Raspberry Pi - Email Server

# Overview

Guide to setting up a RaspberryPi 3 (RPi) IMAP email server with Ubuntu 20.04, Postfix and Dovecot.

# Install Ubuntu

## Write OS to SD

Download lastest copy of Ubuntu Server for Raspberry Pi (v20.04 in at time of writing):

<https://ubuntu.com/download/raspberry-pi>

Write to SD card on unix system using Etcher:

<https://www.balena.io/etcher/>

Remove and reinsert the Ubuntu boot drive to mount it in macOS.

To allow for headless ssh access, touch a ssh file into the Ubuntu boot drive:

touch /Volumes/system-boot/ssh

Remove and put the SD into the Rasperry Pi and connect to the network. SSH into the server using the default login credentials:

ssh ubuntu@<server-ip-address>

=> enter password

## Update System

Once sucessfully logged in, update the system using:

sudo apt-get update

sudo apt-get upgade

## DNS Servers

Update the default DNS servers on the server to be much quicker ones such as cloudfare and google. This will help when the email server performs reverse DNS lookups on IP addresses. Install resolvconf:

sudo apt-get install resolvconf

Then add the name servers to the config file:

sudo vim /etc/resolvconf/resolv.conf.d/head

nameserver 1.1.1.1

nameserver 8.8.8.8

nameserver 1.0.0.1

nameserver 8.8.4.4

Update the system with:

sudo resolvconf --enable-updates

sudo resolvconf -u

# Security

In order to run an RPi server exposed to the web, various system defaults require changing to maximise security and stop attackers gaining control of the system.

## Disable Ubuntu User

Having the default user enabled on the system for ssh login is dangerous, since attackers are likely to test loggin in with this first.

Disable the ubuntu user by create a new admin user under a different name, and deleting the old. Create a new user and add them to the sudo group using:

sudo adduser <username>

sudo adduser <username> sudo

Close the SSH session and restart it using the new user, then delete the ubuntu user using:

sudo deluser -remove-home ubuntu

## Password on Sudo

To limit scope if an attacker does access a sudoer user account, make the use of sudo require a password each time. To do this login as root and edit the sudoers group:

sudo su

sudo vim /etc/sudoers.d/90-cloud-init-users

Edit ubuntu line to:

ubuntu ALL=(ALL) PASSWD: ALL

exit and restart the SSH session.

## Disable SSH Root Login

Stop attacker logging into root directly via ssh by editing the ssh daemon file for the server:

sudo vim /etc/ssh/sshd\_config

Change the PermitRootLogin setting, uncomment and set to no:

PermitRootLogin no

Restart the ssh daemon:

sudo service ssh restart

## Change SSH Port

Similarly to trying the ubuntu user first, attackers will also scan for prospective default ssh ports which respond to their requests. Therefore, to limit this change the default port to another available port:

sudo vim /etc/sshd\_config

Uncomment and change Port setting to new port:

Port <new-port>

Restart ssh daemon

sudo service ssh restart

Exit ssh session and restart using the -p flag to specify the new port:

ssh <user>@<server-ip-address> -p <new-port>

## SSH Login with SSH Keys

Its much more secure to use strong crypography keys such as those generated by OpenSSH for user authentication on SSH login, than potentially vunerable user created passwords.

While logged into the server, create a new ssh folder to store keys with authorized keys list in the user home directory:

mkdir ~/.ssh

touch ~/.ssh/authorized\_keys

Exit the session and create a new ssh key for the user in .ssh folder on your local machine using openssh, add a password for extra security:

ssh-keygen -t rsa -b 2048 -f ~/.ssh/<key-name> -C "<user-defined-comment>"

Add public keys to server from your pc using ssh cat:

cat ~/.ssh/<key-name>.pub | ssh <username>@<server-ip-address> -p <port> "cat >> /home/<username>/.ssh/authorized\_keys"

Add key to local machine using ssh-add:

ssh-add ~/.ssh/<key-name>

Login to server using ssh, no password should be required. Disable password authentication on the ssh daemon, to stop attackers from trying to crack your password:

sudo vim /etc/ssh/sshd\_config

Change following settings to no:

PasswordAuthentication no

ChallengeResponseAuthentication no

Restart ssh daemon

sudo service ssh restart

## Unattended Upgrades

Keeping software upto date is critical for system security, therefore add unattended upgrades to the server to keep it upto date automatically.

