Global Information Assurance Certification Paper

SANS GCUX Program
Unix System Administration

Secure access to mail resources for remote users

Pierre Amoudruz
January 29th, 2004

Version 1.9: securing Unix Step by Step

Abstract

This document describes a solution to provide access to messaging resources for remote users of a company. In others words, how remote users who can connect from anywhere on the Internet can have access to their mailboxes? In addition, as the information exchanged by e-mail might contain some confidential information, an appropriate level of security is required.

A mail server located in the company Internet enclave (Internet demilitarized zone) offer messaging service to remote. The solution includes a robust mail transfer agent Postfix to ensure emission and reception of messages; the mail server Cyrus IMAP manages mailboxes access. These 2 applications provide the basic functionalities of the system.

The identity of the users has to control very closely. We use digital certificate to authenticate users and restrict access to the mail server for only the appropriate users. Without a valid certificate, it will not be possible for a user to simply open a connection to the mail server. The second level of authentication lies on user credentials that are verify against a central repository.

The last step of the architecture resides in ensuring that the data exchanged between the users and the server is securely exchanged. The digital certificates help in setting up an SSL (Secure Socket Layer) tunnel between the 2 machines and a consequence ensuring data integrity and non-disclosure.

What can we do to increase the overall level of security of the system? First of all, the default installation of the operating system Red Hat Linux 9 has to be improve: we are talking here about OS patching, startup service control, host-based firewall… Second, we perform deep analysis of the system on a regular basis, including for example log analysis and crucial file changes.

Table of Content

1 Introduction

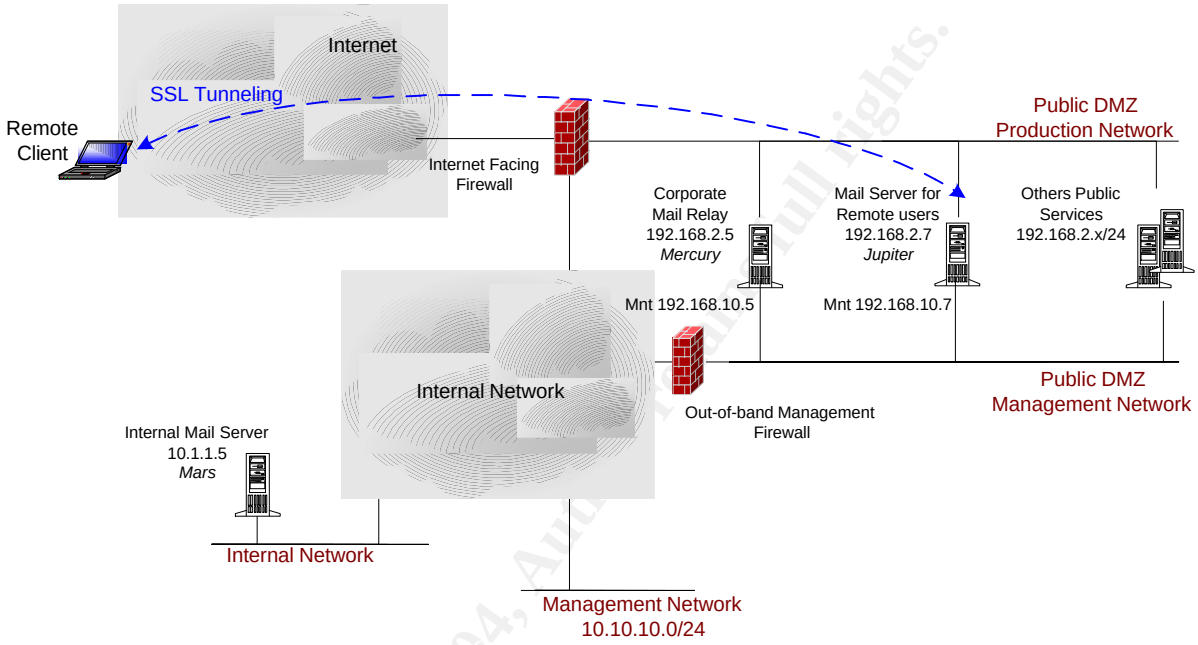
Nowadays, the electronic mail has become a major means of exchanging information between people. For many companies, e-mail is so tied to their day-to-day activities that it can be considered critical to their business. While it is easy for employees located inside the company's private network to access their messaging resources, the situation tends to be much harder for roaming users who might want to access their mail system from different locations.

This document is written as part of the SANS Institute GCUX program. We intend to describe a solution for the issue mentioned above: how to provide remote users with a secure access to their messaging system. We shift from an "out of the box" machine to a system capable of terminating SSL tunnels that gives remote users the ability to send and receive electronic message in a secure way.

This system is setup on a dedicated Intel platform running the RedHat Linux 9 operating system. The steps include the configuration of Postfix for mail delivery, Cyrus IMAP for mail storage and retrieval. The protection of the data being sent over the network is performed using Stunnel to build the SSL tunnel.

2 Description of the system

2.1 Architecture



2.1.1 Overall presentation

The system to set up has to provide access with messaging resources for roaming users with an appropriate level of security. These users are susceptible to connect to their mail server - let's call the mail server for remote users Jupiter - from anywhere on the Internet. They also possibly need to connect to it from the internal network. As a consequence, the server Jupiter needs to reside in the public DMZ where it can be visible from both the Internet and the corporate network. This enclave being separated from the rest of the corporate network by a firewall, the traffic flows at this level can be properly controlled and monitored.

In addition to its production interface that connects to the public DMZ, the mail server requires a second interface dedicated for out-of-the-band management. All administration traffic and supervision should go through this management interface.

The solution detailed in the document lies on 3 main applications: Postfix, Cyrus IMAP and Stunnel. In addition, several specific libraries like OpenSSL or Cyrus SASL are required for the applications to work properly. Finally, a toolkit of security tools like Tripwire or IP Table helps in tightening up the system.

2.1.2 Functionalities

The 2 basics functionalities a mail server provides are emission and reception of electronic messages. The mail server Jupiter should offer the ability for the users to send electronic message using SMTP [1]. After reception of a message, the mail server should either keep it locally for messages destined to local recipients or forward it to the appropriate remote server for messages destined to non-local recipients. All traffic to outside recipients should be forwarded to the corporate mail relay called Mercury. Postfix was selected as a good candidate to perform these tasks.

Then comes mail retrieval. Using a standard mail client, remote users need to connect to Jupiter using either POP or IMAP [2] in order to retrieve their messages. The server has to properly authenticate the user based on the credentials he provides. Upon successful authentication, the server delivers the messages to the user.

2.1.3 Authentication

The protocols POP or IMAP require an authentication mechanism, whereas the SMTP protocol does not natively support nor require authentication. This means that anybody can send a message without the need to prove his identity to the mail server. We would like to avoid this behavior in our architecture and as a consequence, we decided to force the authentication of remote users before allowing them to send a message to the mail server. We use the Cyrus SASL [3] library that offers several authentication mechanisms for protocols like IMAP or SMTP. Cyrus IMAP server comes with build-in SASL support whereas Postfix must be compiled to support SASL. Both use the SASL library to authenticate users against a central database used as a user password repository.

2.1.4 Data protection

As critical data might be sent over an insecure network like the Internet between remote users and our mail server, an SSL tunnel is established as a way to protect the information. Running Stunnel on the mail server to terminate the SSL tunnel at the server side provides some interesting functionalities. First, it ensures a first level of authentication based on the certificate the client is presenting when trying to open a new connection. Second, Stunnel guarantees the confidentiality and the integrity of the data being exchanged using strong cryptography.

As mentioned above, we use a two-factors authentication system. The first level of authentication relies on client certificate when building a new SSL tunnel. The second level of authentication is based on the traditional UserName/Password credentials verification. This is performed at the protocol level (both SMTP or POP/IMAP) using a central credentials repository looked up through SASL.

[1] SMTP : Simple Message Transfert Protocol
[2] POP : Post Office Protocal / IMAP : Internet Message Access Protocol
[3] SASL: Simple Authentication Security Layer

2.2 Hardware description

The first target for this system was around 100 remote users but might be gradually extended upon demand. Sizing the hardware for a mail server can be a difficult task. In our case, we choose an entry-level Dell server like a Dell PowerEdge 600SC with sufficient CPU and memory to offer adequate performance. We include an additional network card along with a Tape Backup Unit for local backup of the machine.

