MuntsOS Embedded Linux

Application Note #16: Sending Email

Revision 5
25 March 2025

by Philip Munts
***dba* Munts Technologies**
http://tech.munts.com

Introduction

This application note describes how **send** email messages from a ***MuntsOS Embedded Linux*** (hereafter just **MuntsOS**) target computer. How to **receive** email messages is beyond the scope of this application note.

Prerequisites

MuntsOS must be installed on the target computer (AppNote #3 or AppNote #15).

Test Platform Hardware

The test platform for the purposes of this application note consists of any **MuntsOS** target board with Internet connectivity.

Mail Transfer Agent (MTA)

A ***Mail Transfer Agent*** is a system program that accepts an email message text conforming to RFC5322 and then dispatches it to one or more recipients.

Examples of full service MTA's include Sendmail and Postfix. Full service MTA's are often large and difficult to configure and may not suitable for small embedded systems.

Examples of lightweight MTA's more suitable for small embedded systems include DragonFly Mail Agent, and E-MailRelay.

By tradition every MTA for Unix systems provides a command line program named **`/usr/sbin/sendmail`** that reads an email message text conforming to RFC5322 from standard input and then dispatches the message to its recipient(s).

Given a file **`message.txt`** containing an email message text conforming to RFC5322, the following command sends an email:

```
/usr/sbin/sendmail -t <message.txt
```

The **`sendmail`** command included in **MuntsOS** is provided by BusyBox. It is not really an MTA *per se*, as it just dispatches emails to an SMTP (***Simple Mail Transfer Protocol***) server listening on **`localhost:25`** that must do all the work of dispatching emails to recipients.

MuntsOS does not include an SMTP server, so in order to send email from a **MuntsOS** target computer, you must configure and/or install either an MTA or an SMTP server or both. Three options are described later in this document.

Junk mail countermeasures have made it increasingly difficult for an Internet of Things device like a **MuntsOS** target computer to send email successfully, even with a robust MTA. You will most likely need to dispatch emails to an authenticating intermediate SMTP relay (*aka* smart host) instead of directly to the recipient's domain SMTP server(s).

Mail User Agent (MUA)

A ***Mail User Agent*** is a user program that generates an email message text conforming to RFC5322 and then passes it to an MTA for dispatching. On Unix and Linux, the MUA often runs **`/usr/sbin/sendmail`** using the **`popen()`** Standard I/O Library function and writes the generated email message text to the file handle returned by **`popen()`**.

MuntsOS includes the MUA **`/usr/bin/mail`** from the GNU Mailutils package. The **`mail`** program reads a message payload from standard input, generates an envelope conforming to RFC5322, and then passes the result to **`/usr/sbin/sendmail`** for dispatching.

The following command sends an email with the subject **`Test1`** to recipient **`you@me.com`**:

```
echo "This is a test" | mail -s "Test1" you@me.com
```

An application program can run the **`mail`** command with **`popen()`** to originate an email message.

It is recommended that all **MuntsOS** shell scripts and application programs use the **`mail`** program for originating emails.

The **`mail`** command constructs the sender address from the user name and the host name. This can be overridden couple of ways, if the MTA doesn't already override it. This is fraught with peril: Depending on what junk mail countermeasures have been implemented on the SMTP relay, overriding the sender address may be required, forbidden, or limited to certain domains.

You can add **`-r <sender>`** to the **`mail`** command line:

```
echo "This is a test" | mail -r me@you.com -s Test1 you@me.com
```

Or you can create a file call **`.mail`** in each user's home directory containing the following:

```
address {
  email-addr me@you.com;
};
```

Do not include a display name. The **`mail`** command does not process the sender display name correctly.

An application program can also act as its *own* MUA, dispatching RFC5322 conforming email message texts to **`/usr/sbin/sendmail`** via **`popen()`** or to an SMTP server via TCP at **`localhost:25`**. The Ada Web Server library provides email client services for the Ada programming language. The System.Net.Mail namespace provides email client services for .Net Core programs.