Install unattended-upgrades:

sudo apt-get install unattended-upgrades

Edit the configuration:

sudo vim /etc/apt/apt.conf.d/50unattended-upgrades

Uncomment the updates line so that it looks like this:

"${distro\_id}:${distro\_codename}-updates";

It is recommended to enable the following by uncommenting, but if the system is critical do not allow automatic rebooting:

Unattended-Upgrade::Remove-Unused-Kernel-Packages "true";

Unattended-Upgrade::Remove-New-Unused-Dependencies "true";

Unattended-Upgrade::Remove-Unused-Dependencies "false";

Unattended-Upgrade::Automatic-Reboot "false";

Unattended-Upgrade::Automatic-Reboot-Time "02:55";

Enable update intervals by adding the following lines of code fo the auto-upgrades file:

sudo vim /etc/apt/apt.conf.d/20auto-upgrades

APT::Periodic::Update-Package-Lists "1";

APT::Periodic::Download-Upgradeable-Packages "1";

APT::Periodic::AutocleanInterval "7";

APT::Periodic::Unattended-Upgrade "1";

Test with dry-run:

sudo unattended-upgrades --dry-run --debug

The server should now update once a day.

## Firewall

While having a firewall on the WAN to LAN router is common to stop unauthorized packets entering the network, it is also important to have a firewall on the server itself to control what packets enter the system. Therefore, make sure iptables is install to start creating a firewall in ubuntu:

sudo apt-get install iptables

Then install iptables persistent so that the firewall rules are reloaded on system reboot, which will keep saved rules in /etc/iptables/rules.v4 (.v6):

sudo apt-get install iptables-persistent

For a email server there are a number of ports required to be open to accept and send emails for users. First up is incoming connections and the following ports will be required:

* TCP 25: SMTP - used for receiving and relaying emails from other servers or clients
* TCP 587: SMTPs - secure SMTP used by clients relaying email
* TCP 993: IAMPs - secure IMAP
* TCP <ssh-port>: SSH - typically 22 but changed to custom during security setup, this setup limits the ip address of SSH connections to block attackers using the '-s' flag, remove this section if you want to allow ssh from anywhere
* Allow loopback traffic for applications running on the server localhost ports.
* Established and related ports - Allows new incomming connections on any port if they are already established or related to other connections

Forward connections should be dropped and outgoing connections allowed.

Create a new iptables ruleset fr IPv4 file for the above ports:

sudo vim /etc/iptables/current-rules.v4

Entering:

\*filter

:INPUT DROP [0:0]

:FORWARD DROP [0:0]

:OUTPUT ACCEPT [0:0]

# Inbound Rules - Drop unless rule allows

# ======================================

# Allows new SSH connections from local machine IP

-A INPUT -i eth0 -s <source-ip> -p tcp -m state --state NEW --dport <ssh-port> -j ACCEPT

# Allows new SMTP connections

-A INPUT -i eth0 -p tcp -m state --state NEW --dport 25 -j ACCEPT

-A INPUT -i eth0 -p tcp -m state --state NEW --dport 587 -j ACCEPT

# Allows new IMAPs connections

-A INPUT -i eth0 -p tcp -m state --state NEW --dport 993 -j ACCEPT

# Accepts all established inbound connections

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

# Allows all loopback, drop all traffic to 127/8 that doesn't use lo0

-A INPUT -i lo -j ACCEPT

-A INPUT ! -i lo -d 127.0.0.0/8 -j DROP

# Forward Rules - Drop all

# ======================================

# Outbound Rules - Allow all

# ======================================

COMMIT

Create new ruleset for IPv6 blocking all connection (change this if you are connecting to your router through IPv6:

sudo vim /etc/iptables/current-rules.v6

Adding:

\*filter

:INPUT DROP [0:0]

:FORWARD DROP [0:0]

:OUTPUT DROP [0:0]

