Mail Relay

mx.domain.tld

Server Operations Guide

Section 1: OS Fundamentals

I. Design and Organization

A. Hardware and Operating System

The Mail Relay is a <hardware> with <hardware> CPU and <hardware> of RAM; it has a <hardware> providing about <hardware> of total storage. Novell SUSE Linux Enterprise Server (SLES) v9 has been installed, with Support Pack 1 (SP1). As of this writing, the Linux kernel is **v2.6.5-7.139**.

Linux distributions, such as SUSE, do not reckon their versions in the ways we traditionally associate with NetWare and Windows (*e.g.* NetWare v6.5, NT v4.0). Any Linux distribution (SUSE, Red Hat, Gentoo, Debian, *et. al.*) consists of the Linux kernel, added to which are added any number of packages that the distributor (Novell, Red Hat, *et. al.*) wishes to include in their distribution. So while Novell might call it "SLES 9 SP1", that is really a marketing term. From the system management perspective, it's a **v2.6.5-7.139** kernel (although that it generally shortened to **v2.6.5**).

The Mail Relay was installed with a minimum number of add-on packages. These are discussed below, but in general, the Mail Relay was stripped down as far as possible. It is designed to support 4 basic services: Secure Shell (SSH), anti-virus scanning (ClamAV), E-Mail analysis and modification (MIMEDefang), and E-Mail routing (sendmail). For both security and ease-of-maintenance, virtually every other service is either disabled or not installed at all. This is discussed more in **Run Level and Boot Sequence** below.

B. Devices

Linux refers to system components as **devices**. A **device** can be a disk, processor, memory, keyboard, *etc*. Devices are generally referenced in a tree/directory structure, starting with **/dev**. For example, the mouse device might be **/dev/mouse**. The Mail Relay, in specific, refers to the <hardware> as **/dev/sda**, where **sda** refers to **S**CSI **D**isk **A**. If the <hardware> had been configured to show two logical disks instead of one, then the second disk would be **/dev/sdb**.

C. Partitions and Filesystems

As with other operating systems, the available storage is partitioned, and then the OS is installed and runs on a filesystem. For our NetWare servers, there are DOS/FAT and NetWare partitions, and typically Novell Storage Services (**NSS**) filesystems; for the Windows boxes, the partition (DOS/FAT or NTFS) defines the filesystem, as there are no options to use alternate filesystems. The Mail Relay server uses the Linux **ext3** filesystem, a common journaled filesystem type in the Linux world.

Note that Linux, like all ***NIX**es (***NIX** is term encompassing the various UNIX platforms as well as Linux), specifies filesystem paths using the forward slash, **/**, rather than the backslash. It is important to remember this. For example, the command **cd \etc\mail** will not work; it must be entered as **cd /etc/mail**. Paths in commands and configuration files must be entered using the forward slash. The Windows-specific UNC notation (*e.g.* \\server\file\path) is not supported (except through SAMBA, which the Mail Relay does not use).

The various filesystems are on several partitions, also called **slices**. A storage device (like a RAID array, which appears to the OS as one disk) is divided into partitions (slices) in much the same way an NSS partition is divided into Volumes. There is no "typical" or "standard" way for storage to be divided – its done based on the needs of the system. In case of the Mail Relay, a very simple layout, consisting of three (3) total partitions (slices): **/dev/sda1**, **/dev/sda2** and **/dev/sda3**. All disk space on the relay is allocated to one of these three partitions. The various filesystems are on these partitions:

Device partition	Filesystem
/dev/sda1	**SWAP**

Unlike Windows (or NetWare), Linux does not use a "real" filesystem for its swap disk. Rather, disk space allocated to swap is referred to as **raw** disk. This has the advantage of no filesystem overhead. The OS directly tracks what disk blocks are used for memory swapping. As a result, virtual memory is more efficient than in Windows or NetWare.

On the Mail Relay, the first disk slice (arbitrarily selected – it could have just as easily been the fifth slice) is allocated to swap, and is about <value> in size, or roughly one half of physical RAM. While all *NIX systems require some swap disk, there are no hard rules as to how much is needed. The allocation of <value> is probably a bit generous for the purpose the Mail Relay serves, but guarantees it has room to grow.

Device partition	Filesystem
/dev/sda2	**/**

The **root** filesystem, or **/**, is analogous to the **SYS:** Volume of a NetWare server, or the **C:** of Windows. The OS kernel resides here, as do the device drivers and boot files. All other files and directories reside under the root filesystem, except those filesystems that have their own partition. On the Mail Relay, this filesystem is allocated about <value>.

Since the Mail Relay has only one other filesystem (**/var**, see below), this means that every bit of data not in swap or not on **/var** resides on the root filesystem. This is somewhat unusual, but because the Mail Relay is a limited-access server with a very narrow purpose, there is no need for a more complex disk allocation scheme, such as we might use for a server hosting, say, user home directories.

Device partition	Filesystem
/dev/sda3	**/var**

The **/var** filesystem, which can be thought of as a Volume (other than **SYS:)** on a NetWare server, holds *var*iable system data, such as logs and mail queues. Its contents are generally transitory (for example, E-Mail, which resides only so long as it takes for it to be processed), or generated by the operating system or its components (for example, the various log files).

Since the Mail Relay's primary purpose is to receive, analyze and route E-Mail, this is the most-active filesystem. It has been allocated about <value> of disk space, so even large floods of E-Mail can be queued for processing.

D. grub

grub is short for **Gr**and **U**nified **B**ootloader, and is a popular Linux bootloader. It serves much the same purpose as DOS does for NetWare and Windows – it creates an initial environment where the "real" OS may be loaded. **grub** can be used to load just about any OS that can be booted from the media in question. The **grub** on the Mail Relay is discussed more in **Run levels and Boot Sequence**.

E. initrd

initrd is the Linux component that starts the OS loading process. The closest analogy is what happens in NetWare's **server.exe** as the **STARTUP.NCF** file is executed. The purpose of **initrd** is to load the bare minimum drivers needed for Linux to access the devices it needs to complete the OS boot process, just as **STARTUP.NCF** lists the drivers NetWare needs to access all its basic hardware.

initrd is a static file and should not need to change. Its mentioned specifically because following the application of SP1, the system could no longer detect/use the RAID array. It turned out that the driver selected for the array during install no longer worked after application of SP1. It was necessary to rebuild the **initrd** file to include the proper driver, using the **mkinitrd** program that is included in Linux.

F. Drivers

Linux device drivers are analogous to the **.HAM**, **.CDM** and **.LAN** files in NetWare, and **.SYS** files in DOS/Windows. The primary device drivers used by the Mail Relay are for the <hardware> controller (the <driver>) and for the <hardware> Ethernet NIC (the <driver>). In the NetWare world, these might be named <driver> and <driver>; in *NIX environments, filename extensions (*e.g.* **.SYS**, **.HAM**, *et. al.*) are mostly meaningless (see **G. Files, directories and links** below), so the driver files have no extension.

The Mail Relay loads other drivers, but those two are the most likely to have problems following application of an SP or a hardware change. The drivers are self-contained and self-initializing.