2.3 Software description

The operating system running on the mail server is RedHat Linux 9, which includes the Linux Kernel 2.0.20.

The mail server is running the Postfix Mail Transfer Agent. Postfix was designed to offer an alternative to the old Sendmail program. Unlike Sendmail, Postfix was build from the beginning with security and manageability in mind according to its developer. Postfix is relatively easy to configure. Cyrus IMAP is the implementation of the IMAP standard by Carnegie Mellon University and is recognized to be a robust IMAP server with many successful large deployments.

Finally, we use Stunnel as an SSL wrapper. The new 4.x branch of Stunnel offers text-based configuration method instead of a command line interface like the former and now obsolete (but still supported) 3.x branch. Stunnel does not provide any encryption mechanism by itself but relies on the capabilities of the OpenSSL library for this.

2.4 Details for hardware and software

Hardware	*Description*
CPU	Intel Pentium4 2.4Ghz
RAM	512 MB DDR SDRAM
Hard drive	36GB 10K RPM Ultra 320 SCSI Drive
CD Rom	48X IDE Internal CD ROM Drive
Network card	On board NIC + Intel Pro 100S NIC
Tape Backup Unit	PV100T DDS4 20/40GB Internal TBU
Tape Backup Media	Tape, Media for DDS4, 20/40GB

Application	*Version*	*Description*
Postfix	2.0.16	Mail Transfer Agent
Cyrus IMAP	2.1.14	IMAP/POP server
Stunnel	4.0.4	Certificate-based authentication and SSL tunneling
Cyrus SASL	2.1.16	Library used for authentication purpose
BerbleyDB	4.0	Used for credential storage (RedHat distribution patched)
OpenSSL	0.9.7	Library for encryption functionality (RedHat distribution patch)
OpenSSH	3.5	Remote administration (RedHat distribution patched)

3 Risk Analysis

A risk analysis of the system includes a good understanding of the assets hosted on the system that we need to protect. Once the valuable assets are defined, we need to list the different ways that can be used to access the system. Finally, we should be able to define the risk profile of the system in a production phase.

A proper assessment of the risks to the system at the very beginning of the conception of the infrastructure can help setting up countermeasures and therefore minimize attacks on the system. In our case, we consider the following major risks: physical access abuse, unauthorized remote access, inherent risk to the application being set up, information eavesdropping, mail abuse.

3.1 Asset protection

The need for providing access to a mail server for remote users is principally lead by business reasons, as remote users need to exchange messages with their colleagues and partners for their day-to-day work.

The information being sent by users might include confidential pieces of information related to their activities that should not be disclosed. These data are the main asset of our system.

The repository of users passwords is also a greatly valuable asset. In case of compromision, the authentication of users might be corrupted.

3.2 Access to the server

In addition to the physical access to the server, we need to pay attention to the way the server can be accessed over the network. This means that the services running on the server and offering remote access to the machine should be kept as the only ones necessary.

On its production interface, the mail server is running POP, IMAP and SMTP but should only accept SSL-enabled communication for these mail protocols, meaning POPS, IMAPS and SMTPS. In addition, the mail server need to exchange SMTP traffic with the corporate mail relay for messages between its local recipients and the outside recipients.

The management of the server is performed on its management interface using the protocol SSH.

3.3 Threats description

Physical protection should be the first step when building a properly secured system. Controlling physical access to the machine prevents unauthorized and possibly harmful actions on the system. For example, an intruder with physical access to the machine might reboot the system to access the single-user mode level and then gain root access if no protections are in place. He might also take the hard drive of the server, mount it to an outside machine to obtain the root password and put the hard drive back in place. Physical protection also includes protection against all natural disasters like fire or flood. In order to protect our system against this type of threats, it will be installed in the computer room. Access is restricted to IT team and controlled

by badges with sufficient privileges. The computer room is equipped with adequate fire protection.

The protocols the mail server is running can be used to gain inappropriate remote access on the system. As a consequence, we need to filter the traffic at the host level to only accept mandatory traffic. This brings a second level of traffic filtering as the traffic is properly filtered at the corporate firewall level. In addition, a host-based firewall solution can be of great help in the eventuality of a server located on the public DMZ being compromised, as the corporate firewall would be useless in this case.

The solution we set up includes Cyrus IMAP, Postfix and Stunnel. Even if the POP, IMAP and SMTP ports are not directly accessible from the Internet, we should be using the last version of this software in our infrastructure. Stunnel is used in the termination of the SSL tunnel initiated by remote users, so it needs to be kept up-to-date. As Stunnel relies on the OpenSSL library that has shown some vulnerabilities in the past, this library should also be updated as needed.

The system can encounter fake authentication from users. We mitigate this risk by using 2-factors authentication, one based on digital certificates and the second using login/password validation. The access to secret elements like server private key or user credential database is restricted to system daemons that need to perform operation with them. In addition, these elements are included in the Tripwire list of files that are controlled for integrity on a regular basis.

Information eavesdropping needs to be taken into consideration. E-mail can be used to send confidential information that should not leak outside the company. When sent over the Internet, this type of information may be intercepted or modified. Cryptography is an appropriate response to this threat and using SSL guarantees the integrity and confidentiality of the traffic flows.

The last threat we consider is the threat related to the mail exchange itself and principally includes viruses and spam. These controls are performed on the corporate mail relay in order to reduce the load on the local mail server. This brings security to the exchange between the company users and the external recipients, but nothing to protect exchange inside the company. We mitigate this risk with a strong anti-virus policy in place at the desktop level making it difficult for users to send harmful files. In addition, depending on the performance of the system, the installation of a local anti-virus solution on the local mail server may be considered.

4 Step-by-step guide

4.1 Linux RedHat 9

4.1.1 Pre-requisite

Linux RedHat 9 has been chosen as the operating system for this mail server.

We decided to have the RedHat distribution CDs shipped directly from a retailer to our company. This avoids having to download large distribution files from a RedHat mirror site.

Once we receive the CD sets, it is crucial to be sure that those CDs have not been tampered with or corrupted in any ways. A tool like GPG [4] can greatly help in this task as it enables to verify the checksums and digital signature of the CD sets. The validation of the checksums ensures that the CD sets have not been altered. The next step at this point is to guarantee the validity of the checksums. We have to verify the digital signature of the checksums to be sure that they were effectively signed by somebody that we trust –in this case the RedHat security team. The owner of the key used to provide digital signature of the RedHat distribution is security@redhat.com, the key ID is: GPG#db42a60e. You might need to download these keys at: http://www.redhat.com/solutions/security/news/publickey.html for use with GPG.

During the installation of the operating system, the system is not connected to any other machine. No network cable should be plug-in the system before the appropriate OS patches have been installed on the machine.

4.1.2 Linux RedHat 9 installation

We insert the CD 1/3 of the distribution to the machine and then boot on the CD to start the installation process. We choose the TEXT installation method.

The first screen is a simple welcome screen so we can click on OK. Then we are presented with language, keyboard and mouse selection.

Language selection	English
Keyboard	US English
Mouse	2 Mouse Button PS/2

Once the appropriate options are selected, another welcome screen is displayed and we are prompted for the type of installation we want to perform. As we intend to take full control of the packages that will be installed on the machine, we opted for a “Custom” installation mode.

DiskDruid is used for disk partition. If the disk contains any existing partitions, we are prompted to keep this partition schema or delete it. As we are building a completely fresh system, we do not need to preserve any existing partition, so we can select the “Remove all existing partition” option.

[4] GPG : Gnu Privacy Guard

Perform a new Linux installation	
Installation Type	Custom
Disk Partitioning	Manually partition with DiskDruid
Remove all existing partition	

Defining an appropriate partition schema for the system we are building is a key part of the installation process. We do not select auto-partitioning and define manually the entire partition schema. The partition we create are /, swap, /boot, /usr, /var and /home. This permits to control several options when mounting these partitions and then improve the security of the system. For example, we mount the /usr partition where all the binaries live with the “read-only” mount option to prevent harmful modifications.

First, we set up a root partition with 1024 Mbytes, mounted under /.

The kernel images used for booting are stored on the dedicated /boot partition with 512 Mbytes allocated. This allows to keep several images of the Linux kernel.

We allocate twice the amount of memory, hence a 1024 Mbytes for the swap space.

The /usr partition will contains all the binaries of the application installed on the system.