COMMIT

Restore the newly created rules using:

sudo iptables-restore /etc/iptables/current-rules.v4

sudo ip6tables-restore /etc/iptables/current-rules.v6

Check rules have been applied using:

sudo iptables -L -v

sudo ip6tables -L -v

Save the rules to persistence by loggin into root and using:

sudo su

iptables-save > /etc/iptables/rules.v4

ip6tables-save > /etc/iptables/rules.v6

Reboot the system, ssh in again, and check whether the rules are still applied:

reboot

ssh <username>@<server-ip-address> -p <ssh-port>

sudo iptables -L -v

sudo ip6tables -L -v

## Logs

Keeping upto date with logs is important for ensuring security. In general all logs are kept in /var/log/ and some important ones to take note of are:

* /var/log/syslog: main log file for all services
* /var/log/message: whole systems log file
* /var/log/auth.log: all authentication attempts are logged here
* /var/log/mail.log: trace of recent emails sent here

# ZSH

ZSH is nice… so install I like to use it instead of bash.

## Install

Install using:

sudo apt-get install git zsh

Change default shell for user:

chsh -s /bin/zsh

Install oh-my-zsh (package manager for zsh):

sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"

Add plugins:

git clone https://github.com/zsh-users/zsh-syntax-highlighting.git ${ZSH\_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-syntax-highlighting

git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH\_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions

git clone https://github.com/zsh-users/zsh-completions ${ZSH\_CUSTOM:=~/.oh-my-zsh/custom}/plugins/zsh-completions

Install powerlevel9k (current theme):

git clone https://github.com/bhilburn/powerlevel9k.git ~/.oh-my-zsh/custom/themes/powerlevel9k

Use default theme or apply a custom .zshrc config.

Note environment variables for the shell can be added into .zshrc.

# DNS

For emails to be routed to the correct address DNS records are required.

## MX

Mail Exchange (MX) records are the DNS records for delivering mail. When a email server wants to send mail, it will look up a MX record which will direct it to the mail server which is responsible for accepting mail on behalf of the domain.

Create an MX record for your domain:

Domain Type Host Priority

<your-domain> MX <your-email-server-host> <val>

example.com MX example.com 10

Higher priority servers will get sent to first, using numbers here for many different servers can be used for primitive load balancing.

## A Record

An A Record will resolve a domain or subdomain request to an IP address, and can therefore be used to direct the mail subdomain to the server IP address.

Domain Type Host

<your-domain> A <ip>

example.com A <server-public-ip-address>

Note the server IP address must have the ports which were opened earlier in firewall, open for use. It is common for a ISP to block incomming and outgoing port 25 request to stop spam.

# Email MTA - Postfix

An email Mail Transfer Agent (MTA) is a core email application which transmits emails between servers, allowing emails to be sent and received. Postfix is a common linux MTA and is very configurable and quick to setup.

## Postfix Install

Install using:

sudo apt-get update

sudo apt-get install mailutils libsasl2-modules

sudo DEBIAN\_PRIORITY=low apt-get install postfix

DEBIAN\_PRIORITY=low will give a setup GUI to answer the following:

* Internet Site - Mail is sent a received directly
* Domain name - registered domain name
* Postmaster mail recipient - emails sent to 'root@domain' and 'postmaster@domain' will get sent to this unix user account
* Other mail destinations - Other domains which the server is the final destination for
* Synchronous updates - 'no', mail is processed slower as the system will only process commands synchronously
* Mail relay - Use only localhost or spam bots will send mail to other addresses
* Mailbox size - 0 for unlimited
* Local address extension: leave default
* IP protocols - select all for compatibility

This will then give the default setup for postfix on the system.

The postfix configuration file can be found at /etc/postfix/main.cf, however configuration settings can be edited directly using the command:

sudo postconf -e '<key=value>'

Enable and start the service using:

sudo systemctl enable postfix

sudo systemctl start postfix

## Enabling TLS with SSL Certificates

Option for self signed or CA signed SSL certificates.