G. Files, directories and links

The Linux filesystems, like their NetWare and Windows counterparts, provide storage of files in a hierarchical directory structure. As noted before, path specifications in the *NIX world use **/** instead of ****. However, that is not the only difference.

*NIX filesystems are case-sensitive. **myprogram** is a different filename from **MyProgram**. **/etc/mail** is a different directory than **/etc/Mail**. Most filenames and directory names are all lower-case, just to make things easier, but this is an important difference to keep in mind. Generally, when you create files and/or directories, use all lower-case.

Generally, filename extensions have no meaning in *NIX. A program can be named **program.txt**, and a text file can be named **message.exe**. Since *NIX filesystems are not bound by DOS/Windows **8.3** naming conventions, names of directories and files may contain multiple periods (**.**) and the number of characters after a period is not limited to 3 – the filename **list.of.special.recipients** is perfectly valid. Like other filesystems, there are reserved characters: **?**, *****, **<**, **>**, **|,** and **/** are all prohibited in filenames.

*NIX filesystems often have links, essentially multiple names for files or directories. The closest analogy with which you may be familiar is the Windows *shortcut*. Links are used to provide shorthand methods of referring to long-winded filenames or directory names, or to provide a pointer to a file that is not in its default location. For example, many of the programs mentioned **Appendix A** are not installed in **/usr/bin**, the traditional place where system programs are stored (similar to **SYS:PUBLIC** in NetWare), but links are provided from **/usr/bin** to their actual location. On the Mail Relay, **/usr/bin/nano** (see **Section 6: Tools** below) is a link to **/opt/nano/bin/nano**.

Moving around the filesystem at the command prompt is very similar to DOS/Windows. The **cd** command is used in the same fashion, *e.g.* **cd /etc/mail**. **cd ..** moves you up one level in the directory, **cd /** moves you to the root of the filesystem, and **cd ~** moves you to your home directory. Remember that there are no drive letters, and to use **/** instead of ****.

II. Run Levels and Boot Sequence

A. Run Levels

Linux, like all *NIXes, has **Run Levels**, which are similar to the **LOADSTAGE**s in NetWare. The normal **Run Level** for the Mail Relay is **3**, which means full multi-user services with

networking support. **Run Level S**, or **0**, is the system maintenance mode, or single-user mode, most often used for applying OS patches. **Run Level 1** is minimal services, **Run Level 2** adds networking support. The other common **Run Level** in the Linux world is **5**, which adds the GUI.

Because the Mail Relay is a server and not a workstation, it has no need of a GUI, and hence stops short of **Run Level 5**. A GUI may still be invoked, in Linux, by entering the **startx** command (similar to NetWare; note, however, that the GUI software was not loaded on the Mail Relay).

By default, the Mail Relay will boot up to **Run Level 3**. This will load everything needed for normal operations.

B. Bootup

The Mail Relay boots much as any other machine through POST. However, when it reaches its bootloader, **grub**, a menu appears. This menu is similar to the one that appears when you install Windows 2K on a DOS system and preserve the DOS partition, and has the same purpose as the menu that appears when we boot a NetWare server.

The top and default selection on the menu is **Linux**, which boots the machine to the SLES v9 installation. After a short delay, this selection is automatically made.

The process is fairly automatic – by default, the system boots to **Run Level 3** and loads all necessary drivers, programs and daemons. The primary control of what gets loaded when are the contents of the **/etc/init.d/rc?.d** directories, including **/etc/init.d/rcS.d**, **/etc/init.d/rc1.d**, **/etc/init.d/rc2.d** and **/etc/init.d/rc3.d**. The files in these directories are usually links (see above) to scripts in **/etc/init.d**. The numbers after **rc** correspond to the **Run Level** with which the scripts are associated, and the name of the link to the script defines what should be done with it when the associated **Run Level** is entered.

For example, when the system enters **Run Level 2**, it examines the **/etc/init.d/rc2.d** directory. On the Mail Relay, it will find a number of filenames, some starting with **K** (for example, **K50clam**) and some starting with **S** (for example, **S06syslog**). **K50clam** is actually a link to the script **/etc/init.d/clam**, while **S06syslog** is a link to **/etc/init.d/syslog**. The leading **S** or **K** tells the system whether to **Start** or **Kill** the service associated with the script. If the name of the link starts with **S**, then the script is called and passed the word **start** as a parameter. If the leading letter is **K**, then the script is called and the word **stop** is passed.

Thus, on entering **Run Level 2**, the Mail Relay will execute **/etc/init.d/rc2.d/K50clam** (actually, **/etc/init.d/clam**) and pass the word **stop** to the script. Similarly, the script **/etc/init.d/rc2.d/S06syslog** (actually, **/etc/init.d/syslog**) will be executed and passed the word **start**.

This system allows one actual script file (for example, **/etc/init.d/syslog**) to be called differently depending on what **Run Level** the system is entering. The 2-digit numbers following the **S** or **K** indicate the relative order in which the particular script should be called. Thus, on the Mail Relay, in **/etc/init.d/rc3.d**, the **S06syslog** script is executed before the **S09sshd** script, when the system enters **Run Level 3**.

The scripts that provide the Mail Relay's services (**ClamAV**, **MIMEDefang**, **sendmail**) are all in **/etc/init.d**, and are all linked for starting in **/etc/init.d/rc3.d**

C. Logging in

Once the system is up and running, you may login at the console using your own user ID.You may also login at the console as **root**.

Login via Secure Shell (SSH, discussed below) is limited to your user ID. The **root** account is not permitted to login remotely.

Normally, you should login using your own user ID, and never as **root**. The **sudo** privilege-escalation tool is provided to allow you to perform privileged operations without having to use the **root** account. The terms "privilege" and "privilege-escalation" are explained in detail below.

III. Security

A. Access to command prompt

System operation is controlled through scripts and configuration files, and there are no provisions for control through a web-based interface such as NetWare Remote Manager or Novell iManager. As noted before, the Mail Relay installation was stripped down to the bare necessities.

Because of this, it is necessary to login to your shell account in order to manage the system. You need to use either an SSH client, or the Virtual Console software for the KVM switch, or go to the server console.

When you login, you'll be presented with a command prompt like this:

xyz@mx:~>

xyz is my user ID. The @ symbol is a separator between my user ID and the hostname, **mx**. The colon (**:**) is a separator between the user/host information and the current working directory (CWD). **~** is a shorthand for my home directory; it could also be expressed **/home/dxb**.

Commands are entered after the **>** of the prompt. When commands are shown in this document,

they will be in ***bold italics***; for example:

xyz@mx:~> ***clear***

where **xyz@mx:~>** is the prompt and ***clear*** is the command.

B. Logging

The Mail Relay performs extensive logging of its activities via the **syslog** (System Log) daemon (**syslogd**). These logs are found in the directory **/var/log**, and are generally named according to the activity they are logging (for example, **sendmail** logs to a file named **mail**).