The /var partition will host user mailboxes and mail queues. It will also store all the log files. This partition requires all the remaining disk space on the hard drive available, around 30GByes.

The /home partition host user home directories and does not require a lot of disk space as only administrative account will be set up on the system. A 1024 Mbytes is appropriate for this use.

All the partitions will be formatted using the ext3 file system and have to be checked for errors.

Mount point	Size(MBytes)°	Partition type
/	1024	Force to be a primary partition
/boot	512	Force to be a primary partition
swap	1024	
/usr	4096	Force to be a primary partition
/home	512	
/var	~30Gbytes (Fill all the remaining space)	

The installation process continues with the boot loader configuration. We select Grub and set a password for the boot loader. This prevents edition of the boot parameters by inadequate people.

Boot loader parameters	
Boot loader	Grub
Boot from /dev/hda2 (partition /boot)	
Use a boot loader password	XXX
Configure advance boot loader option	Choose boot from MBR

It is now time to configure the network settings of the system. This includes IP address, network, gateway, name of the machine and DNS server. We assign a free IP of the public DMZ on the management network.

Network settings	
Manually assigned IP address	192.168.10.7
Netmask	255.255.255.0
Manually assigned hostname	jupiter.example.com
Gateway	192.168.2.1
Primary DNS	192.168.2.10

We will configure the firewalling capabilities later so we do not edit the firewall configuration parameters at that time.

We then select the appropriate language and time zone for the system.

Firewall configuration	No firewall
Additional language support	English US
System clock uses UTC	UTC +02

The root password choice should follow the company standards. The means for example a password with 8 characters, a mix of upper and lower case, numbers and symbol. Once the 2 passwords entered match, the system validates the choice.

Root Password	XXX

Next is the configuration of the authentication features. We can leave the default here enabling shadow and MD5 password. We will not be using NIS, LDAP or Kerberos functionality.

Authentication	Leave default
	Enable MD5
	Enable shadow
	No NIS, LDAP, Kerberos, SMB

The package selection is performed so that we keep the OS relatively small. There is no need to installed packages that will be of no use on the system. A good rule a that point is: “do not to install packages if you are not sure but install it later if you need it”.

We uncheck all packages and then go to the individual package selection. This enables us to select only the required packages for our system.

After package selection is complete, the system checks for dependencies between packages. If a dependency test fails, the system will prompt for the missing

packages and ask to install them or not. If no error appears here, the system starts installing the selected packages. It will ask for CD2 or CD3 if needed.

Category	Section	Off	On
Amusements	Games	All off	
	Graphics	All off	
Applications	Archiving	All off	
	CPAN	All off	
	Communications	All off	
	Databases	All off	
	Editors	All off	
	Engineering	All off	
	File	All off	
	Internet	*Keep*	Ethereal
			Openssh-clients
			Tcpdump
			Telnet
			Wget
	Multimedia	All off	
	Productivity	All off	
	Publishing	All off	Groff
	System	*Keep*	Logwatch, GnuPG
			sudo
			tripwire
	Text	All off	
Development	Debuggers	*Keep*	lsof
			ltrace
			strace
	Languages	*Keep*	Perl
	Libraries	*Keep*	perl-filter
			libpcap
	System	*Keep*	All off
	Tools	*Keep*	All off
Documentation		*Keep*	Man Page
System environment	Base		quota
			crontabs
			devlabel
			eject
			logrotate
			Man
	Daemons	*Keep*	Ntp
			Tcp_wrappers
	Kernel	All off	
	Libraries	*Keep*	libstdc++
			libtool-libs
			compat-libstdc++
			libgtop2

			lipcap
	Shells	All off	
User Interface	Desktops	All off	
	X	All off	
	X hardware support	All off	

It is a good practice to create a boot disk for a recovery purpose. This enables to boot off this disk and perform administrative tasks on a broken system.

We can then reboot the system and log in for the first time to the system.

4.2 Locking down

This part intends to describe the steps to perform in order to improve the default configuration settings of the newly installed system. This includes applying the latest patches to the OS, turning off the services that will not be used, improving the behavior of our network parameters.

4.2.1 OS patching

Following the fresh installation of our system, we need to keep our system up to date with the latest versions of the installed packages.

The latest patches for RedHat 9 systems are available from the RedHat update web site at http://updates.redhat.com/9/en/os/. From there, you should download all available packages in the i386 and i686 directories. We download from the RedHat update web site in a properly up and running dedicated system. Once the download is completed, we have to verify the integrity of the patches that have been downloaded as we did for the distribution files. If this test is successful, we can pursue the patches installation.

We directly connect our server to the dedicated machine where the upgrades resides using a crossover cable. Then we copy all the updated packages to our system. This ensures that our system, which is not properly patched and configured at that time will not be eventually comprised when connected to the network.

At the time of writing, there were several application patches available along with a single kernel patch that we can install on our system. The kernel patch is copied to /var/install/kernel-patch whereas the applications patches are copied to /var/install/app_patch.

We need first to install the kernel patch and then reboot the machine for the changes to take effect. We use the “-i” option when installing the new kernel as we want to keep a copy of the old kernel in case something went wrong during the upgrade.

Then we edit the boot loader configuration file and specify to boot on the newly installed kernel. So we edit the /boot/grub/grub.conf file and check that an entry for the new kernel is created. In order to specify which kernel should be used at boot time, we have to change the parameter called “default” and have it point to the new kernel. In our case, we set default to 1 to boot on the kernel called “title Red Hat Linux (2.4.20-20.9)”.

```
# cd /var/install/kernel-patch
# /bin/rpm –ivh kernel-2.4.20-20.9.i686.rpm

# vi /boot/grub/grub.conf
<…skip…>
default 1
title Red Hat Linux (2.4.20-20.8)
        root (hd0,0)
        kernel /vmlinuz-2.4.20-20.8 ro root=LABEL=/
        initrd /initrd-2.4.20-20.8.img
title Red Hat Linux (2.4.20-20.9)
        root (hd0,0)
        kernel /vmlinuz-2.4.20-20.9 ro root=LABEL=/
        initrd /initrd-2.4.20-20.9.img
<…skip…>

# init 6
```

We only want to install updated versions of our currently installed packages. As a consequence, we will use the –F option of the rpm utility that enables to only install updated versions of the previously installed packages (-F stands for freshen).

```
# cd /var/install/app-patch
# rpm -Fvh *.rpm
```

We now have a system properly patches and we can start working on its configuration.

4.2.2 Restrict boot services

The utility chkconfig shipped with RedHat permits to control services started at boot time. This utility manages the various links between the run level directories –ie /etc/rc[1-6].d and the standard repository of startup scripts –ie /etc/init.d. chkconfig ensures that the service under its control has an appropriate start and kill script in every run entry

As we greatly reduce the number of selected packages during the installation of the mail server, only a few services have been enabled by default during the boot sequence. They should however be disabled using the chkconfig utility. So, we turn off xinetd as there is no need for it to be running on the system. We also turn off kudzu (used for new hardware detection), netfs (used for network file sharing), rawdevices (used to assign raw devices to block devices), pcmcia (as there is no pcmcia devices attached). Only the necessary services will be turned on at boot time including for example the SSH daemon.

We will add later some services under the control of chkconfig. In our case, the startup of the mail services will be controlled using chkconfig..

For the services that we need to turn off, we use the –level option of chkconfig to specify the running level where to turn off the service. We can check the remaining services that are start up at boot time with the –list option.

```
# /sbin/chkconfig --level 2345 xinetd off
```

```
# /sbin/chkconfig --list
kudzu           0:off   1:off   2:off   3:off   4:off   5:off   6:off
syslog          0:off   1:off   2:on    3:on    4:on    5:on    6:off
netfs           0:off   1:off   2:off   3:off   4:off   5:off   6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
random          0:off   1:off   2:on    3:on    4:on    5:on    6:off
rawdevices      0:off   1:off   2:off   3:off   4:off   5:off   6:off
pcmcia          0:off   1:off   2:off   3:off   4:off   5:off   6:off
keytable        0:off   1:off   2:off   3:off   4:off   5:off   6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
xinetd          0:off   1:off   2:off   3:off   4:off   5:off   6:off
ntpd            0:off   1:off   2:off   3:off   4:off   5:off   6:off
iptables        0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

Even if disable the xinetd daemon from automatic startup at boot time, we can go further by completely removing the files and directories tied to xinetd. Consequently, we remove /etc/xinetd.conf and /etc/xinetd.d.