**Self Signed**

Create a self signed SSL certificate for the server to allow for secure email connections. Use open ssl to generate a certificate:

openssl genrsa -des3 -out <domain-name>.pem 2048

Create a certificate signing request (only fill in common name option with domain name):

openssl req -new -sha256 -key <domain-name>.pem -out <domain-name>.csr

Create self signed certificate from signing request:

openssl req -x509 -sha256 -days 365 -key <domain-name>.pem -in <domain- name>.csr -out <domain-name>.crt

Remove password on key:

openssl rsa -in <domain-name>.pem -out <domain-name>.pem.nopass

mv <domain-name>.pem.nopass <domain-name>.pem

Copy files to correct location:

sudo chmod 600 <domain-name>.pem

sudo chmod 600 cakey.pem

sudo mv <domain-name>.pem /etc/ssl/private/

sudo mv <domain-name>.pem /etc/ssl/certs/

sudo mv cakey.pem /etc/ssl/private/

Add certs to the postfix configuration:

sudo postconf -e 'smtpd\_tls\_key\_file = /etc/ssl/private/<domain-name>.pem'

sudo postconf -e 'smtpd\_tls\_cert\_file = /etc/ssl/certs/<domain-name>.pem'

**Signed by LetsEncrypt (Preffered)**

Install the acme.sh ACME client for LetsEncrypt certificate signing software. Since this is an email server, performing the dns ownership validation is more suitable via the DNS-01 challenge, since it does not require the opening of port 80. However, this challenge does require API access to the DNS provider, if this is not available use a ACME client such as certbot.

curl https://get.acme.sh | sh

This will:

* Create the ~/.acme.sh/ folder for certs
* Create an alias acme.sh
* Create a daily cron job to check and renew certs if required

Add DNS provider API keys to environment variables (risk if server is comprimised, read up on security of API keys on live servers), https://github.com/acmesh-official/acme.sh/wiki/dnsapi :

export Key="<key>"

export Secret="<secret>"

Run following command to create certificate:

acme.sh --issue --dns dns\_ -d example.com

This will create a certificate and put it in the ~/.acme.sh directory.

Add certs to the postfix configuration:

sudo postconf -e 'smtpd\_tls\_key\_file = /home/<username>/.acme.sh/<domain-name>/<domain-name>.key'

sudo postconf -e 'smtpd\_tls\_cert\_file = /home/<username>/.acme.sh/<domain-name>/fullchain.cer'

## SMTP Relay Restrictions

To stop the server relaying spam mail, set restrictions are required on smtp sessions.

Domain and interface:

sudo postconf -e 'mydomain = <your-domain>'

sudo postconf -e 'inet\_interfaces = all'

sudo postconf -e 'inet\_protocols = all'

Clients and senders:

sudo postconf -e 'smtpd\_helo\_required = yes'

sudo postconf -e 'smtpd\_helo\_restrictions = reject\_invalid\_helo\_hostname reject\_unknown\_helo\_hostname'

sudo postconf -e 'smtpd\_sender\_restrictions = reject\_non\_fqdn\_sender reject\_unknown\_sender\_domain'

Restrictions on relaying of mail:

sudo postconf -e 'smtpd\_relay\_restrictions = permit\_sasl\_authenticated permit\_mynetworks reject\_unauth\_destination reject\_invalid\_hostname'

Spam filters on relaying to recipient:

sudo postconf -e 'smtpd\_recipient\_restrictions = permit\_sasl\_authenticated permit\_mynetworks reject\_rbl\_client zen.spamhaus.org reject\_rhsbl\_reverse\_client dbl.spamhaus.org reject\_rhsbl\_helo dbl.spamhaus.org reject\_rhsbl\_sender dbl.spamhaus.org'

Delay evaluation of restriction list for better logging and compatibilty with servers which don't accept rejection early into SMTP session:

sudo postconf -e 'smtpd\_delay\_reject = yes'

Block clients that speak too early:

sudo postconf -e 'smtpd\_data\_restrictions = reject\_unauth\_pipelining'

Disable VRFY command, which allows postfix to let anyone determine if an account exists on the mailserver. Having this setting enabled allows hackers to easily target account which they know exist:

sudo postconf -e 'disable\_vrfy\_command = yes'

## SMTP Authentication

Add STMP Authentication with Dovecot to postfix's configuration to allow clients to identify themselves securely.