When troubleshooting, these logs can provide vital clues as to what the problem might be. Because these log files can grow to an enormous size, and because the entries are sequential (that is, the oldest log entries are at the top of the file, while the newest ones are at the bottom), it is often most useful to merely look at the last few log entries. This can be done using the **tail** command, like so:

xyz@mx:~> ***cd /var/log***
xyz@mx:/var/log> ***tail –10l mail***

That command would display the last 10 lines of the **/var/log/mail** file. The letter "l" following the number "10" tells **tail** to display the number of *lines*. The number can be increased or decreased depending on need. You can use the **more** program to pause output:

dxb@mx:/var/log> ***tail –10l mail | more***

where the **pipe**, or |, symbol, is between the filename **mail** and the command ***more***. This command will work on just about any log file in **/var/log**.

C. Privilege and privilege-escalation

Operations in *NIX are generally categorized as either "privileged" or "unprivileged". "privileged" operations require special permissions, while an "unprivileged" operation may be performed by any user.

Some examples of an "unprivileged" operations include accessing files you own or have rights to, starting a program that does not bind to a TCP or UDP port of less than 1025, or killing a process that you started without privilege.

"privileged" operations include starting a program that binds to a TCP or UDP port of less than 1025 (these are the *privileged* ports in *NIX), killing a process owned by another user, or accessing a file you don't own or have rights to.

Because the daemons on the Mail Relay bind to *privileged* ports, you must have **root** authority when starting them up. During boot, the various startup scripts are executed with that authority, so that is not an issue.

Similarly, because you do not own the processes in question, you must have **root** authority to stop or restart the processes. In the past, it was common for people who needed to perform such system administration tasks to have the **root** password, which is much like having the <access> in our environment. Just as there is little limit to what someone with <access> right at the <environment>, there are few restrictions on the **root** account in the *NIX environment.

In order to prevent accidental damage through use of the **root** account and to preserve accountability, the Mail Relay uses the **sudo** tool to provide the necessary access without everyone needing the **root** password.

sudo has a list of commands which users may execute, and the privilege level associated with those commands. You can see the commands available to you using the following command:

xyz@mx:~> ***sudo -l***

sudo will prompt you for your password, which is the same password you used to login. Once you provide it, you will see the commands you may use under **sudo**, and the privilege level with which they will run (typically **root**).

The various commands are discussed below in **Sections 2**, **3**, **4** and **5**.

IV. Daemons

A. What is a daemon?

This document will refer frequently to **daemons**, which may also be called **demons** in other documents. This is a term that has long been used in the *NIX world, and derives from the Greek word for **guardian spirit**, rather than the negative meaning often associated with **demon** in Western cultures. A **daemon** is similar to a **service** in Windows.

One daemon always present on every *NIX system is the **init** daemon. It is essentially the running kernel process and is responsible for creating every other process.

The words **daemon** and **process** are very similar, but not the same. While every **daemon** is a **process**, not every **process** is a **daemon**. A **process** typically executes a single task and exists for a very short time. A **daemon** is a **process** that remains running after completing a task and may complete multiple tasks.

B. System daemons

Because the Mail Relay has been stripped down to the bare necessities, there are very few of the traditional *NIX daemons still running. You can view all running processes with the command:

xyz@mx:~> ***ps -ef***

Some of the "standard", or system, daemons you will see include **kjournald**, the journaled filesystem daemon; **mingetty**, the TTY interface daemon that allows us to login via the console, and **syslogd**, the System Log daemon mentioned above.

C. Daemons that have been added

When you issue the command:

xyz@mx:~> ***ps –ef***

a lot of running processes are displayed, and a lot of information is displayed about them (see **Process IDs** below). There are several daemons that are not a standard part of most *NIX installations, are instead denote daemons that have been specifically added to the Mail Relay. The four that concern us the most are (looking at the right-most column of the information displayed):

sendmail: accepting connections
sendmail: Queue runner@00:10:00 for /var/spool/clientmqueue
/opt/clam/sbin/clam
/opt/clam/bin/freshclam –d
/opt/mimedefang/bin/mimedefang-multiplexor …
/opt/mimedefang/bin/mimedefang …
sshd:

Respectively, these are the:

sendmail Message Transfer Agent (MTA)
sendmail Mail Submission Program (MSP) and queue runner
ClamAV anti-virus scanning daemon
ClamAV signature database update daemon
MIMEDefang Perl process multiplexor
MIMEDefang filter daemon
Secure Shell daemon

Each of these services is discussed in more detail in its respective **Section** below.

D. Process IDs

Every running process in *NIX has a **P**rocess **ID**, or **PID**. This is a number (from 1 to 5 digits) that identifies the process to the operating system. A **PID** is unique in that a given **PID** is never associated with more than one running process at a time.

Process **ID 1** is always the **init** daemon (the kernel).

All processes have a **P**arent **P**rocess **ID** (**PPID**), which is the **PID** of the process that spawned the given process. For many processes, especially ones that are part of the operating system, the **PPID** is **1**, indicating that were spawned by **init**.

E. Controlling daemons

All daemons are controlled using the same startup/shutdown scripts described in **Run Levels and Boot Sequence** above. These scripts are named the same as the daemons they control: **sendmail**, **mimedefang**, **clam** and **sshd**. Note that all the script names are in lowercase, although many references to daemons may include uppercase (*e.g.* MIMEDefang).

While controlling each individual daemon is discussed in detail below, in general, you may manually **start**, **stop**, check **stat**us and, in some cases, **restart** a given daemon (or set of daemons, as some systems, like MIMEDefang, sendmail and ClamAV, consist of multiple daemons) by running the associated script and passing the proper parameter. You should use **sudo** to escalate your privilege level when doing this. For example, the command:

xyz@mx:~> ***sudo /etc/init.d/mimedefang stat***

would check and display the current status of the MIMEDefang system.

F. Sockets

A **socket** is a way for a daemon to communicate with other processes/daemons. Some daemons communicate over a TCP or UDP port: **sendmail** and **sshd** are examples of this, binding to TCP ports 25 and 22, respectively.

A **socket** uses the filesystem to create a special place for communications; similar to how GroupWise once used queue directories for transmitting information (E-Mail, status notifications, administrative traffic) from a Post Office Agent (POA) to a Message Transfer Agent (MTA).

In the case of the Mail Relay, **sendmail** uses a **socket** to communicate with MIMEDefang, MIMEDefang uses **sockets** for its internal communications, and also to communicate with

clamd. All of these uses are documented with the respective package documentation.

From the system-management perspective, **sockets** appear as a special type of directory. Since they are used for inter-process communications, they appear in a **netstat** listing (see **Section 6: Tools** below).

Section 2: The SSH daemon

I. Purpose

Traditionally, access to a *NIX server shell account used **telnet**, a relatively simple client/server communications protocol that grew to be used on a wide variety of systems. A major weakness of telnet, however, is its basic insecurity. It has no practical access control, and no protection against eavesdropping or session hijacking. When you logged into a system, using telnet, your username and password were sent unencrypted. Every keystroke and every server response could be watched, in real-time.

