than using a password, create an SSH key pair to use with the user that you previously created. These instructions work with any Linux distribution.

**Note:** These instructions are for Linux and Mac OS X desktops. If you are connecting from a Windows desktop, follow the instructions in [**Generate RSA Keys with SSH PUTTYgen**](https://support.rackspace.com/how-to/generating-rsa-keys-with-ssh-puttygen/) and [**Log in with a SSH private key on Windows**](https://support.rackspace.com/how-to/logging-in-with-an-ssh-private-key-on-windows) to generate and add the SSH key pair.

1. Run the following command to generate a key pair on your *local* Linux or Mac computer.
2. ssh-keygen -b 4096 -t rsa

When asked where to save the key, use the default location is. Adding a password is optional; it’s more secure, but can be inconvenient.

Two files are created. The default names are id\_rsa for your private key and id\_rsa.pub for your public key.

1. After you have created the key pair on your local computer, upload the public key to your remote server for the user that you created previously.

**Warning:** Be sure to upload the *public* key, and *not* your private key.

ssh-copy-id -i ~/.ssh/id\_rsa.pub {username}@{remotePublicIPAddress}

1. Connect to the remote server, with ssh {username}@{remotePublicIPAddress}, and run the following command to verify that no extra keys were added that you were not expecting:
2. cat .ssh/authorized\_keys

**Linux SSH daemon configuration**

Now that your additional user is created with sudo permissions and an SSH key pair, you can move on to the next part of securing your Linux system. You will work with the SSH daemon (server) configuration to improve security.

**Note:** **Managed Operations and RackConnect customers only** To ensure that our automated systems have access to your server when needed, we request that you do not change the SSH configuration, and that you skip to the next section. When connecting to your server, Rackspace Support logs in as the user rack using password authentication on port 22. In addition, rebuilding existing servers or building a new server from a snapshot requires that root logins are enabled via the PermitRootLogin option set to yes. If you need to change these values, speak with an administrator at Rackspace to do so in a way that does not impact our ability to provide you with**Fanatical Support**®.

The example commands assume that you’re no longer logged in as root and are logged in as your new user, using sudo to perform privileged operations.

**SSH configuration options**

This section covers common options in the SSH configuration file that help improve security. You will use this information to configure your firewall later.

This section outlines only a few options to change and what they do. For details about on other configuration options, see the **[OpenSSH documentation](http://www.openssh.com/cgi-bin/man.cgi/OpenBSD-current/man5/sshd_config.5)**.

This section focuses on the following options:

* Port XX — This is the port on which the SSH daemon listens (port 22 by default).
* PermitRootLogin — This flag enables (yes) or disables (no) root login via SSH.
* PubkeyAuthentication — This flag enables (yes) or disables (no) SSH keys for authentication.
* PasswordAuthentication — This flag enables (yes) or disables (no) password authentication.

SSH uses port 22 by default for communication. Hackers try port 22 with the username root on every server that they attack. For this reason, disabling the root user via SSH and changing SSH to listen on a nonstandard port helps prevent a breach.

Changing the port won’t stop a determined intruder, but it does cause most superficial scans for SSH connection opportunities to overlook your server. Similarly, removing SSH access for the root user interferes with casual brute force attacks via SSH.

Another part of this configuration is the authentication method to use when logging in. By default, all Linux systems use password authentication. There are multiple ways to perform authentication on the server, but the main two are using a password and SSH keys.

SSH keys are generated in pairs, one public and the other private, and they can be used only in combination with each other. The private key is meant to be stored in a safe location on the computer from which you connect, and should never be given out. The public key can be given out, and it is that key that you place on the server to which you are connecting. The private key on your local computer is run through an algorithm when you make a connection, granting access if the key pair hash matches up with the public key.

**Modify sshd\_config**

By this point, you have added a new user with sudo permissions, created an SSH key pair, and uploaded your public SSH key. Now, change your SSH configuration file to improve your security. To do this, you can change SSH to listen on a custom port, restrict root login via SSH, enable public key authentication (already enabled for Ubuntu 14.04), and disable password authentication.

1. Open the SSH daemon configuration file for editing:
2. sudo vi /etc/ssh/sshd\_config
3. Change the following lines (original value > new value):
4. Port 22 > Port 9001
5. PermitRootLogin yes > PermitRootLogin no
6. PasswordAuthentication yes > PasswordAuthentication no

Replace **9001** with the port you want to use. Ensure the new port isn’t already in use by another program by using a tool **[netstat](https://support.rackspace.com/how-to/checking-listening-ports-with-netstat)**.

**Important:** As mentioned earlier, you should not make this change to the sshd\_config file if your server has a Managed Operations service level. These changes could prevent Rackspace access to your server.