```
# rm -fR /etc/xinetd*
```

4.2.3 Access control

A system can be accessed by several means; one of them is the serial connection. Even if this might be handy, we opt for disabling access to our system through the serial port. This is achieved by removing the adequate entries in the /etc/inittab file, which controls the way the init process setup the system when it enters a run level.

We comment out the tty[2-6] to only keep the tty1 line. This ensures that only the standard system console device is available.

```
# vi /etc/inittab
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
#2:2345:respawn:/sbin/mingetty tty2
#3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
```

By default, RedHat does not prompt for the root password when entering in single-user mode. This prevents an attacker with physical access to the machine to access the single-user mode without knowing the root password. This parameter is also controlled by the /etc/inittab file

```
# vi /etc/inittab
```

```
sum:S:wait:/sbin/sulogin
```

With the view to facilitate the accounting of the system, we would like to control the way the users can enter the system using the root account. We configure the system to only allow access for the root account to the system console device. The file /etc/securetty has to be cleared of all its entries except the tty1. We detail later in the document how to control root login remotely via SSH.

4.2.4 Prevent shutdown

Linux comes with a keyboard sequence CTRL+ALT+DEL that enables to reboot the system. This behavior is unwanted and must be disabled. The /etc/inittab should be modified by commenting out the following line.

```
# vi /etc/inittab
# ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

4.2.5 IP stack

The file /etc/sysctl.conf is used to control the behavior of the IP stack. The contents of this file are parsed at boot time and then set the parameters defined in this file. Below is the output of this file with some comments on the different parameters.

```
# vi /etc/sysctl.conf

# We do not perform routing
net ipv4.ip_forward=0

# Disable Smurf attack. Pinging broadcast/multicast addresses not allowed
net.ipv4.icmp_echo_ignore_broadcasts = 1

#No ping requests against network mapping
net.ipv4.icmp_echo_ignore_all = 1


# Interface configuration
# Our routing is symmetric
# Internal traffic on internal interface; External traffic on external
#interface
net.ipv4.conf.all.rp_filter = 1

# Log Martians packets
net.ipv4.conf.all.log_martians = 1

# Disable source routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.send_redirect = 0
net.ipv4.conf.accept_redirect = 0
net.ipv4.conf.secure_reditect = 0

# Set this parameters as default
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.log_martians = 1
net.ipv4.conf.default.accept_source_route = 0
```

```
net.ipv4.conf.default.send_redirect = 0
net.ipv4.conf.default.accept_redirect = 0
net.ipv4.conf.default.secure_reditect = 0
```

4.2.6 Remove unneeded user and group accounts

User accounts and group accounts are set up by default when installing a new RedHat system. It is recommended to remove the unnecessary accounts in order to make the /etc/passwd or /etc/group files shorter and then facilitates accounting of our system.

```
# for user in pcap ntp games gopher news; do /usr/sbin/userdel $user; done
```

```
for group in XXX; do /usr/sbin/groupdel $group; done
```

4.2.7 Banners

Banners can be configured at several locations on the system. For our mail server, we want a banner to be displayed at the console and when connecting remotely using SSH. So we configure the file /etc/issue with the appropriate corporate banner. In addition, the /etc/motd should also be displayed after a successful logon.

4.3 Core applications setup

4.3.1 Development environment

The applications to install on our server come as a zipped tarball that we need to compile to create the corresponding binaries. However, when installing the base OS of our mail server, we reduce to the minimum necessary packages, which do not include a development environment. Our server does not use these development tools on a daily basis, so there is no reason to install them. In addition, we want to make the life of a potential intruder a (little) bit harder by not installing the tools he may need to install harmful applications like Tojans or backdoors.

We setup a complete development environment on a dedicated development server to compile our applications. Once we get the binaries on the development server, we can easily transfer them to Jupiter, our production server.

The listing of the applications that we need to compile gives the following output. The directory /var/install is used as a base directory for compiling operation. This includes Cyrus SASL, Cyrus IMAP, Postfix and Stunnel. All of these tarballs come with checksums and signatures that have to be process in order to ensure the integrity and validity of the binaries. The tool GPG can be used in the same way as the validation of the Red Hat distribution and patches. This step is not luxury and really worth spending some time on it.

```
$ ls /var/install
cyrus-imapd-2.1.14.tar.gz  postfix-2.0.14.tar.gz   cyrus-sasl-2.1.15.tar.gz
stunnel-4.04.tar.gz
```

In order to perform a painless compiling task, we need to properly indicate to the system where to find the various libraries needed to compile our application. So we add the Berkley DB, Kerberos libraries to the /etc/ld.so.conf file. Then we run the utility ldconfig for the changes to take effect.

Note that the SASL lib has also to be added once it has been compiled. Here is the output of the ld.so.conf

```
#vi /etc/ld.so.conf
/usr/lib
/usr/local/lib
/usr/local/lib/sasl2
/usr/kerberos/lib
```

```
# ldconfig
```

In addition, we need to create several symbolic links to indicate the place for Kerberos headers files. By default the compiler look for them in the /usr/include though they reside in /usr/Kerberos/include, so we create the following appropriate symbolic links for these header files.

```
#ln -s /usr/kerberos/include/krb5.h /usr/include/krb5.h
#ln -s /usr/kerberos/include/com_err.h /usr/include/com_err.h
#ln -s /usr/kerberos/include/profile.h /usr/include/profile.h
```

4.3.2 Pre-requisite

Cyrus SASL and Cyrus IMAP require the use of a database for storage. We use the Berkley database 4.0 shipped with the RedHat 9 distribution as backend storage.

4.3.3 Cyrus SASL

SASL stands for Simple Authentication Security Layer and offers protocols some mechanisms to perform an authentication task. In our case, the SMTP or IMAP protocol uses the authentication mechanism the SASL library offers in order to authenticate users. During the configuration of SASL, we select a PLAIN mechanism, meaning that the server has to authenticate the client based on its username and password. A major point to note is that a PLAIN mechanism should be used in coordination with some protections at the transport layer in order to avoid sending the clear-text version of the user password over the network. SSL is our candidate to perform this task (the configuration of the configuration of SSL is detailed below).

We select the auxiliary plugin called sasldb shipped with the SASL library as our plain mechanism. This mechanism is used to validate the credentials provided by the user against the credentials stored in the database file /var/db/sasldb2. What is really interesting in this architecture is that remote users do not need to have a regular Unix account on the system but only an account on the sasldb2 user credentials database. Only the system administrator of the system requires a local user account on the mail server.

For more details on Cyrus SASL, you might want to have a look at the system administrators guide at: http://asg.web.cmu.edu/cyrus/download/sasl/sysadmin.html.

To install the SASL library, we start by unzipping and extracting the Cyrus SASL archive.

```
$ gzip -d /var/install/cyrus-sasl-2.1.15.tar.gz
$ tar xf /var/install/cyrus-sasl-2.1.15.tar
$ cd /var/install/cyrus-sasl-2.1.15
```

Then we run the configure script with several options in order to control what will be installed. In particular, we disable the authentication mechanisms that will not be used. These ones are for example digestMD5, which is an authentication mechanism based on shared secret keys. As explained above, we only keep the plain mechanism for authentication. The directory for storage of user credentials is /var/db/sasldb2 (option –with-dbpath).

```
$ /var/install/cyrus-sasl-2.1.15/configure \
--disable-krb4 --disable-digest – disable-otp --disable-gssapi \
--disable-anon --disable-cram --disable-digest –disable-ntlm \
--with-dbpath=/var/db/sasldb2

$ make
# make install
```

The SASL libraries is installed in /usr/local/lib/sasl2 and look for plugin in /usr/lib/sasl2. Then we need to link /usr/local/lib/sasl2 to /usr/lib/sasl2. You may want to read the SASL sysadmin guide for more information.

```
# ln -s /usr/local/lib/sasl2/ /usr/lib/sasl2
```

Note: if needed, you need to add /usr/local/lib to /etc/ld.so.conf and run ldconfig as explained above in 3.1.

The access to the database where the users credentials are stored should be properly controlled. As a consequence, we set up the ownership of this directory to root with read and write permissions. This ensures that only root can add new users to the database.