Secure **Sh**ell, or **SSH**, was developed as a replacement for telnet, and is now almost as widely used. It provides an SSL-encrypted tunnel for terminal sessions and other services (for example, Secure FTP). This is the only remote-access protocol available on the Mail Relay server (KVM switch remote access functionality is separate from the Mail Relay).

While Windows comes with a built-in telnet client, you must install an SSH client. Linux workstations include an SSH client.

As of this writing, the Mail Relay runs the OpenSSH server **v4.0p1**.

II. Configuration

The primary configuration file for the OpenSSH daemon (**sshd**) is **/opt/openssh/conf/sshd_config**. This is a text file containing the various configuration directives for the server. It may be edited using **nano** (see **Section 6: Tools**), and is extensively commented. In general, there should be no need to alter this file.

The configuration is read whenever **sshd** starts, and also if **sshd** is restarted. The restart procedure is designed so as to not interrupt existing SSH connections.

The configuration on the Mail Relay only accepts connections using the **SSH v2** protocol, and does not permit **root** or any other system account to login via SSH.

sshd is configured to use **Privilege Separation** (**PrivSep**), and requires the directory **/var/empty** to exist. The control script verifies this directory is present.

III. Logging

The SSH server daemon is configured (via its configuration file) to log to the **authpriv** logging facility. The **syslog** daemon records messages sent to this facility in the file **/var/log/secure.**

After two unsuccessful login attempts, all failures are logged. After the fourth failed attempt, the connection is dropped. Successful logins are also logged.

V. Controlling

The control script is **/etc/init.d/sshd**. This script accepts one of four parameters: **start**, **restart**, **stat** and **stop**. Any other parameter, or a missing parameter, will cause the script to display a very brief help message and then exit without doing anything. The SSH daemon may be stopped, started and restarted independent of any other process.

The functions of each parameter are:

start This starts up the SSH daemon. Because the daemon binds to a privileged TCP port (22, in this case), you must be privileged to **start** the daemon. Using **sudo**, you may manually run the daemon with the following command:

xyz@mx:~> ***sudo /etc/init.d/sshd start***

If the SSH daemon is already running, an error will be displayed and the script will exit without doing anything. If the configuration file is missing, or the **PrivSep** directory does not exist, the script will also exit with an error.

restart A **restart** would be needed if the configuration file was changed, and using **restart** avoids interrupting existing SSH connections. Using **sudo**, you may manually restart the daemon with the following command:

xyz@mx:~> ***sudo /etc/init.d/sshd restart***

The script will exit with an error if it cannot find the configuration file or if the SSH daemon is not running.

stat This parameter instructs the script to gather and display the status of the SSH daemon and associated files. If the daemon is running, its **PID** is shown. The script also makes sure the daemon's configuration file exists (the content is not checked). Even though **stat** does not attempt privileged operations, **root** privileges are needed for you to read the contents of the PID files, so use **sudo**, like so:

xyz@mx:~> ***sudo /etc/init.d/sshd stat***

stop This operation shuts down the SSH daemon. Any existing SSH or SFTP connections are terminated. You must be privileged to perform this operation, so using **sudo**, you would issue the following command:

xyz@mx:~> ***sudo /etc/init.d/sshd stop***

The script will exit with an error if the SSH daemon is not running.

Section 3: The ClamAV System

I. Purpose

ClamAV is a free and open-source anti-virus scanning system. On the Mail Relay, it is used to examine E-Mail attachments, and compare the contents against a signature database. If a virus is detected, ClamAV reports the specific virus.

As of this writing, the Mail Relay is using ClamAV **v0.86.1**.

II. Configuration

The ClamAV system consists of two (2) components: **clamd** and **freshclam**. **clamd** is the ClamAV daemon, the scanning engine that examines attachments and compares them against the signature database. Its configuration file is **/opt/clam/etc/clamd.conf**, a text file with extensive commenting. In general, it should not be necessary to alter this file.

clamd creates and monitors a UNIX Socket **/var/spool/defang/MIMEDefang/clamd.sock**, where MIMEDefang sends E-Mail attachments for scanning. This location exists only when **clamd** is running, and must exist before MIMEDefang can run. Thus, MIMEDefang is dependent upon **clamd**.

The second ClamAV component, **freshclam**, is a daemon that regularly downloads signature database updates. Its configuration file is **/opt/clam/etc/freshclam.conf**, and it is a text file very similar in structure to the **clamd** configuration file. The only reason to alter this file would be for changes to the **HTTP Proxy** credentials it uses to get past the firewall. These parameters are documented in the configuration file.

The **freshclam** daemon is considered part of the ClamAV system, but MIMEDefang does not depend on **freshclam**, and MIMEDefang does not care if **freshclam** is running.

III. Logging

All output from **clamd** is logged to **/var/log/clam**. This will include error messages from problems during startup. When troubleshooting **clamd**, start with this logfile.

All output from **freshclam** is logged to **/var/log/freshclam**. Like **clamd**, this will include startup errors, and should be the starting point for troubleshooting.

IV. Controlling

The control script is **/etc/init.d/clam**. This script accepts one of three parameters: **start**, **stat** and **stop**. Any other parameter, or a missing parameter, will cause the script to display a very brief help message

and then exit without doing anything.

The script operates on both the **clamd** and **freshclam** daemons at the same time. Thus, all operations will affect both daemons – there are no provisions to, for example, stop **clamd** while letting **freshclam** continue to run. That level of control requires manual intervention.

It is important to reiterate that MIMEDefang is dependent on **clamd**. The MIMEDefang startup process will exit with an error if **clamd** is not running.

Also, neither ClamAV daemon can re-read its configuration file. Changes to **clamd.conf** or **freshclam.conf** will require that both daemons be stopped and then started again.

The functions of each parameter are:

start This starts up the ClamAV system, consisting of both the **clamd** and **freshclam** daemons. Because **clamd** creates a UNIX Socket (this is a privileged operation), you must be privileged to **start** the ClamAV system. Using **sudo**, you may manually execute the script with the following command:

xyz@mx:~> ***sudo /etc/init.d/clam start***

If either ClamAV daemon is already running, an error will be displayed and the script will exit without doing anything. If either configuration file is missing, the script will also exit with an error. The running daemon or missing configuration file will be specified. Finally, the script will exit if any MIMEDefang component is running.

stat This parameter instructs the script to gather and display the status of the ClamAV system. For each daemon, if it is running, its **PID** is shown. The script also makes sure the daemon's configuration file exists (the content is not checked). Even though **stat** does not attempt privileged operations, **root** privileges may be needed in order for you to read the contents of the PID files, so use **sudo**, like so:

xyz@mx:~> ***sudo /etc/init.d/clam stat***

stop This operation shuts down the ClamAV system. You must be privileged to perform this operation, so using **sudo**, you would issue the following command:

xyz@mx:~> ***sudo /etc/init.d/clam stop***

The script will exit with an error if either daemon is not running. It will also exit with an error if any component of MIMEDefang is running.

Section 4: The MIMEDefang System

I. Purpose

MIMEDefang is a tool for analyzing and modifying E-Mail "on the fly". It plugs directly into **sendmail** (using the standard MILTER interface), and provides a framework for controlling E-Mail at a level best described as "surgical". Almost any aspect of an E-Mail can be examined or modified.