SASL setup:

sudo postconf -e 'smtpd\_sasl\_local\_domain = <your-domain>'

sudo postconf -e 'smtpd\_sasl\_auth\_enable = yes'

sudo postconf -e 'smtpd\_sasl\_security\_options = noanonymous'

sudo postconf -e 'smtpd\_sasl\_type = dovecot'

sudo postconf -e 'smtpd\_tls\_security\_level = encrypt'

sudo postconf -e 'smtpd\_sasl\_path = private/auth'

sudo postconf -e 'smtpd\_tls\_auth\_only = no'

sudo postconf -e 'smtpd\_tls\_session\_cache\_timeout = 3600s'

sudo postconf -e 'smtpd\_tls\_loglevel = 1'

sudo postconf -e 'smtpd\_tls\_received\_header = yes'

sudo postconf -e 'smtpd\_tls\_security\_level = may'

sudo postconf -e 'smtp\_tls\_note\_starttls\_offer = yes'

sudo postconf -e 'smtp\_tls\_security\_level = may'

sudo postconf -e ' smtp\_tls\_protocols = TLSv1.3, TLSv1.2, !TLSv1, !TLSv1.1, !SSLv2, !SSLv3'

sudo postconf -e 'tls\_random\_source = dev:/dev/urandom'

Enable the smtp submission port 587, in the master configuration:

sudo vim /etc/postfix/master.cf

Uncomment the submission inet line:

submission inet n - n - - smtpd

Virtual Alias Maping

With virtual alias domains, each hosted email address can be aliased (mapped) to a local UNIX system account or remote address. Enable mapping with:

sudo postconf -e 'virtual\_alias\_domains = $mydomain'

sudo postconf -e 'virtual\_alias\_maps = hash:/etc/postfix/virtual'

Then create the mapping file:

sudo vim /etc/postfix/virtual

and mappings can be entered simply as the following on each line:

<email-address> <unix-user>

for example some standard mappings would be:

postmaster@example.com root

root@example.com root

info@exampe.com info

@<domain-name> nobody

The nobody alias can be used to delete mail which gets past restrictions but has no local recipient. Create the alias by editing the system aliases file:

sudo vim /etc/aliases

postmaster:root

nobody: /dev/null

If you add mappings to users which don't yet exist, make sure to make them using adduser.

Then apply the mappings and aliases:

sudo postmap /etc/postfix/virtual

sudo postalias /etc/aliases

sudo systemctl restart postfix

## Check

Check over settings to see if everything looks correct, such as domain, hostname, networks, and certificates.

sudo vim /etc/postfix/main.cf

If everything looks good restart the service:

sudo systemctl restart postfix

# Email MDA - Dovecot

A Mail Delivery Agent (MDA) gets messages from the email server MTA into the users inboxes and folders using protcols such as POP3 and IMAP, generally handling storage and access of already received emails on the server. Dovecot is widely used, highly configurable and is compatible with Postfix. For this guide, IMAPs will be used.

Dovecot Install

Dovecot is a delivery agent which works very well with Postfix. Install dovecot SASL with the common package and then the dovecot plugins for IMAP and SMTP to give mail delivery protocols for clients such as Outlook and Rails:

sudo apt install dovecot-common dovecot-imapd

Dovecots configurations files will then be located as /etc/dovcot/conf.d/.

Mailboxes

Add the mailbox location to postix for the logged in user in the home directory:

sudo postconf -e 'home\_mailbox = Maildir/'

Then add default mailbox setup for any new users which get added:

sudo maildirmake.dovecot /etc/skel/Maildir

sudo maildirmake.dovecot /etc/skel/Maildir/.Drafts

sudo maildirmake.dovecot /etc/skel/Maildir/.Sent

sudo maildirmake.dovecot /etc/skel/Maildir/.Trash

sudo maildirmake.dovecot /etc/skel/Maildir/.Templates

Add default mailbox to admin user:

sudo cp -r /etc/skel/Maildir /home/$USER/

sudo chown -R $USER:$USER /home/$USER/Maildir

sudo chmod -R 700 /home/$USER/Maildir

sudo adduser $USER mail

Include mailbox Maildir location in the terminal and mail profiles:

echo 'export MAIL=~/Maildir' | sudo tee -a /etc/bash.bashrc | sudo tee -a /etc/profile.d/mail.sh