*Tip: The Linux Simple I/O Library provides an Ada package **`Email_Mail`** and a .Net class **`IO.Objects.Email.Mail.Relay`** for originating an email using the **`mail`** command. These are less flexible but considerably easier to use than either the Ada Web Server library or the **`System.Net.Mail`** namespace.*

E-MailRelay

The `emailrelay` extension package provides an SMTP server at `localhost:25`. It does not replace `/usr/sbin/sendmail`.

E-MailRelay must be configured by editing two configuration files:

`/usr/local/etc/emailrelay/auth.conf` -- Replace `username` and `password` with the login credentials for your SMTP relay.

`/usr/local/etc/emailrelay/emailrelay.conf` -- Replace "`servername:port`" with your SMTP relay settings.

DragonFly Mail Agent

The `dma` extension package replaces `/usr/sbin/sendmail`. It does not provide an SMTP server at `localhost:25`, making it somewhat less flexible than `emailrelay`.

DragonFly Mail Agent must be configured by editing two configuration files:

`/usr/local/etc/dma/auth.conf` -- Replace `username`, `smart host`, and `password` with the login credentials for your SMTP relay.

`/usr/local/etc/dma/dma.conf` -- Replace the values for `SMARTHOST`, `PORT`, and `MASQUERADE` with values specific to your SMTP relay settings.

On Demand SSH Tunnel to a Remote Computer

You can configure a **MuntsOS** target computer to SSH `mailtunnel@foo.bar` whenever `sendmail` or another process opens a TCP connection to `localhost:25`.

If you have administrative access to some other Unix (FreeBSD, OpenBSD, Linux, etc.) computer `foo.bar` that runs an SMTP server listening on *its* `localhost:25`, that is permitted to send email, and that you can log into with `ssh`, then you can create a user `mailtunnel` on `foo.bar` that connects to `foo.bar localhost:25` whenever you SSH `mailtunnel@foo.bar`.

After the configuration described above have been made, when `sendmail` on the target computer connects to `localhost:25`, data is invisibly piped to and from `foo.bar localhost:25` by way of a temporary SSH tunnel.

Target Computer Setup

1. Create or modify `/etc/inetd.conf` and `/root/.ssh/known_hosts` with the following commands, replacing `foo.bar` with the domain name of your remote computer:

```
cat <<EOD >>/etc/inetd.conf
# Mail relay over SSH tunnel
127.0.0.1:25 stream tcp nowait root /usr/bin/ssh -q -T mailtunnel@foo.bar
EOD

ssh-keyscan foo.bar >>/root/.ssh/known_hosts
```

2. Create `/root/.ssh/id_rsa` and `/root/.ssh/id_rsa.pub` using `sysconfig` option `Regenerate superuser id_rsa`.

3. Enable `inetd` by setting bit 10 in the `OPTIONS` word in `/boot/cmdline.txt` (Raspberry Pi) or `/boot.config.txt` (Orange Pi Zero 2W or BeaglePlay) using `sysconfig` option `Edit cmdline.txt` or `Edit config.txt`.

4. Copy `.ssh/id_rsa.pub` to the remote computer superuser and then reboot.

Tip: You can build a custom `mailtunnel` *extension package to encapsulate all of this target computer goop. This is especially handy if you are setting up more than one target computer.*

Remote Computer Setup

1. Modify `/etc/ssh/sshd_config`:

```
sudo su -
cat <<EOD >>/etc/ssh/sshd_config

Match User mailtunnel
        AllowTcpForwarding no
        ForceCommand ncat -4 127.0.0.1 25
EOD

killall -HUP /usr/sbin/sshd
```

2. Create user `mailtunnel`:

```
groupadd -g 555 mailtunnel
useradd -c "Mail Tunnel" -m -g 555 -u 555 -s /bin/sh mailtunnel
rm -rf /home/mailtunnel/.*
mkdir -p /home/mailtunnel/.ssh
cat id_rsa.pub >>/home/mailtunnel/.ssh/authorized_keys
chown -R mailtunnel:mailtunnel /home/mailtunnel
chmod 444 /home/mailtunnel/.ssh/authorized_keys
chmod 500 /home/mailtunnel/.ssh
chmod 500 /home/mailtunnel
```