MIMEDefang is written primarily in Perl, an interpreted scripting language common on *NIX systems. Specifically, on the Mail Relay, MIMEDefang relies on the Perl v5.8.3 installation included in SLES 9. Further, MIMEDefang requires a number of 3rd-party add-on Perl modules. All of these requirements are explained in the MIMEDefang package documentation and are outside the scope of this document.

As of this writing, the Mail Relay is running MIMEDefang **v2.51**.

II. Configuration

The MIMEDefang system consists of 2 components: **mimedefang**, and **mimedefang-multiplexor**. Unlike the ClamAV system, both MIMEDefang components are required to be running at the same time.

MIMEDefang configuration information exists in two places. First, the control script lists a number of command-line variables that are passed to the daemons. These are documented in the script, and additional information is in the MIMEDefang package documentation. In general, these parameters should not need to be changed.

The heart of MIMEDefang is the filter file, **/opt/mimedefang/conf/mimedefang-filter**. It is discussed in detail below.

A "sanity check" script is located in the same directory as the filter file, and is used to check that the filter file is composed of syntactically valid Perl code. It will not catch logic errors, but will make sure that alterations to the filter file use valid Perl. The script may be executed with the following command (and does not require special privileges):

xyz@mx:~> ***/opt/mimedefang/conf/sanity.sh***

The control script also uses the "sanity check" script to validate the filter file prior to many operations. The "sanity check" script will display appropriate messages.

mimedefang-multiplexor monitors a UNIX Socket **/var/spool/defang/MIMEDefang/mimedefang-multiplexor.sock**, for communication with **mimedefang**. This location must exist before either **mimedefang-multiplexor** or **mimedefang** can run.

mimedefang monitors two UNIX Sockets, **/var/spool/defang/MIMEDefang/mimedefang.sock** (where

sendmail places incoming E-Mail for the MIMEDefang system to analyze) and **/var/spool/defang/MIMEDefang/mimedefang-multiplexor.sock** (for communication with **mimedefang-multiplexor**). Both locations must exist before **mimedefang** can run.

III. Logging

The **mimedefang** component of the MIMEDefang system logs to **/var/log/mimedefang**, while the **mimedefang-multiplexor** component logs to **/var/log/mimedefang-multiplexor**. Some messages from the MIMEDefang system will also appear in **/var/log/mail**. In most cases, error messages during startup will appear in one of these three files.

IV. Controlling

The control script is **/etc/init.d/mimedefang**. This script accepts one of four parameters: **start**, **stat**, **restart** and **stop**. Any other parameter, or a missing parameter, will cause the script to display a very brief help message and then exit without doing anything. The MIMEDefang daemons may start only if **clamd** is already running; they may be stopped only if **sendmail** is not running.

The functions of each parameter are:

start This starts up both the MIMEDefang daemons. Because the daemons create UNIX Sockets (a privileged operation), you must be privileged to start the daemons. Using **sudo**, you may manually run the daemons with the following command:

xyz@mx:~> ***sudo /etc/init.d/mimedefang start***

If either MIMEDefang daemon is already running, an error will be displayed and the script will exit without doing anything. If **clamd** is not running, or **sendmail** is running, the script will also exit with an error. The Socket directories (**/var/spool/defang/MIMEDefang** and **/var/spool/defang/MD-Quarantine**) must both exist before the daemons can start, and the script will exit with an error message if they do not. Finally, before attempting to start MIMEDefang, the script will test the filter configuration file – if the file does not exist, or is not sane ("not sane" meaning "contains syntactically invalid Perl"), the script will exit with an error without doing anything else.

restart A **restart** would be needed if the filter file was changed - a **restart** avoids having to shut down **sendmail** (which you would have to do to **stop** and then **start** MIMEDefang). Using **sudo**, you may manually restart the MIMEDefang system with the following command:

xyz@mx:~> ***sudo /etc/init.d/mimedefang restart***

The script will exit with an error if either MIMEDefang daemon is not running, if filter file is missing or not sane, or if **clamd** is not running.

stat This parameter instructs the script to gather and display the status of the MIMEDefang system. For each daemon, if it is running, the **PID** is shown. The script also checks the filter configuration file to insure it is sane. Because MIMEDefang depends on **clamd**, the status of **clamd** is checked – if it is running, its PID is displayed (**freshclam** is not checked); because **sendmail** depends on MIMEDefang, the status of the **sendmail** daemons is checked – if they are running, an appropriate message is displayed (the PIDs are not shown). Even though you are not attempting privileged operations using **stat**, **root** privileges may be needed in order for you to read the contents of the PID files, so use **sudo**, like so:

xyz@mx:~> ***sudo /etc/init.d/mimedefang stat***

stop This operation shuts down the MIMEDefang system. Because **sendmail** depends on MIMEDefang, the script will exit with an error if either **sendmail** daemon is running. You must be privileged to perform this operation, so using **sudo**, you would issue the following command:

xyz@mx:~> ***sudo /etc/init.d/mimedefang stop***

The script will also exit with an error if either MIMEDefang daemon is not running.

V. The MIMEDefang Filter

A. Construction

The filter file is **/opt/mimedefang/conf/mimedefang-filter**, and is actually a Perl program fragment. It defines a number of variables and functions that MIMEDefang uses. It is a complex file, and while it does have some documentation, it should be modified with great care.

The file has been documented in three sections. The first is the variables that MIMEDefang needs to operate. These should not normally change.

The next file section is the **filter*** functions. These are the core of MIMEDefang, and any alterations to the filter file are likely to take place in this grouping of functions.

The third and final file section is called **Other functions** and contains miscellaneous functions called by other functions. A need for altering this section is unlikely.

B. Usage

When MIMEDefang initializes, the filter file is read, compiled by the **mimedefang-multiplexor**, and then made available to the **mimedefang** daemon. If the filter file is altered, changes are not effective until the MIMEDefang system is restarted.

The filter file defines the tests and examinations that are performed against an E-mail (for example, sending it to ClamAV for virus scanning). It also can alter the E-Mail contents, add/delete/change headers, strip MIME parts, *etc*. The MIMEDefang package documentation discusses these operations in depth.

C. Altering

The filter file is composed of simple text and may be altered using **nano**. All alterations must result in valid Perl code.

In general, it should not be necessary to alter the filter file; however, as DOR's needs and system grow and change, the filter file will have to change with them.

D. Testing

The script **/opt/mimedefang/conf/sanity.sh** may be used to verify that the filter file consists of syntactically valid Perl. It does not ensure that the filter logic is appropriate. It is possible to have a filter file pass the sanity check and still not function.

An invalid filter file may be debugged as any other Perl program. A change log at the top of filter file has been established and should be used to document any changes.

Section 5: The sendmail Daemons

I. Purpose

sendmail is a mail router – it does with E-Mail what a Cisco (or other brand) router does with IP packets: receives, examines, and decides how to send to the next "hop" or destination. sendmail is powerful, but complex – on the Mail Relay, we are running a fairly simple configuration using sendmail **v8.13.4**.