The group ownership is set to the group “mail” with read only permissions. We will add later the users “cyrus” and “postfix” to this group “mail” as they need read access to the database file in order verify user credentials. These accounts “cyrus” and “postfix” are the ones used to run the IMAP and STMP daemons.

```
# chown root:mail /var/db/sasldb2
# chmod 640 /var/db/sasldb2
```

Cyrus SASL comes with several useful administrative tools to perform operations on the sasldb2 credential database. These utilities called saslpasswd2 and sasldblistusers2 are located under /usr/local/sbin. These are administrative tools needed to add users or list the inserted users from the authentication database.

We then create 2 accounts, one for the administrator "admin1" in charge of the database and a second account for the "testmail" user.

```
# /usr/local/sbin/saslpasswd2 –c admin1
Password: <admin1 password>
Again (for verification): <admin1 password>

# saslpasswd2 –c testmail
Password: <testmail user password>
Again (for verification): <testmail user password>

# sasldblistusers2
admin@branch1.example.com: userPassword
testmail@branch1.example.com: userPassword
```

Due to the conception of Cyrus SASL, the passwords inside sasldb2 database are stored in clear text. We take some precautions to set up the appropriate level of security for the system:

- Only users with administrative privileges can log on to the system (root and admin1) and access the credential file.
- An intruder cannot get access to the box using a stolen user password as there are no user accounts defined on the box.
- Set strong permissions on the credentials database file.

However, some security policy won't allow clear text password storage. An alternate solution is to use an LDAP database for backend storage and store hashed version of the password. This brings some modifications in our architecture. First the version 4.1 of the Berkley DB (not shipped with RedHat 9) is required to work with the latest version of OpenLdap 2.1.22. In addition, instead of using the sasldb auxiliary plugin, we need to use a specific OpenLdap auxiliary plugin called ldapdb. This plugin, though working properly, is still in "experimental" mode.

For the rest of the document, we stick with our architecture of sasldb2 database file.

4.3.4 Cyrus IMAP

The installation of Cyrus IMAP server is pretty straightforward. We first unzipped and extract the archive.

```
$ gzip -d /var/install/cyrus-imapd-2.1.14.tar.gz | tar xf cyrus-imapd-
2.1.14.tar
$ cd /var/install/cyrus-imapd-system
```

We run the configure script with several options. First, we would like to use /usr/local/cyrus as the base directory for Cyrus IMAP server binaries. Then we want to

run the server as the user “cyrus”. Finally we disable Sieve support, as we don’t want to use it. Sieve performs controls on messages transiting through the IMAP server.

```
$./configure --with-auth=unix --with-cyrus-prefix=/usr/local/cyrus \
–with-cyrus-user=cyrus --disable-sieve

$ make clean
$ make all

# make install
```

The Cyrus IMAP server binaries are located in the /usr/local/cyrus/bin directory. This directory contains the IMAP and POP3 and master application. (This one is used as a wrapper to launch other Cyrus processes, typically imapd or pop3d).

Others Cyrus utilities that come with the server are installed in /usr/local/bin. These are principally administrative and testing tools like “cyradm” used for mailboxes creation or “imtest” used to test IMAP connection.

The next step is the configuration of our IMAP server.

The first thing to do is the creation of the user “cyrus” under which the cyrus imap daemon runs as. This user does not need to log in to the system; as a consequence we set its home directory to /dev/null and do not provide him with a valid shell access. The primary group “cyrus” for the user “cyrus” is created as well. Note that uid/gid below 500 are typically reserved for RedHat build-in accounts. We choose uid/gid starting with 5555 to fulfill our needs.

In addition, we need to make the user “cyrus” member of the group “mail”. This will ensure that the IMAP daemon running as the user “cyrus” can verify users credentials stored in the sasldb2 database. (the –G option is used to set the “alternative” group for a given user).

```
# /usr/sbin/groupadd –g 55551 cyrus
# /usr/sbin/useradd -u 55551 –g 55551 –d /dev/null -s /bin/NOshell cyrus
# /usr/sbin/usermod –G mail cyrus
```

Configuration file: /etc/imapd.conf

The configuration files of the IMAP server is /etc/imapd.conf. We specify the directory that contains the configuration of the IMAP server and the directory that store user mailboxes (/var/spool/imap). These are to be located on the /var partition,which provide enough disk space. The user “admin1” has administrative rights to configure the IMAP server.

In addition to directory locations, we specify the way the cyrus IMAP server will authenticated its users. The SASL auxiliary plugin (Note the keyword: sasl_pwcheck_method=auxprop stated for “auxiliary”) “sasldb” is our candidate for this job.

```
vi /etc/imapd.conf
configdirectory: /var/imap
partition-default: /var/spool/imap
```

```
servername: CGUX IMAP server
admins: admin1
sasl_pwcheck_method: auxprop
sasl_plugin: sasldb
```

Master process

The master process configuration file /etc/cyrus.conf is read at the startup by the Cyrus master process and is used to control the Cyrus services that are launched. In our case, we stick with only imap and pop services. In addition, it contains the Unix socket path (/var/imap/socket/lmtp) used by the MTA (Postfix in our case) for message delivery.

Cyrus IMAP server comes with several templates of control file that are handy to use. Here is an output of this file with the above-mentioned parameters. A complete copy of our /etc/cyrus.conf can be found in Appendix B.

```
# more /etc/cyrus.conf
# Standard standalone server implementation

<…skip…>

# UNIX sockets start with a slash and are put into /var/imap/sockets
SERVICES {
  # add or remove based on preferences
  imap          cmd="imapd" listen="imap" prefork=0
  pop3          cmd="pop3d" listen="pop3" prefork=0

  # LMTP is required for delivery
  lmtpunix      cmd="lmtpd" listen="/var/imap/socket/lmtp" prefork=0

<…skip…>
```

Configuration directory /var/imap

This directory stores information of the IMAP server as a whole. For example, it contains the directory “socket/lmtp” used by Postfix for local delivery of messages.

The ownership of this directory is granted to the user “cyrus” and the group ownership to mail. Accordingly, we set the permissions to 750.

```
# cd /var
# mkdir imap
# chown cyrus:mail imap
# chmod 750 /var/imap/
```

Spool directory /var/spool/imap

This directory is used for storage of user mailboxes.

```
# cd /var/spool
# mkdir imap
# chown cyrus:mail imap
# chmod 750 imap/
```

Finalize the directories structure

Once we have the base directories for the server configuration and user mailboxes, we can run a Perl script provided with the Cyrus server. This simple script creates the remaining directory structure for the Cyrus IMAP server to work properly. It creates /quota and /socket under /var/imap for respectively user quota configuration and LMTP socket directory. In addition, it creates the directory user under /var/spool/imap for user mailboxes.

Once the script is executed, we can set the ownership to the user and group cyrus.

```
# /var/install/cyrus-imap- cyrus-imapd-2.1.14/tools/mkimap
# /bin/chown –R cyrus:cyrus /var/imap/*
# /bin/chown cyrus:cyrus –R /var/spool/imap/*
```

Access to the LMTP socket

The Cyrus IMAP server expects to receive messages in the /var/imap/socket/lmtp. Remember that we set permission 750 on /var/imap with group ownership to “mail”. As the user “postfix” belongs to the group “mail”, it is allowed to drill down through /var/imap and socket as well (permissions 755) to finally access the specified socket file for message delivery.

Automatic startup

We create a startup file in /etc/init.d used to run the server master process located at /usr/local/cyrus/bin/master. In addition, we add support for the chkconfig utility, which is used to start the IMAP server at boot time. This adds start and kill entries for the Cyrus startup script along with a definition for the Cyrus services.

```
# more /etc/init.d/cyrus
#!/bin/bash
# chkconfig: 2345 20 80
# description: startup script for the Cyrus IMAP server
<…skip…>

/usr/local/cyrus/bin/master &

<…skip…>
```

4.3.5 Postfix

Before starting to work with Postfix, it can be worthwhile checking some of the basic configuration options at: http://www.postfix.org/basic.html

As a pre-requisite for the Postfix installation, we need to create the directories where Postfix needs to store its files (ie /usr/local/postfix).

We also need to create a user “postfix” with a primary group “postfix”. This is the user the postifx system runs as. As for the cyrus user, the Postfix user does not need to have home directory and shell access. A supplementary group called “postdrop” needs also to be setup to ensure the proper operation of the postfix system. Note: the installation is likely to fail if these pre-requisites are not met.