1. Test the altered SSH configuration for errors by running the following command:
2. sshd -t

SSH is now configured to run on a custom port and accept only nonroot users that pass a valid SSH key. For these settings to apply and persist, you must restart the SSH service. However, do not restart the service yet. Restarting SSH now might lock you out of the server, requiring you to use [**rescue mode**](https://support.rackspace.com/how-to/linux-server-security-best-practices/how-to/rescue-mode) or the [**web console**](https://support.rackspace.com/how-to/start-a-console-session) to restore the configuration. You must configure the firewall before restarting the server. The firewall is discussed in the next section.

**Amend iptables and restart SSH**

**Note:** **RackConnect customers** To manage firewall rules, use the RackConnect management instead of iptables on the server. You shouldn’t change the SSH port, either, if you use RackConnect, but for more information about firewalls and RackConnect, see [**Managing RackConnect v2.0 network policies**](https://support.rackspace.com/how-to/managing-rackconnect-v20-network-policies).

1. Verify that the firewall is open. By default, Ubuntu does not have any restrictions, whereas CentOS and Red Hat do. The following output shows what the system looks like when no ports are blocked and the policy is ACCEPT:
2. $ sudo iptables -L
3. Chain INPUT (policy ACCEPT)
4. target prot opt source destination
5. Chain FORWARD (policy ACCEPT)
6. target prot opt source destination
7. Chain OUTPUT (policy ACCEPT)
8. target prot opt source destination
9. You can open the new SSH port by running the following command:
10. sudo iptables -I INPUT -p tcp --dport 9001 -m state --state NEW,ESTABLISHED -j ACCEPT

Replace **9001** with the port that you used for the SSH daemon.

If you are running Red Hat Enterprise Linux 5 or CentOS 5, change the INPUT in the command to match the name of the input rules in your current iptables rules. For example:

sudo iptables -I RH-Firewall-1-INPUT -p tcp --dport 9001 -m state --state NEW,ESTABLISHED -j ACCEPT

1. Run the following command to restart the SSH daemon so that the daemon applies the new configuration you set up:
2. sudo service ssh restart

If you get an error, the SSH daemon may be named sshd, so instead run the following command:

sudo service sshd restart

1. Open another window and log in to the server as the user that you created previously. Keep your original connection active in case you need to troubleshoot the configuration.

To connect to SSH with the new configuration you might need to specify the port number and key to use. For example:

ssh -p 9001 -i ~/.ssh/id\_rsa {username}@{remotePublicIPAddress}

The -p option specifies the port, and the -i option specifies the private key to use for the connection.

If you’re connecting from a Windows desktop, when you create the connection in **[PuTTY](https://support.rackspace.com/how-to/connecting-to-linux-from-windows-by-using-putty)**, you can specify the port number and a private key.

**Save the iptables rule**

If you were able to connect with the new configuration, save your iptables rules before continuing to ensure SSH port stays open.

On Ubuntu and Debian, run the following command:

sudo -c "iptables-save > /etc/iptables.rules"

On CentOS, Red Hat, and Fedora, run the following command:

sudo service iptables save

**Firewalls**

iptables is a firewall and networking tool that is available to all Linux distributions and operates by analyzing packets at the kernel level as they are received.

For an introduction to iptables and how to use it, read [**Introduction to iptables**](https://support.rackspace.com/how-to/introduction-to-iptables).

**Note:** **RackConnect users** To manage firewall rules, use the RackConnect management interface to manage firewall rules instead of iptables on the server. For more information, see [**Managing RackConnect v2.0 network policies**](https://support.rackspace.com/how-to/managing-rackconnect-v20-network-policies).

**Sample iptables ruleset**

**Important:** The following code is only a template to help you build an iptables ruleset that will best fit your solution. It is not intended to be a security standard nor will it fit every environment. Use this sample only to help you with syntax and ideas on how to use iptables.

Following are the default locations where iptables rulesets are saved on a given Linux distribution. Some distributions allow you to save elsewhere by using the iptables-save command.

* Fedora, Red Hat Enterprise Linux, and CentOS: /etc/sysconfig/iptables
* Ubuntu and Debian: /etc/iptables.rules

You can save the following example iptables ruleset in the appropriate location for your distribution. Change the rule for port 9001 to use the port that you set up for SSH, or remove that line if you didn’t change the SSH port, remove the line.

**Note:** **RHEL 5 and CentOS 5 servers** Red Hat Enterprise Linux 5 and CentOS 5 and earlier useRH-Firewall-1-INPUT instead of INPUT in their firewall rules. In this example file, the firewall rules are the entries that begin with -A. For Red Hat Enterprise Linux 5 and CentOS 5 and earlier, the second line should still use INPUT.