The sendmail system consists of two daemons:

MTA - the Message Transfer Agent; despite the name, this is roughly equivalent to a GroupWise GWIA, not a GroupWise MTA
MSP - the Mail Submission Agent (**MSP**), also known as the queue-runner

In a simpler relay environment, only the **MTA** is needed, but MIMEDefang requires the use of the **MSP** as a queue-runner, so both daemons must be running in order for mail to flow.

II. Configuration

A. Process

Configuring **sendmail** involves both configuration files and data tables. The configuration files are **/etc/mail/sendmail.cf** and **/etc/mail/submit.cf** and, while composed of text, are in a format peculiar to **sendmail**. One can learn to read and even modify these files, but much easier methods exist (discussed in **B.** below).

The **.cf** files are read when **sendmail** is started, and **sendmail** will re-read them if it is restarted. These files must exist in order for **sendmail** to start.

sendmail also relies on various data tables in **/etc/mail**. Some are required for **sendmail** to start, some are not. All are read when **sendmail** starts; some may be changed while **sendmail** is running, others require **sendmail** to be restarted when they are changed. These are all discussed in **C.** through **G.** below.

B. .mc *vs.* .cf

The **sendmail** configuration files are built from **macro** files stored in the **/work/sendmail/V8.13.4/cf/cf** (no, that is not a typo) directory. **sendmail.mc** is used to build **sendmail.cf**, and **submit.mc** is used to build **submit.cf**.

The **.mc** files are plain text files and are designed to be read and modified by humans; they

contain extensive commenting. To change the configuration parameters in the **.cf** files, the corresponding .**mc** file is edited, and then the **.cf** file is built using the command (this example is for **sendmail.cf**):

xyz@mx:/work/sendmail/V8.13.4/cf/cf> ***make sendmail.cf***

This command must be executed in the **/work/sendmail/V8.13.4/cf/cf** directory. The resultant **.cf** file must then copied to **/etc/mail** (overwriting the existing file) and **sendmail** started (or restarted).

It would be rare to need to modify the **.mc** files or rebuild the **.cf** files.

C. Access Map

The access map data table defines for **sendmail** limitations on what other hosts may connect to it. This table is built from the text file **/etc/mail/access**, and consists of **/etc/mail/access.db**. The text file is commented; the **.db** file cannot be directly edited.

It may be necessary to modify this table when IP addresses of hosts using the Mail Relay change. To change what hosts **sendmail** allows to connect to it, modify **/etc/mail/access** to reflect the necessary changes. Then, rebuild **/etc/mail/access.db** using the script for this purpose, **/etc/mail/makeaccess.sh**. While you may edit the text file without special privileges, you need **root** privileges to rewrite the **.db** file, so after editing the text file, use **sudo**, like so:

xyz@mx:/etc/mail> ***sudo /etc/mail/makeaccess.sh***

Once the **.db** file has been rebuilt, the changes are immediately effective for all future connections – existing connections are not affected.

D. Mailertable

The **mailertable** database tells **sendmail** where to route E-Mail, based on Domain Name, similar to a routing table for a packet router. The table is built from the text file **/etc/mail/mailertable**, and consists of **/etc/mail/mailertable.db**. The text file is commented; the **.db** file cannot be directly edited.

It should not be necessary to change this table, but if it is, modify the text file to reflect the necessary changes, then rebuild **/etc/mail/mailertable.db** using the script for this purpose, **/etc/mail/makemailer.sh**. While you may edit the text file without special privileges, you need **root** privileges to rewrite the **.db** file, so after editing the text file, use **sudo**, like so:

xyb@mx:/etc/mail> ***sudo /etc/mail/makemailer.sh***

Once the **.db** file has been rebuilt, the changes are immediately effective.

E. Aliases

Using the **aliases** data table, multiple E-Mail addresses may exist for a single recipient. This is similar to the **Nickname** feature in GroupWise. Thus, **postmaster@domain.tld** may be directed to **david.bank@domain.tld**. This only re-directs inbound E-Mail, and the headers are not changed. The table is built from the text file **/etc/mail/aliases**, and consists of **/etc/mail/aliases.db**. The text file is commented; the **.db** file cannot be directly edited.

It should not be necessary to change this table, but if it is, modify the text file to reflect the necessary changes, then rebuild **/etc/mail/aliases.db** using the **newliases** program. While you may edit the text file without special privileges, you need **root** privileges to invoke **newaliases**, so after editing the text file, use **sudo**, like so:

xyz@mx:/etc/mail> ***sudo /usr/bin/newaliases***

Once the **.db** file has been rebuilt, the changes are immediately effective.

F. Genericstable

With the **genericstable** database, it is possible to re-write the **From:** address on outbound E-Mail. Using this feature, mail sent from **root@domain.tld** could appear to be from **otheruser@dornc.com**. This is similar to the **Gateway Aliases** feature of GroupWise. The table is built from the text file **/etc/mail/genericstable**, and consists of **/etc/mail/genericstable.db**. The text file is commented; the **.db** file cannot be directly edited.

It should not normally by necessary to change this data table. If it is, modify the text file to reflect the necessary changes, then rebuild **/etc/mail/genericstable.db** using the script for this purpose, **/etc/mail/makegenerics.sh**. While you may edit the text file without special privileges, you need **root** privileges to rewrite the **.db** file, so after editing the text file, use **sudo**, like so:

xyz@mx:/etc/mail> ***sudo /etc/mail/makegenerics.sh***

Once the **.db** file has been rebuilt, the changes are immediately effective.

This functionality relies on the file **/etc/mail/generic-domains**, which is a simple text file that lists one Domain Name per line. A Domain Name must be listed in order for the **genericstable** functionality to work on E-mail addresses from that Domain – if a Domain is not listed, any entries for it in the **genericstable** database are ignored. This file is only read when **sendmail** starts, and changes to it are not effective unless **sendmail** is restarted.

G. Other files

Other data tables used by sendmail include (all are located in **/etc/mail**):

trusted-users – This simple text file lists all user IDs that **sendmail** considers "trusted", or able to have privileged interactions to it (similar to **Trusted Apps** in GroupWise). It should not be necessary to edit this file. It is only read when **sendmail** starts, and if it is changed, **sendmail** must be restarted in order for the changes to be effective.

domaintable - This data table is built from the text file **domaintable** and consists of **domaintable.db**. It is used for Domain Name migrations, and has been included for completeness. It should not be necessary to edit the text file or rebuild the database.

local-host-names - A plain text file that tells **sendmail** what hostnames it should consider "local", or that are hosted on the machine on which **sendmail** runs. Since **sendmail** is running as a Mail Relay, it has no local mailboxes. This file should always be empty.

virtuser.domains, virtusertable virtusertable.db - These files comprise the virtual user data table, used by **sendmail** to provide E-Mail address translation between Domain Names in a manner more sophisticated than is possible with **aliases**. A Domain Name must be listed in **virtuser.domains** (which is a simple text file, one Domain Name per line) before **sendmail** will pay any attention to its entries in **virtusertable.db**. The database is built from the text file **virtusertable**. This functionality is not really used at this time, but may be employed as needs grow.