The user “postfix” should be able to access the local database password file sasldb2 along with the Cyrus IMAP socket. We therefore assign it to the “mail” group.

```
# /bin/mkdir /usr/local/postfix

# /usr/sbin/groupadd –g 55552 postfix
# /usr/sbin/groupadd –g 55553 postdrop

/usr/sbin/useradd -u 55552 –g 55552 –d /dev/null -s /bin/NOshell postfix

/usr/sbin/usermod –G mail postfix
```

We unzipped and extract the archive. Our Postfix server has to be compiled with SASL support, so we have to specify the location of the SASL include files and libraries to the compiler.

```
$ gzip -d postfix-2.0.14.tar.gz
$ tar xvf postfix-2.0.14.tar
$ cd postfix-2.0.14

$ make makefiles CCARGS="-DUSE_SASL_AUTH -I/usr/local/include/sasl"
AUXLIBS="-L/usr/local/lib -lsasl2"
```

The installation process is interactive and prompts us for several key features of Postfix: location of the configuration files, location of daemon programs, location of the queue directory. We present the responses we provide below.

```
# make install

Please specify the prefix for installed file names. Specify this ONLY
if you are building ready-to-install packages for distribution to other
machines.
install_root: [/]

Please specify a directory for scratch files while installing Postfix. You
must have write permission in this directory.
tempdir: [/var/install/postfix-2.0.14] /var/tmp

Please specify the final destination directory for installed Postfix
configuration files.
config_directory: [/etc/postfix]

Please specify the final destination directory for installed Postfix
daemon programs. This directory should not be in the command search path
of any users.
daemon_directory: [/usr/local/postfix/bin]

Please specify the final destination directory for installed Postfix
administrative commands. This directory should be in the command search
path of adminstrative users.
command_directory: [/usr/sbin] /usr/local/bin

Please specify the final destination directory for Postfix queues.
```

```
queue_directory: [/var/spool/postfix]

Please specify the final destination pathname for the installed Postfix
sendmail command. This is the Sendmail-compatible mail posting interface.
sendmail_path: [/usr/sbin/sendmail]

Please specify the final destination pathname for the installed Postfix
newaliases command. This is the Sendmail-compatible command to build
alias databases for the Postfix local delivery agent.
newaliases_path: [/usr/bin/newaliases]

Please specify the final destination pathname for the installed Postfix
mailq command. This is the Sendmail-compatible mail queue listing command.
mailq_path: [/usr/bin/mailq]

Please specify the owner of the Postfix queue. Specify an account with
numerical user ID and group ID values that are not used by any other
accounts on the system.
mail_owner: [postfix]

Please specify the group for mail submission and for queue management
commands. Specify a group name with a numerical group ID that is
not shared with other accounts, not even with the Postfix mail_owner
account. You can no longer specify "no" here.
setgid_group: [postdrop]

Please specify the destination directory for the Postfix on-line manual
pages. You can no longer specify "no" here.
manpage_directory: [/usr/local/man]

Please specify the destination directory for the Postfix sample
configuration files.
sample_directory: [/etc/postfix]


Modify various options in the Postfix configuration file
/etc/postfix/main.cf

In order to deliver mail to Cyrus, Postfix need to access the socket lmtp.
We need to set appropriate rights on this directory. See paragraph above on
the permissions set on this directory.

mailbox_transport = lmtp:unix:/var/imap/socket/lmtp
```

The postfix daemon program are stored in /usr/local/postfix/bin and the administrative commands to flush the queue for example are stored in /usr/local/bin. The configuration files are located under /etc/postfix.

The configuration of Postfix is performed using the main.cf file located in /etc/postfix.

The options that we set are detailed below (extract from our main.cf).

```
# vi /etc/postfix/main.cf
<…skip…>
```

```
#The Postfix server should only listen on its production interface
inet_interfaces = 192.168.2.7

#The name of the machine and domain of the machines
myhostname = jupiter.example.com
mydomain = example.com

#The SMTP banner
smtpd_banner = Jupiter.example.com Mail Server
<…skip…>
```

Messages handling

The mail server handles messages for branch1.example.com for local delivery, so we set accordingly the "mydestination" parameter.

Only the corporate mail relay Mercury and the properly authenticated users should be authorized to relay messages through the mail server, so we set "my networks" to 192.168.2.5 and 127.0.0.1. We will detail below how we can restrict to only authenticated users the ability to send messages.

The messages not destined to local recipients should be sent to the corporate mail relay called Mercury with a 192.168.2.5 @ IP. The standard SMTP port 25 is used for message exchange.

```
# vi /etc/postfix/main.cf
<…skip…>
mydestination = branch1.example.com
mynetworks = 127.0.0.1, 192.168.2.5
relayhost = 192.168.2.5:25
<…skip…>
```

We need to verify that the proper Unix socket is configured for message delivery. The option in the main.cf file is mailbox_transport, which should target the appropriate directory. LMTP [5] is used for delivery of messages by Postfix to the back end store, bringing higher performance.

```
# vi /etc/postfix/main.cf
<…skip…>
mailbox_transport = lmtp:unix:/var/imap/socket/lmtp
<…skip…>
```

Configure Postfix for SASL authentication

We compile Postfix with SASL support with the view to authenticate senders using the SASL auxiliary pluging sasldb. The configuration for SMTP AUTH is quite simple and requires the creation of a file called "smtp.conf" located in the directory "/usr/local/sasl2". This file controls the behavior of Postfix when using the SASL library. We indicate in this file the method to use –ie the auxiliary plugin- , the name of the plugin along with the type of mechanism to use –ie only plain mechanism in our case.

[5] LMTP : Local Mail Transport Protocol

```
# more /usr/local/lib/sasl2/smtp.conf
pwcheck_method: auxprop
auxprop_plugin: sasldb
mech-list: plain
```

In addition, we need to activate the SASL option in the main.cf file. The corresponding option to perform SASL authentication is smtpd_sasl_auth_enable that needs to be set to yes. In addition, we want to restrict the capability of sending a message to the users that have been previously authenticated by the server. In addition, we want to authorize only our mail relay Mercury (see my_networks parameter) to send messages. So we set appropriately the smtpd_recipient_restrictions.

```
# vi /etc/postfix/main.cf
<…skip…>
smtpd_sasl_auth_enable = yes
smtpd_sasl_local_domain =
smtpd_sasl_security_options = noanonymous

smtpd_recipient_restrictions =
   permit_sasl_authenticated,
   permit_mynetworks
<…skip…>
```

4.3.6 Stunnel installation

Some critical information might be contained in the messages exchanged between our remote users and their mail server Jupiter. In addition, as we use a PLAIN authentication mechanism, the user password for access to the mailboxes is sent over the Internet in clear-text. Using an SSL tunnel between the client and its mail server offers an adequate level of protection at that point. We setup Stunnel to run on the server side and offers SSL support for the applications that needs it. We intend to use this functionality to protect the IMAP/POP and SMTP traffic.

First, SSL provides support for authentication. We would like to authorize our remote clients to connect to the mail server Jupiter only if they have been previously authenticated at the SSL layer. May a client failed to pass this step, it should be impossible for him to simply open an IMAP or STMP connection to the server. On the server side, Stunnel offers an option to validate the certificate provided by the client and therefore controlling the users with potential access to the messaging resources. Clients should be able to validate the certificate of the mail server, which guarantees that they are speaking to the proper server.

In addition to support authentication, SSL brings encryption features as well. The data sent from the client to the server is encrypted on the client side; then send over the Internet in an encrypted form. Once the traffic reach the client side, the Stunnel daemon decrypts it and then forwards the decrypted data flow to the appropriate final destination port, either the IMAP/POP or the SMTP. This ensures that no data travel in clear over the network and as a consequence the confidentiality of the data is not compromised.

We need first to create a user stunnel who is used to run the Stunnel binaries. This user does not need a home directory and a shell account.

```
# /usr/sbin/groupadd –g 55554 stunnel
# /usr/sbin/useradd -u 55554 –g 55554 –d /dev/null -s /bin/NOshell stunnel
```

Then comes the unzipped and extraction of the Stunnel archive.

```
$ /usr/bin/gzip -d /var/install/stunnel-4.04.tar.gz
$ /bin/tar xf /var/install/stunnel-4.04.tar
$ cd /var/install/stunnel-4.04
```

We use the prefix option to specify the base directory /usr/local/stunnel for the Stunnel application. Then we run make to compile Stunnel and make install to install the binaries at the appropriate location.

```
$./configure --prefix=/usr/local/stunnel --sysconfdir=/etc
$ make
$ make install
```

The stunnel executable is install in /usr/local/stunnel/sbin and the sytem configuration directory is /etc/stunnel. This is where resides the stunnel.conf, which is read by the stunnel daemon at startup.