Make the mail delivery setting for Dovecot match that of Postfix:

sudo vim /etc/dovecot/conf.d/10-mail.conf

mail\_location = maildir:~/Maildir

SASL

To enable SASL, disable plaintext auth and enable auth login in the auth.conf file:

sudo vim /etc/dovecot/conf.d/10-auth.conf

disable\_plaintext\_auth = yes

auth\_mechanisms = plain

Add the SSL files for auth to the dovecot ssl config and update settings (use different cert and key path if selfsigning certs):

sudo vim /etc/dovecot/conf.d/10-ssl.conf

ssl = required

ssl\_cert = </home/<username>/.acme.sh/<domain-name>/fullchain.cer

ssl\_key = </home/<username>/.acme.sh/<domain-name>/<domain-name>.key

ssl\_min\_protocols = TLSv1.2

ssl\_prefer\_server\_ciphers = yes

Generate new dh keys (this can take some hours):

openssl dhparam -out /etc/dovecot/dh.pem 4096

IMAPs

Edit the master configuration to add the IMAP port so that clients can access emails:

sudo vim /etc/dovecot/conf.d/10-master.conf

Uncomment and edit the following lines:

service imap-login {

inet\_listener imaps {

port = 993

ssl = yes

}

}

service auth {

unix\_listener /var/spool/postfix/private/auth {

mode = 0660

user = postfix

group = postfix

}

}

Once all settings are set, check the dovecot configuration and restart the service using:

dovecot -n

sudo systemctl restart dovecot

sudo systemctl restart postfix

# Fail2Ban

Cracking passwords or keys, generally takes a lot of attempts, limit these using Fail2Ban rate limits.

Install using:

sudo apt-get install fail2ban

Start and enable the service using:

sudo systemctl start fail2ban

sudo systemctl enable fail2ban

## SSH

Add jail configuration for ssh logins on the new port number defined previously:

sudo vim /etc/fail2ban/jail.local

Add following contents:

[sshd]

enabled = true

port = <ssh-port-number>

filter = sshd

logpath = /var/log/auth.log

maxretry = 5

findtime = 600

bantime = 1200

Restart service:

sudo systemctl restart fail2ban

Test by attempting to ssh with an unkown and failing more than 3 times, after this the server will refuse to connect. Note your IP address will be locked out for 10 minutes, so make a cup of tea.

## IMAP

Add fail2ban filters and jail configuration, to stop repeated attack attempts.

Add filter regex:

sudo vim /etc/fail2ban/filter.d/dovecot-pop3imap.conf

[Definition]

failregex = (?: pop3-login|imap-login): .\*(?:Authentication failure|Aborted login \(auth failed|Aborted login \(tried to use disabled|Disconnected \(auth failed|Aborted login \(\d+ authentication attempts).\*rip=`<HOST>`

Add jail configuration:

sudo vim /etc/fail2ban/jail.local

[dovecot-pop3imap]

enabled = true

filter = dovecot-pop3imap

action = iptables-multiport[name=dovecot-pop3imap, port="pop3,imap,pop3s,imaps", protocol=tcp]

logpath = /var/log/mail.log

maxretry = 5

findtime = 600

bantime = 1200

Restart service:

sudo systemctl restart fail2ban

## SMTP

Add smtp fail2ban jail configuration, to stop repeated attack attempts.

Add SASL filter to fail2ban:

sudo vim /etc/fail2ban/filter.d/postfix-sasl.conf

# Fail2Ban filter for postfix authentication failures

[INCLUDES]

before = common.conf

[Definition]

\_daemon = postfix/smtpd

failregex = ^%(\_\_prefix\_line)swarning: [-.\_\w]+\[<HOST>\]: SASL (?:LOGIN|PLAIN|(?:CRAM|DIGEST)-MD5) authentication failed(: [ A-Za-z0-9+/]\*={0,2})?\s\*$

Add the jail configuration:

sudo vim /etc/fail2ban/jail.local

[sasl]

enabled = true

port = smtp, submission

filter = postfix-sasl

logpath = /var/log/mail.log

maxretry = 5

findtime = 600

bantime = 1200

[postfix]

enabled = true

port = smtp, submission

filter = postfix

logpath = /var/log/mail.log

maxretry = 5

findtime = 600

bantime = 1200

Restart service:

sudo systemctl restart fail2ban

# Deliverability

Now that mail can be relayed and received, there are various standards which need to be applied to the domain and server to allow for maximum deliverability of the mail, i.e. it doesn’t get classed as spam mail.