H. Testing

It is possible to test **sendmail**'s configuration. Naturally, when **sendmail** starts, it checks its configuration file for validity, and makes sure all the data tables and other files it expects are present and readable.

However, **sendmail** may be running normally and not properly resolve E-Mail addresses or route E-Mail. To test it, **sendmail** may be invoked in address test mode, an interactive mode that lets

you feed it E-Mail addresses and it displays how it would route that E-Mail.

You must be privileged to invoke **sendmail** in address test mode, and a number of command-line parameters are needed, so use **sudo**, as in this example:

xyz@mx:~> ***sudo /usr/sbin/sendmail –bt –d21.12 –d60 –d38***

When you do this, the following prompt appears:

ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
>

sendmail is now waiting for you to enter the rulesets you want it to use, and the E-mail address you want it to resolve. Use rulesets **3,2,0** for all tests, as in this example:

Enter <ruleset> <address>
> 3,2,0 david.bank@domain.tld

sendmail will resolve the address, and display the delivery method, host name and E-Mail address it will use for delivery. The example above should result in **SMTP**, **mailhost.domain.tld** and **david.bank@domain.tld** expressed in **sendmail**'s "triple" notation. Any address can be tested in this fashion to insure that **sendmail** is properly routing E-Mail. The address test mode may be invoked even when the daemons are running.

You exit from the address test mode with the command **/quit**, and a help screen may be displayed by entering **?**.

III. Logging

The primary logfile for all **sendmail**-related entries is **/var/log/mail**. All **sendmail** error messages and operational messages are logged here.

IV. Controlling

The control script is **/etc/init.d/sendmail**. This script accepts one of four parameters: **start**, **restart**, **stat** and **stop**. Any other parameter, or a missing parameter, will cause the script to display a very brief help message and then exit without doing anything. The MIMEDefang daemons may start only if **clamd** is already running; they may be stopped only if **sendmail** is not running.

The functions of each parameter are:

start This starts up both the **sendmail** daemons. Because the daemons bind to a privileged TCP port (25, in this case) and access UNIX Sockets in restricted directories, you must be privileged to start the daemons. Using **sudo**, you may manually run the daemons with the following command:

xyz@mx:~> ***sudo /etc/init.d/sendmail start***

If either **sendmail** daemon is already running, an error will be displayed and the script will exit without doing anything. If MIMEDefang is not running, or the script cannot locate both **sendmail** configuration files, the script will also exit with an error.

restart A **restart** would be needed if certain configuration data files changed (as discussed in **II. Configuration** above), and using **restart** is generally simpler than completely shutting down **sendmail**. Using **sudo**, you may manually restart both **sendmail** daemons with the following command:

xyz@mx:~> ***sudo /etc/init.d/sendmail restart***

The script will exit with an error if either **sendmail** daemon is not running, or if the script cannot locate the **sendmail** configuration files, or if either MIMEDefang daemon is not running.

stat This parameter instructs the script to gather and display the status of both **sendmail** daemons. If the daemons are running, their **PID**s are shown. The script also checks to see if the **sendmail** configuration files exist (their contents are not checked). Because **sendmail** depends on MIMEDefang, the status of the MIMEDefang system is checked – if it is running, its PIDs are displayed. Even though you are not attempting privileged operations using **stat**, **root** privileges may be needed in order for you to read the contents of the PID files or the configuration files, so use **sudo**, like so:

xyz@mx:~> ***sudo /etc/init.d/sendmail stat***

stop This operation shuts down both **sendmail** daemons. You must be privileged to perform this operation, so using **sudo**, you would issue the following command:

xyz@mx:~> ***sudo /etc/init.d/sendmail stop***

The script will exit with an error if either **sendmail** daemon is not running.

Section 6: Tools

I. man

The online manual pages are accessed using the **man** command, a standard command found on virtually every *NIX implementation. Similar to the **help** command available at the NetWare console prompt, you get information on commands by adding the command name to the invocation, like so:

xyz@mx:~> ***man man***

would display the manual page for the **man** command.

Virtually all of the daemons, programs and files necessary to operate the Mail Relay server are documented to one extent or another in their associated **man** pages (a notable exception is **sendmail**). External documentation will frequently refer you to "**man** pages". Some helpful **man** pages include:

OpenSSH
- **man sshd**
- **man sshd_config**
- **man sftp-server**

ClamAV
- **man clamd**
- **man freshclam**
- **man clamd.conf**
- **man freshclam.conf**

MIMEDefang
- **man mimedefang**
- **man mimedefang-multiplexor**
- **man mimedefang.pl**
- **man mimedefang-filter**
- **man md-mx-ctrl**

nano
- **man nano**
- **man nanorc**

sudo
- **man sudo**
- **man sudoers**
- **man visudo**

System programs and files

man init
man initrd
man mkinitrd
man shutdown
man mount
man fsck
man passwd
man shadow
man syslogd
man syslog.conf
man resolv.conf
man hosts
man perl
man gcc
man make

Usually, a given **man** page will have a list of associated **man** pages. Exploration is encouraged.

II. nano

This is a free and open-source text editor – think of NetWare's **EDIT.NLM**, just a lot better. A whole lot better. SLES v9 shipped with **vim**, an updated version of the venerable **vi** (**VI**sual editor) program that has been in every *NIX since the Big Bang. But even **EDIT.NLM** is better than **vi**. **nano** is better than both **vim** and **EDIT** put together. **nano** is available to all users. It gets its name because its user interface emulates the **pico** editor that is part of the PINE E-mail package.

There is a global configuration file for **nano** at **/opt/nano/etc/nanorc**. All the **nano** files reside in the **/opt/nano** subdirectory structure, and there are links to them in **/usr/bin** and **/usr/share/man**.

You may invoke nano at the command line by simply entering its name, like so:

xyz@mx:~> ***nano***

This will open the editor in a new (blank) file. A helpful key reference is along the bottom of the screen, and a line also displays your current cursor position, total number of bytes in the file, and the byte location you are currently editing.

An existing file may be edited by specifying its name when invoking **nano**, like so:

xyz@mx:~> ***nano /opt/mimedefang/conf/mimedefang-filter***

Online help is displayed with **CONTROL-G**, and **CONTROL-X** exits the program.

III. nslookup

This command tests DNS resolution. Using it will gave a warning that it is "deprecated", which just means its old and no longer maintained. It is best used in "single-query" mode, like so:

xyz@mx:~> ***nslookup domain.tld***

which will tell you that host's IP address is **192.168.1.2**.

IV. passwd

Using this program, you can change your password. User account information is stored in **/etc/passwd**; however, your password itself is stored in **/etc/shadow**, the "shadow" file. The **passwd** program will complain if you choose a "weak" or insecure password.

V. YaST

It stands for **Y**et **a**nother **S**etup **T**ool and is the standard system configuration program for SUSE Linux, much as **NWCONFIG.NLM** is used for NetWare. You must be privileged to do anything more than look at the screens. **YaST** is a not a GUI – it operates much like the C-Worthy interface in NetWare. A GUI version (**YaST2**) is available, but not installed on the Mail Relay.