Creation of a local CA

When it comes to certificate-based authentication, trust between parties should be detailed. We want to setup our own local “certificate authority”. The CA is only used to sign certificates for the mail server Jupiter and the remote users certificates. The OpenSSL binaries are used to assist us in this task.

The first step is to create the proper directory architecture. The base directory is /var/ca.

```
# cd /var
# mkdir ca
# cd ca
# mkdir certs crl newcerts private
# echo "01/n" > serial
# touch index.txt
```

Then we create the local CA private and public keys, the private key being used to sign its own certificate along with server and user certificates. Note: for all the certificate creation, we use a template stored at /var/ca/example.cnf. This is a customized version of the default OpenSSL template and can be found in Appendix B. Keys and certificates of this local CA must not be modified so permissions of 400 should be set on these elements, root being the owner. Special care should be paid on the CA private key as its compromising would result in the compromising of all the

certificates that were signed by this private key. Once the initial setup of the infrastructure is complete, it is a good practice to store a copy of the CA private key on a secure box using a floppy for example.

We use 1024 bits RSA keys that are considered enough long at that time.

```
#/usr/bin/openssl req -new -x509 -config /var/ca/example.cnf -keyout
./private/cakey.pem -out /ca/cacert.pem
Generating a 1024 bit RSA private key
......++++++
............++++++
writing new private key to './private/cakey.pem'
Enter PEM pass phrase: whatever passphrase
Verifying - Enter PEM pass phrase: the same one
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Ile-de-France]:
Locality Name (eg, city) [Paris]:
Organization Name (eg, company) [Example Company]:
Common Name (eg, your name or your server's hostname) []:Certificate
Authority Server
Email Address []:admin-ca@branch1.example.com
```

Now that we have built a CA private key and associated certificate, we can create certificate requests for the mail server Jupiter, the user “testmail”. We only provide the output for the mail server Jupiter, as the procedure is similar for the other operations.

It is important to note the use of the “-nodes” option in the certificate request for server and user requests. This indicates to OpenSSL not to protect the private key with a passphrase. The passphrase is used to protect the private key and needs to be provided every time access to the private key is required. For example, as the mail server will use its private key every time a new SSL connection is opened, it is impossible for somebody just to sit here and wait for entering the server passphare…

```
#/usr/bin/openssl req -new -nodes -config /ca/example.cnf -keyout
/ca/private/jupiterkey.pem -out /ca/reqs/jupiterreq.pem  Generating a 1024
bit RSA private key
........++++++
..++++++
writing new private key to '/ca/private/jupiterkey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

```
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Ile-de-France]:
Locality Name (eg, city) [Paris]:
Organization Name (eg, company) [Example Company]:
Common Name (eg, your name or your server's hostname) []:jupiter.example.com
Email Address []:admin-jupiter@branch1.example.com
```

Now that we have a brand new certificate request, we can have it signed by our private CA key. OpenSSL CA mode would do that for us. The CA private key passphrase is required to sign the certificate request (Access to CA private key). We enter yes to sign the certificate request for the server.

```
# openssl ca -config /ca/example.cnf -policy policy_example -out /ca/certs/mailkey.pem -infiles
/ca/reqs/mailreq.pem
Using configuration from /ca/example.cnf
Enter pass phrase for /ca/private/cakey.pem: CA private key passphrase
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 3 (0x3)
        Validity
            Not Before: Aug 20 06:04:51 2003 GMT
            Not After : Aug 19 06:04:51 2004 GMT
        Subject:
            countryName               = FR
            stateOrProvinceName       = Ile-de-France
            localityName              = Paris
            organizationName          = Example Company
            commonName                = jupiter.example.com
            emailAddress              = admin-mail@branch1.example.com
        X509v3 extensions:
            X509v3 Basic Constraints:
            CA:FALSE
            Netscape Comment:
            OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
            90:DE:34:50:90:79:1A:13:A8:8F:51:EB:D7:C8:52:BC:B6:5C:D4:6A
            X509v3 Authority Key Identifier:

keyid:35:8C:0F:BB:47:F4:BC:3F:96:F6:78:7F:4A:59:5A:CB:C5:2C:CD:E7
            DirName:/C=FR/ST=Ile-de-France/L=Paris/O=Example Company/CN=Certificate Authority
Server/emailAddress=admin-ca@branch1.example.com
            serial:00

Certificate is to be certified until Aug 19 06:04:51 2004 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Now we can verify that our certificates are valid with the OpenSSL "-verify" option.

```
#/usr/bin/openssl verify -verbose -CApath /ca -CAfile /ca/cacert.pem
/ca/certs/jupitercert.pem
/ca/certs/jupitercert.pem: OK
#/usr/bin/openssl verify -verbose -CApath /ca -CAfile /ca/cacert.pem
/ca/certs/testmailcert.pem
/ca/certs/testmailcert.pem: OK
```

Delivery to the users

Once generated the user private key and certificate should be deliver to the final user. We do not setup a complete PKI (Public Key Infrastructure), which includes procedures that can be time and resource consuming. This is not the purpose of the document to detail that part.

Finalize the Stunnel configuration

CA certificate:

The Stunnel daemon needs to read the CA certificate, which should be copied in "/etc/stunnel/stunnel.pem". The CA certificate is used to validate the certificates a client or server is presented to the Stunnel daemon. The digital signature of a certificate is the fingerprint of the certificate signed by the CA private key. By running the same hash algorithm on the submitted certificate and encrypting the corresponding digital signature with the CA public key, we should obtain the same piece of data, which then guarantees the integrity of the certificate. The controls that need to be operated on the certificate are integrity check (the certificate has not been altered), the date check (the certificate has not expired) and the validity check (the certificate is not on a revocation list). Note that Stunnel does not support revocation list checking but another mechanism, which is detailed below.

```
# cp /ca/cacert.pem /etc/stunnel/stunnel.pem
```

Jupiter server requirements:

Stunnel requires the location of the private key and certificate for the mail server Jupiter. This certificate has to be provided to the client when requesting for an SSL.

In order to work properly, Stunnel requires one single file that contains both element –ie private key and certificate. We use the command cat to create the concatenated version of the 2 files.

```
# cat /ca/private/jupiterkey.pem mailcert.pem >
/etc/stunnel/servercert/jupitercert.pem
# cp /ca/private/jupiterkey.pem /etc/stunnel/servercert/
```

Client requirements:

The Stunnel daemon needs to know which client certificates are valid and which are not. Generally, when a user looses its private key or in case of a compromising of this key, it should refer this to the administrator. This should lead to

the publication of a CRL (Certificate Revocation List) containing all the certificates that are not valid any longer. Stunnel does not support CRL checking at that time. However, Stunnel does provide certificate validity check through a simple repository that contains all the certificates considered as trusted. This seems to fit our needs in a small to medium architecture.

To perform certificate checking, Stunnel parse the content of the repository and look for specific file name. The certificate inside the repository should be named according to the result of an OpenSSL hash function. As a consequence, we generate a hash for the client certificate in order to rename the certificate with it, as per what is expected by Stunnel.

We define the directory containing the certificate that we trust to /etc/stunnel/trustedcerts.