Testing

1. Try to log in from the **MuntsOS** target computer to the remote computer:

```
ssh mailtunnel@foo.bar
quit
```

You should get responses similar to the following:

```
220 bethel.munts.net ESMTP OpenSMTPD
221 2.0.0 Bye
```

2. Try to send yourself an email with a command similar to the following:

```
echo "This is a test" | mail -s Test1 me@you.com
```

Best Practices

Email is not a guaranteed delivery service. Messages can be dropped anywhere along the delivery path for a variety of reasons, with or without any notification. Furthermore, as junk mail countermeasures are implemented by email providers, formerly working email setups can fail without warning. Because of the lack of guaranteed delivery, in the context of an embedded system, email is more appropriate for non-critical notifications.

Email can be incredibly useful for embedded system notifications, provided one can accept the reality of messages being dropped occasionally. Some cell phone carriers even provide a bridge from email to SMS, allowing your embedded system to email a notification directly to your cell phone.

A Brief Description of how Internet Mail Works

Every Internet domain that can accept incoming emails must publish a list of one or more SMTP servers that incoming emails can be dispatched to. Each SMTP server is registered with an `MX` record in the domain's DNS (Domain Name System) servers.

Formerly any computer on the Internet, specifically including a **MuntsOS** target computer, was able to deliver a message directly to a recipient's domain SMTP server(s), by performing a DNS `MX` record lookup, and then walking the list of `MX` records and trying to delivery the message to each recipient domain SMTP server. In short, a computer from which an email originated (*e.g.* a **MuntsOS** target computer) made a TCP connection to a recipient domain SMTP server published in a DNS `MX` record and passed on the message in just one hop. This was how Internet email was originally designed to operate.

Unfortunately, in reason years, service providers such as Gmail have implemented admittedly necessary junk mail countermeasures that have broken that original design. Now recipient domain SMTP servers such as `smtp.gmail.com` refuse connections from SMTP clients that cannot be authenticated. This prevents junk mail generators from impersonating legitimate email service providers.

Part of the authentication process is to perform both forward and reverse DNS lookups on the domain name that an SMTP client introduces itself as with the `HELO` or `EHLO` command. Any client without matching forward and reverse DNS records or an IP address that doesn't match what was reported by `HELO` or `EHLO` is rejected. This will necessarily and by design exclude any client behind a NAT (Network Address Translation) firewall, and any client with a dynamically assigned IP address that cannot also register a dynamic reverse DNS address as DHCP (Dynamic Host Configuration Protocol) servers that assign IP addresses to Internet nodes seldom have the ability to register IP address changes to DNS servers.

Therefore reverse DNS lookup authentication limits the set of eligible SMTP clients to those with a fixed IP address *not* behind a firewall *and* with matching forward *and* reverse DNS lookups. But there are still more hurdles.

Email service providers now require each domain wishing to connect to *e.g.* `smtp.gmail.com` to publish a white list of computers that are authorized to dispatch emails from that domain, in yet another kind of DNS record, the `TXT` record.

As a concrete example, to enable my OpenBSD dedicated server `bethel.munts.net` with IP address 64.156.192.118 to send an email to my Gmail account, I had to edit a `TXT` record for the `munts.net` domain at GoDaddy, the domain name registrar for the `munts.net` domain, changing `v=spf1 include:secureserver.net -all` (GoDaddy's email service now outsourced to Microsoft) to `v=spf1 include:secureserver.net ip4:64.156.192.118 -all`. (GoDaddy's email service *plus* IP address 64.156.192.118). This SPF (Sender Policy Framework) record announces to the world that valid emails from the `munts.net` domain can *only* come from `secureserver.net` or IP address 64.156.192.118.

It is likely that junk mail countermeasures will continue to squeeze out privately administered SMTP relays and that email service providers will eventually only accept incoming SMTP connections to TCP port 25 from other recognized email service providers.

For all practical purposes, end point computers such as **MuntsOS** target computers can no longer originate emails and deliver them to recipient domain SMTP servers. Instead, end point computers must pass emails to an intermediate SMTP relay *aka* Smart Host, either a privately administered Unix computer such as `bethel.munts.net` or one administered by your email service provider, such as `smtp.gmail.com` or `smtp.office365.com`, *etc*.