VI. shutdown

As the name suggests, this program shuts down the server. You must be privileged in order to use it. To initiate a system shutdown immediately, use this command via sudo, like so:

xyz@mx:~> ***sudo /sbin/shutdown –h now***

The system will immediately begin to shut down, but will not power off. Substitute ***–r*** for ***–h*** to cause a reboot after the shutdown. To insure that all mail-related services exit cleanly, you can invoke the various control scripts for **sendmail**, MIMEDefang and ClamAV before initiating the shutdown (this shouldn't be necessary, but won't hurt, either).

VII. sudo

This is a controlled privilege-escalation tool, allowing an unprivileged user account to perform privileged operations. Using **sudo** to control services on the Mail Relay is preferable to individual admins using **su** (the *NIX **s**uper **u**ser command) to gain **root** privileges. Most normal operations can be

accomplished via **sudo** (the name is short for **su do**). It resides in **/opt/sudo**.

If Windows had something like **sudo**, we wouldn't be tempted to stay logged into our admin accounts for everything, we'd just escalate our privilege for certain tasks, like running ConsoleOne.

The configuration file for **sudo** is **/opt/sudo/etc/sudoers**. The **man** pages explain **sudo**'s operation in more detail.

VIII. netstat

This tool displays the ports to which daemons are bound, and also what sockets processes may have open. The man page explains how to interpret its output.

During normal operation, the Mail Relay should always be show a **LISTEN** state on ports 22 (ssh) and 25 (smtp). Additionally, UNIX sockets should exist for **mimedefang.sock**, **mimedefang-multiplexor.sock**, and **clamd.sock**.

Appendix A: Important Software

I. Added software

A number of softwares have been added to the base installation of SLES, or have been upgraded from the versions supplied in SLES. These include:

zlib v1.2.2

This is a data compression library, upon which **OpenSSL**, **OpenSSH** and **ClamAV** depend. SLES 9 shipped with **v1.2.1**, which has a known security bug. The updated **v1.2.2** was installed in **/opt/zlib**.

OpenSSL v0.9.7g

The **OpenSSL** package is a free, open-source implementation of the SSL v2/v3 standards, and is used heavily in the *NIX world, as well as by many SSL-ized functions of NetWare (including NetWare's implementations of SSH and SFTP). The typical Apache webserver relies upon OpenSSL for SSL certificate support. SLES v9 shipped with **v0.9.7d**, and this was upgraded to **v0.9.7g**, installed in **/opt/openssl**.

OpenSSH v4.0p1

The **S**ecure **Sh**ell protocol is a replacement for **telnet** as discussed above. OpenSSH is a popular package to provide SSL-encrypted telnet (Secure Shell, SSH) and FTP (Secure SFTP, SFTP) services (although the NetWare implementation only provides SFTP). SLES v9 shipped with **v3.8p8**, which has some known security issues, and so **v4.0p1** was installed in **/opt/openssh**.

GNU make v3.80

The **make** tool is essential for building software from source, as was done with **zlib**, **OpenSSL** and **OpenSSH**. The older **make** program shipped with SLES v9 was replaced with this version, which was installed in **/opt/make**.

GNU nano v1.2.5

It is installed in **/opt/nano** and provides a well-featured full-screen text editor.

sudo 1.6.8p8

This is a tool for allowing controlled privilege escalation. It resides in **/opt/sudo**.

GNU mp v4.1.4

mp is a math library and toolset, used for heavy-duty number crunching. It was installed in **/opt/mp** for the purpose of testing **OpenSSL**, and is not used for normal operations.

sendmail v8.13.4

This software is older than just about any other software you might care to name, save the original UNIX itself. sendmail is a mail router, and the main component of the Mail Relay. It does with E-Mails what a Cisco router does with IP packets – takes them in, analyzes them, and routes them based on that analysis.

sendmail is complex and is discussed at greater length in **The Sendmail Daemons**.

MIMEDefang v2.51

Known as a **MILTER**, or **M**ail F**ilter**, **MIMEDefang** allows us to dissect and potentially rewrite every E-Mail that passes through the Mail Relay. **MIMEDefang** also provides an interface to the **ClamAV** virus scanner.

MIMEDefang is nearly as complex as sendmail, and is discussed in **The MIMEDefang Daemons**.

ClamAV v0.86.1

The ClamAV package is a free and open-source anti-virus scanning package. It works on the same general principles as any other package, scanning through files looking for viral signatures supplied by a database.

On the Mail Relay, this package functions as a component of **MIMEDefang**; it is discussed in **The ClamAV Daemons**.

Perl Modules

The **MIMEDefang** system depends on a number of add-on modules for Perl, downloaded from **CPAN** (**C**omprehensive **P**erl **A**rchive **N**etwork, sort of a **SourceForge** for Perl developers). These are documented in the **MIMEDefang** documentation. See also **Perl v5.8.3** below.

II. Included Software

The Mail Relay makes use of a few packages that were included in the SLES v9 installation, and which did not need to be upgraded. These include:

Perl v5.8.3

A powerful and complex scripting language, **MIMEDefang** relies on Perl.

gcc v3.3.3

The **G**NU **C**ompiler **C**ollection, this is a very popular software development tool for *NIX. It is found on practically every *NIX system, and was installed on the Mail Relay to facilitate building many of the packages above.

Appendix B: Locations of Important Files and Directories

I. SSH

Configuration files and encryption keys - **/opt/openssh/conf**
Command-line executables - **/opt/openssh/bin**
Daemon executable - **/opt/openssh/sbin**
Control script - **/etc/init.d/sshd**
Log files - **/var/log/secure**

II. ClamAV

Configuration files - **/opt/clam/etc/clamd**, **/opt/clam/etc/freshclam.conf**
Virus signature databases - **/opt/clam/db**
Command-line executables and **freshclam** daemon - **/opt/clam/bin,**
/opt/clam/bin/freshclam
clamd daemon executable - **/opt/clam/sbin/clamd**
Control script - **/etc/init.d/clam**
Log files - **/var/log/clam**, **/var/log/freshclam**

III. MIMEDefang

Filter file - **/opt/mimedefang/conf/mimedefang-filter**
Daemon executables - **/opt/mimedefang/bin/mimedefang**,
/opt/mimedefang/bin/mimedefang-multiplexor
Command-line executables - **/opt/mimedefang/bin**
Control script - **/etc/init.d/mimedefang**
Log files - **/var/log/mimedefang**, **/var/log/mimedefang-multiplexor**

IV. sendmail

Daemon configuration files - **/etc/mail/sendmail.cf**, **/etc/mail/submit.cf**
Other Configuration files - **/etc/mail/*.db** and other files in that directory
Configuration source (macro) files and build script - **/work/sendmail/V8.13.4/cf/cf**
Daemon executable - **/usr/sbin/sendmail**
Command-line executable - **/usr/bin/newaliases**
Data table build scripts - **/etc/mail/make*.sh**
Control script - **/etc/init.d/sendmail**
Log file - **/var/log/mail**