## SPF

Sender Policy Framework (SPF) is a open standard which allows a domain owner to place a TXT DNS entry on their domain, allowing for receiving servers to validate which domains and IP address are allowed to send emails on behalf of the domain.

When receiving mail, email servers will do a checkup on the Return-Path value to validate the originating server which send the mail. The checkup will validiate if the address is allowed by the sending domains SPF addresses. If the address is valid, the mail will go on to other checks, if it is invalid it will be marked as spam.

There are several parts to a SPF record:

* v=spf1 - version of SPF record being used
* a - if domain includes an A record of the sender, SPF will match
* ip4: - IPv4 addresses included (ip4:192.0.0.0)
* ip6: - IPv6 addresses included (ip6:2a05:d018:e3:8c00:bb71:dea8:8b83:851e)
* include: - domains included (include:mail.example.com)
* -all - anything else should be hard fail, mail rejected (if errors use ~)
* ~all - anything else should be soft fail, letting the email server decide next action

Create a SPF record for your domain including all servers and ip address which are allowed to send mail on your behalf. For example:

TXT DNS: v=spf1 a ip4:<server-ip> include:<another-mail-server-domain> ~all

SPF filtering can also be added to incoming mail, adding further spam checks. First install the postfix SPF system:

sudo apt-get install postfix-policyd-spf-python

Add the following to the end of the master postfix configuration file:

sudo vim /etc/postfix/master.cf

policyd-spf unix - n n - 0 spawn

user=policyd-spf argv=/usr/bin/policyd-spf

Edit the postfix main configuration file:

sudo vim /etc/postfix/main.cf

Add the following to the end of the smtpd\_recipient\_restrictions:

check\_policy\_service unix:private/policyd-spf

Add the following configuration:

sudo postconf -e 'policyd-spf\_time\_limit = 3600'

Restart postfix to apply changes:

sudo systemctl restart postfix

## DKIM - Setup

Domain Keys Identified Mail (DKIM) is a open standard which helps prevent spoofing on outgoing messages sent from your email server domain. DKIM public keys are added to the domains DNS records, then outgoing email servers add an encrypted signature of the email to the email header, which allows receiving servers to check whether the email was changed after it was sent.

To set up DKIM, add the required packages:

sudo apt-get install opendkim opendkim-tools

Add postfix to the opendkim group:

sudo gpasswd -a postfix opendkim

Edit the opendkim configuration file:

sudo vim /etc/opendkim.conf

Uncomment, edit and add the following:

Canonicalization relaxed/simple

Mode sv

SubDomains no

AutoRestart yes

AutoRestartRate 10/1M

Background yes

DNSTimeout 5

SignatureAlgorithm rsa-sha256

Add the following to the end of the file:

# Map domains in From addresses to keys used to sign messages

KeyTable refile:/etc/opendkim/key.table

SigningTable refile:/etc/opendkim/signing.table

# Hosts to ignore when verifying signatures

ExternalIgnoreList /etc/opendkim/trusted.hosts

# A set of internal hosts whose mail should be signed

InternalHosts /etc/opendkim/trusted.hosts

Create the signing and key tables and the trusted hosts file:

sudo mkdir /etc/opendkim

sudo mkdir /etc/opendkim/keys

sudo chown -R opendkim:opendkim /etc/opendkim

sudo chmod go-rw /etc/opendkim/keys

sudo touch /etc/opendkim/signing.table

sudo touch /etc/opendkim/key.table

sudo touch /etc/opendkim/trusted.hosts

Edit the signing table to contain the server domain:

sudo vim /etc/opendkim/signing.table

\*@example.com default.\_domainkey.example.com

Edit the key table to contain the location of the private signing key:

sudo vim /etc/opendkim/key.table

default.\_domainkey.example.com example.com:default:/etc/opendkim/keys/example.com/default.private