`\*filter

# Dropping incoming connections that don't have explicit rules below

:INPUT DROP [68:4456]

:FORWARD ACCEPT [0:0]

:OUTPUT ACCEPT [1628:151823]

# Allow established connections for both public and private connections

-A INPUT -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT

-A INPUT -i eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT

# Opening ports wide open

-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 21 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 3306 -j ACCEPT

-A INPUT -p tcp -m tcp --dport 9001 -j ACCEPT

# Opening a port to a specific IP

-A INPUT -p tcp -m tcp --dport 10000 -s 192.168.1.1 -j ACCEPT

# Opening a port to a range of IPs

-A INPUT -p tcp -m tcp --dport 20000 -s 192.168.0.0/24 -j ACCEPT

# Commmiting the rules to the firewall

COMMIT`

You should then restart the iptables service to activate your firewall rules:

sudo service iptables restart

If your system tells you there isn’t an iptables service, use iptables-restore to load the new rules. Run the following command, using the location of your iptables rules file as the second argument:

sudo -c "/sbin/iptables-restore < /etc/iptables.rules"

**Account-level security**

Although securing your servers is an essential part of operations on the Internet, securing your account is also necessary. Your account name, password, and API keys are essential parts of how you interact with the Rackspace Cloud offerings. Just like any other password or access credentials, you want to keep them secure, but you also need to allow your team to take action and perform necessary tasks.

Through the use of [**Role Based Access Control (RBAC)**](https://support.rackspace.com/how-to/overview-role-based-access-control-rbac), you can create users and grant permissions to individuals or applications that are responsible for using various Rackspace services. By leveraging RBAC, you can give your team and contractors access only the utilities that they need and revoke the access when and if necessary.

Following are some usage scenarios:

* Give contractors access to set up the environment that you have hired them to create while limiting their ability to view or change any credit card information or delete your account.
* Allow your accountant to see the bill but not to delete your servers.
* Hire a DBA and give the DBA access to your DBaaS instances.
* Allow a client to upload files directly to your Cloud Files account.
* Configure your servers to register and use specific users for your monitoring and backup agents that are separate from your admin account.

For more information about RBAC, see the [**Role-Based Access Control (RBAC) FAQ**](https://support.rackspace.com/how-to/faq-role-based-access-control-rbac/).

**Simple intrusion prevention**

Most would-be intruders run multiple attacks against the same port to try to find something that they can exploit in the software running on that port. Fortunately, you can set up an intrusion prevention tool like[**Fail2ban**](http://www.fail2ban.org/wiki/index.php/Main_Page) on your server to block repeated attacks on a port.

**Note:** Managed Operations servers will have Fail2Ban installed and configured by default to watch over SSH login attempts. Contact your support team if you have any questions or concerns.

Fail2ban monitors logs and automatically blocks connections if it sees too many from the same host in a short period of time. To set up and configure Fail2ban on your server, see [**Basic Cloud Server security**](https://support.rackspace.com/how-to/basic-cloud-server-security)

**Intrusion detection**

An intrusion detection system (IDS) can help an administrator monitor systems for suspicious activity and possible exploits. An IDS is more robust than a prevention tool like Fail2ban, but can be more complicated to set up and maintain.

A popular open source IDS is [**OSSEC**](http://www.ossec.net/). OSSEC maintains agents on multiple systems that report back to a main server, allowing investigation of logs and alerts from a potentially compromised server even if that server is shut down.

If you suspect that a system has already been compromised, you can investigate with procedures such as [**checking for backdoors and intruders**](https://support.rackspace.com/how-to/check-for-a-security-compromise-back-doors-and-intruders) and [**rescue mode investigation**](https://support.rackspace.com/how-to/check-for-a-security-compromise-rescue-mode-investigation).

**Keep your OS up to date (patching)**

Keeping your kernel, packages, and dependencies up-to-date is very important. This is especially true for security-related modules and packages. Some updates (for example, kernel updates) require your server to be rebooted. Plan for a time that will cause the least amount of disruption during the (normally very short) downtime while your server is rebooting.

To check for and install updates on Ubuntu and Debian systems, run the following commands:

sudo apt-get update

sudo apt-get upgrade

To check for and install updates on CentOS, Red Hat, and Fedora systems, run the following command:

sudo yum update

**Operating system end-of-life**

Find out when the Linux distribution release that you are running on your servers reaches its end-of-life (EOL). When a release reaches its EOL, the distribution’s maintainers no longer support it or supply package updates via their official repositories. You want to plan your migration to a newer release well before your current release reaches its EOL.

Use the following links to find out when your Linux distribution release will reach its EOL:

* Ubuntu: [**https://wiki.ubuntu.com/Releases**](https://wiki.ubuntu.com/Releases)
* Red Hat Enterprise Linux: [**https://access.redhat.com/support/policy/updates/errata/**](https://access.redhat.com/support/policy/updates/errata/)
* CentOS: Same as Red Hat
* Fedora: [**https://fedoraproject.org/wiki/End\_of\_life**](https://fedoraproject.org/wiki/End_of_life)
* Debian: [**https://wiki.debian.org/DebianReleases**](https://wiki.debian.org/DebianReleases)