Mail service provider SMTP servers such as `smtp.gmail.com` now have two functions: They accept email account specific, encrypted, password or `OAuth2` authenticated, connections from MUA's (such as Thunderbird) on TCP ports 465 or 587, or connections from recognized mail service provider SMTP relays on port 25.

The `emailrelay` and `dma` MTA extension package configuration examples described earlier in this document require a user name, password, and SMTP relay domain name specific to a single email account. The email provider may or may not accept email messages with a different sender address.

Methodology #1 -- Shared Email Account for all MuntsOS Target Computers

1. Create an email account like `muntsos.mydomain@gmail.com`. For Gmail and other mail service providers that require `OAuth2` or other two factor authentication, you will also need to create an application password after your email account has been created.

2. Install one of the `emailrelay` or `dma` MTA extension packages on each **MuntsOS** target computer.

3. Edit the MTA configuration files in `/usr/local/etc` on each **MuntsOS** target computer, to plug in the user name (usually the email address), password, SMTP relay domain name, and TCP port number (usually 587).

Advantage: Very quick and easy to set up. You might even be able to use your existing email account.

*Disadvantage: Emails from all of your **MuntsOS** Target Computers will appear to be from the same sender. You will need to include information in each message body about what computer the message originated from.*

Methodology #2 -- Separate Email Account for each MuntsOS Target Computer

1. Create an email account like `mytarget.muntsos.mydomain@gmail.com` for each **MuntsOS** target computer. For Gmail and other mail service providers that require `OAuth2` or other two factor authentication, you will also need to create an application password after your email account has been created.

2. Install one of the `emailrelay` or `dma` MTA extension packages on each **MuntsOS** target computer.

3. Edit the MTA configuration files in `/usr/local/etc` on each **MuntsOS** target computer, to plug in the user name (usually the email address), password, SMTP relay domain name, and TCP port number (usually 587).

*Advantage: You can immediately tell in your inbox which **MuntsOS** target computer a notification email came from.*

Disadvantage: You have to create and administer many email accounts. Some free email service providers, e.g. Gmail, limit the number of accounts you can create.

Methodology #3 -- Fictitious Email Account for each MuntsOS Target Computer

*Warning: This is likely only possible if you have administrative control over your SMTP relay **and** its DNS records, such as `bethel.munts.net` described earlier in this document.*

1. Install the `mailtunnel` (or *possibly* `emailrelay` or `dma`) MTA extension package on each **MuntsOS** target computer.

2. For `emailrelay` or `dma`, edit the MTA configuration files in `/usr/local/etc` on each **MuntsOS** target computer, to plug in the user name (usually the email address), password, SMTP relay domain name, and TCP port number (usually 587).

3. If required by your email service provider, create a file `/root/.mail` on each **MuntsOS** target computer, containing something like the following:

```
address {
  email-addr mytarget@mydomain.com;
};
```

This instructs the `mail` program to use `mytarget@mydomain.com` for the sender address instead of something like `root@mytarget.mydomain.com`. If you use a different MUA or connect to `localhost:25` to dispatch emails, you will have to specify the sender address some other way.

Some mail service providers attempt to validate the sender domain name even for messages arriving from an authenticated SMTP relay. As a concrete example, using `bethel.munts.net` as an SMTP relay, Gmail refuses messages from `root@tarsus.munts.net` but accepts messages from `tarsus@munts.net`.

4. If your email service provider allows it, you should create an email alias or redirection for each fictitious sender address that diverts incoming emails to your regular email account. This will enable you to receive delivery error messages or other replies.

*Advantage: You don't need to create email accounts for any of your **MuntsOS** target computers. You can immediately tell in your inbox which **MuntsOS** target computer a notification email came from.*

*Disadvantage: If your email provider SMTP relay blocks messages with a sender address different from which you authenticated with, you **have** to use a privately administered SMTP relay.* `.mail` *only works if you use the* `mail` *command to create emails.*

Tip: The Linux Simple I/O Library provides an Ada package `Email_Mail` *and a .Net class* `IO.Objects.Email.Mail.Relay` *for originating an email using the* `mail` *command.*