Edit the trusted hosts file to contain the localhost and server domain:

sudo vim /etc/opendkim/trusted.hosts

127.0.0.1

localhost

\*.example.com

Make the private-public keypair for opendkim:

sudo mkdir /etc/opendkim/keys/example.com

sudo opendkim-genkey -b 2048 -d example.com -D /etc/opendkim/keys/example.com -s default -v

sudo chown opendkim:opendkim /etc/opendkim/keys/example.com/default.private

Add public key /etc/opendkim/keys/example.com/default.txt to a TXT record named 'default.\_domainkey' on the email servers domain DNS, delete all whitespace and double quotes:

sudo cat /etc/opendkim/keys/example.com/default.txt

default.\_domainkey IN TXT ( v=DKIM1;h=sha256;k=rsa;p=As+585BPdVcQIDAQABMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAyH9C7FxnsgRSEYFUMEeTudB4IzithCm5Kkylq9+EFGEWguXMH8VZFxm9oE1kTXSIWxHD16QTNnVu6Y8eOLsqt75uve/t8gfy1RvTaKqci04WyEd8lFObzRgmgSPo0uj/O3PF1/juUMwks54wCyPF6RlCahuZzGK3NzGF91a6itL6YLhpAQG1a0GXveOonJbrK+wQLldEool7T17w/iqRMBum3rbCC9akznliyrjweV7/3ywG9zBhG8hI3d2bwUy90HFeJWTkoY7bF9ePZPAxbZWkVt5j9fZDxBs1Y2VObSsQuQDSEFSlbf6mFCrEf4Mu/ExiPV/mA0)

Test DKIM using the following on the email server:

sudo opendkim-testkey -d example.com -s default -vvv

## DKIM - Connect to Postfix

Postfix and opendkim can communicate through through opendkim's unix socket, however since postfix runs in a chroot jail, the default unix socket must be moved to be inside /var/spool/postfix.

Create a new directory for the unix socket:

sudo mkdir /var/spool/postfix/opendkim

sudo chown opendkim:postfix /var/spool/postfix/opendkim

Edit the opendkim configuration file to place its socket in the new directory:

sudo vim /etc/opendkim.conf

Socket local:/var/spool/postfix/opendkim/opendkim.sock

Edit the opendkim defaults file, to move the socket location:

sudo vim /etc/default/opendkim

SOCKET="local:/var/spool/postfix/opendkim/opendkim.sock"

Add the mail filters to the end of the postfix main configuration:

sudo vim /etc/postfix/main.cf

# Milter configuration

milter\_default\_action = accept

milter\_protocol = 6

smtpd\_milters = local:opendkim/opendkim.sock

non\_smtpd\_milters = $smtpd\_milters

Restart the postfix and opendkim services:

sudo systemctl restart opendkim postfix

## DMARC

Domain-based Message Authentication Reporting and Conformance (DMARC) is built ontop of SPF and DKIM, allowing a domain owner to define what happens to mail which fails SPF and DKIM authentication.

The TXT record is named \_dmarc.example.com, and there are various policies which can be added to the record:

* none - dmarc will allow emails to pass results
* quarantine - Emails will be put in quarantine, junk folder
* reject - emails will be rejected in sending process

Emails reports can also be sent onto other addresses and analysed, with there being various services which provide reports for spam. This is provided by adding an rua address, which will then get daily reports. The rf=afrf defines what type of report, and the pct=100 defines the percentage of mail to apply the policy to.

Add a DMARC TXT DNS record to your domain:

v=DMARC1\;p=reject\;rua=postmaster@example.com\;rf=afrf\;pct=100

## Reverse DNS

Reverse DNS allows a reciving email server to check the IP address of the sending server is related to the sending domain.

Reverse DNS records are setup by the IP address owner, therefore since it will be very specific to who is hosting your email server, contact your provider for more information.

# Testing

Local

Load up telnet:

telnet localhost 25

create new message:

ehlo <domain-name>

mail from: root@<domain-name>

rcpt to: contact@<domain-name>

data

<add message>

And then send the message with:

.

The message should now in the /home/contact/Maildir/new inbox and can be read using the less command.

Live

Open up the firewall ports, on the firewall to allow outside connections to the email server:

* 25 (SMTP - MTA Access)
* 587 (SMTP Secure)
* 993 (IMAP Secure)

Then test the server using SMTP Diagnostics online:

https://mxtoolbox.com/diagnostic.aspx

Test deliverability with:

<https://www.mail-tester.com/test-ue6pv6xtl>

<https://glockapps.com/?AFFILIATE=122991&__c=1>