In the example below, the result of the hashed function is 81dbaa70.0.

```
# /usr/bin/openssl x509 -hash -noout -in testmailcert.pem
81dbaa70
# mv /etc/stunnel/trustedcerts/testmailcert.pem
/etc/stunnel/trustedcerts/81dbaa70.0
```

Completing the configuration of Stunnel

There are some points in the configuration file of Stunnel (/etc/stunnel/stunnel.conf) that need to be detailed:

- We specify the location of the Jupiter server certificate and key, along with the path for the CA certificate (CAfile).
- As indicated, the Stunnel daemon runs as the user “stunnel”.
- We set the verify attribute to the level of 3 in order to perform authentication based on the certificate the client presents, and we specify the location of the directory that contains the certificates that we trust (CApath).
- Stunnel runs in server mode. This means that for each services we want to SSL-enabled (POP, IMAP or SMTP), we specify an accept option. This means that the Stunnel daemon listens for SSL connection on this interface/port. Stunnel listens on 192.168.2.7 on the port 995, 993 and 465.
- Once Stunnel receives a connection on one of this port, it should decrypt the traffic flow and forward it to the appropriate service. This is the goal of the instruction “connect”. For the IMAP service, once Stunnel receives an encrypted flow on the port 993, it forwards the decrypted flow to the port 143.
- As detailed in the previous section, the IMAP server and Postfix listens on port 25, 110 and 143. These ports are all used for the Stunnel “connect” parameter.

```
# more stunnel.conf
#Configuration file for example.com
```

```
key = /etc/stunnel/servercert/jupiterkey.pem
cert = /etc/stunnel/servercert/jupitercert.pem
CAfile = /etc/stunnel/stunnel.pem

pid = /var/stunnel/stunnel.pid

setuid = stunnel
setgid = stunnel

#Authentication attributes
verify = 3
CApath = /etc/stunnel/trustedcerts/

# Some debugging stuff
debug = 2
output = /var/log/stunnel.log

# Service-level configuration
[pop3s]
accept  = 192.168.2.7:995
connect = 192.168.2.7:110
TIMEOUTclose=0

[imaps]
accept = 192.168.2.7:993
connect = 192.168.2.7:143
TIMEOUTclose=0

[smtps]
accept = 192.168.2.7:465
connect = 192.168.2.7:25
TIMEOUTclose=0
```

4.4 Post-installation tasks

4.4.1 SSH configuration

The remote administration of the system must be performed using SSH. The SSH server provides encryption of traffic between the administrator machine and our server. This is a great advantage compared to a protocol like telnet, which can only support exchange of non-encrypted data traffic between 2 hosts.

However, the configuration of SSH demands some work to be done properly thus avoiding the setting up a poorly protected system:

- The SSH daemon should only listen on the management interface of the server.
- The default port to connect to the SSH server is 22. In order to increase the level of security, it can be changed to another dedicated port, which is only known by the administrators of the system. Note that we stick with port 22 in the rest of the paper.

- Force to use SSH v2 only instead of SSH v1, as some limitations and vulnerabilities was discovered in the first release of SSH.
- Privilege separation using the pseudo-user “sshd" should be used.
- Administrators must authenticate to the server using their certificate. As a consequence, we disable the password-based authentication and make only public key mechanism available to connect to the server. In addition, we disable the parameters controlling the host-based authentication mechanism.
- Disable root login. Administrator should log in using their personal account and then if needed “su” to root. This ensures a better accountability on the system as we can trace which account has elevated its privilege to root.
- The access to the SSH daemon should be restricted only to the machines of the corporate management LAN, ie 10.10.10.0/24. This part is detailed later on the TCP Wrapper configuration.
- Only authorized users should be able to log in to the system.

The file that triggers these parameters is /etc/ssh/sshd_config
What is configured is depicted below.

```
# vi /etc/ssh/sshd_config
# General configuration parameters
Port 22
Protocol 2
ListenAddress 192.168.10.7

# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
AuthorizedKeysFile .ssh/authorized_keys

# Authentication features: only public key is enabled
PubkeyAuthentication yes
HostBasedAuthentication no
PasswordAuthentication no
PermitEmptyPasswords no

# Loging
SyslogFacility AUTH
LogLevel INFO

# Control the users who can log in
AllowUsers admin1 admin2
PermitRootLogin no

# the SSH server should be run as the SSH users
UsePriviledeSeparation yes

# General parameters
Compression yes
Banner /etc/issue
PrintMotd yes
```

```
KeepAlive yes
```

Public/private key pair for the SSH server:

We use 1024 bits DSA and RSA key pair for the server SSH daemon and store them as specified in the configuration file (respectively, /etc/ssh/ssh_host_rsa_key and /etc/ssh/ssh_host_dsa_key). We do not have to set a passphrase to protect these keys. We modify access control on these files: the owner should be root and the permissions should be set to 400.

Creation of public/private key pair for the administrator “admin1”:

```
MachineAdmin1$ /usr/sbin/ssh-keygen -t dsa -b 1024
Generating public/private dsa key pair.
Enter file in which to save the key (/home/admin/.ssh/id_dsa):
Created directory '/home/admin/.ssh'.
Enter passphrase (empty for no passphrase): Any Passphrase
Enter same passphrase again: Any Passphrase
Your identification has been saved in /home/admin/.ssh/id_dsa.
Your public key has been saved in /home/admin/.ssh/id_dsa.pub.
The key fingerprint is:
cf:07:79:24:f9:ef:4a:4a:46:bc:ed:52:c6:1d:2c:3e admin1@jupiter.example.com
```

The user private DSA key is located under /home/admin/.ssh/id_dsa and the public key is located under /home/admin/.ssh/id_dsa.pub. It is important to set up a passphrase that acts as a password to protect the private key used for each SSH connection.

Following this, in order for the user “admin1” to be authenticated to our server, he needs to copy its public key to his own home directory on the Jupiter server where he has access: /home/admin1/.ssh/authorized_keys.

Admin1 can now log on to the server after successful authentication using its private key/certificate.

4.4.2 TCP Wrapper configuration

TCP Wrapper is used to control network access to resources on our mail server. Some executables like SSH are compiled with the support of the TCPWrapper library.

Fist we can verify that our SSH server was properly compiled for using TCPWrapper library. Just issue a ldd /usr/sbin/sshd and verify that libwrap.so.0 is present. (The ldd utility enables to check which loadable module an executable was compiled with).

TCPWrapper performs access control after consultation of 2 files: first the /etc/hosts.allow file, which contains any pairs of daemon/IP address that have access to the system; second the /etc/hosts.deny file, which contains pairs of daemon/IP address that are not allowed access to the system. The first match stops the lookup.

A good practice is to explicitly define in the hosts.allow what pairs is granted access and prevent anything else from accessing the system in the hosts.deny file.

As a consequence, we configure TCPWrapper as follow:

- Permit full access from localhost 127.0.0.1 and permit access to the sshd daemon from the management network 10.10.10.0/24

```
# more /etc/hosts.allow
ALL: 127.0.0.1
sshd: 10.10.10.0/24
```

- Deny all other access

```
/etc/host.deny
ALL: ALL
```

The TCPWrapper library provides an additional layer of security to our host in addition to the host-based firewall configuration that will be detailed below.

4.4.3 Sudo configuration

Sudo - superuser do - can be used to provide certain users the ability to run a set of commands with root level privilege. In addition, Sudo permits to log the commands run by users on the system. The behavior of the Sudo utility is controlled by the /etc/sudoers file. A specific command (run as root) called visudo enables to configure the "sudoers" file.

How it works is relatively simple: if a user want to run a command that requires more privilege that he has, he use the command "sudo" followed by the command he want to run. For example, he might want to edit the /etc/passwd, so in this case, he would type "sudo /etc/passwd". The system prompts him for his own password (NOT the root password). If the password he enters is correct and the access control permits it, he gains access the desired file. The obvious advantage here is that the root password does not need to be known by the user.

In our case, we would like to permit the administrators, who are members of the group wheel to run any command on the system. With the NOPASSWD option enabled, the administrator will not be prompted for his password every time they try to call Sudo.

Here is what we need to add in the sudoers file using the visudo:

```
# visudo
%wheel ALL=(ALL)       NOPASSWD: ALL
```

4.4.4 File system security

The /usr partition should be mounted as ro in order to protect binaries from being potentially tampered. /home; /var should be mounted as rw, nosuid, nodev. As explained in SANS GCUX practicum, the root file system cannot be mounted with any of these options as some SUID binaries reside in /bin or /lib, which makes it impossible to use the nosuid option. The file to modify is /etc/fstab, which is read for mounting the partition during the boot of the machine.

4.4.5 Firewall configuration

The kernel version 2.4.20 supports the use of the Netfilter iptable filtering module. IPtable is a host-based firewall and control traffic entering and leaving the network interfaces of the machines.

We implement a “Deny everything except what is explicitly authorized” policy based on the following traffic flow definition:

- Production interface: accept the traffic from anywhere to the server for the IMAPS, POPS and SMTPS. This ensures that the remote users can access only the secured network services running on the machine whether they are connected outside the company or on the corporate network. The second point is not lead by a security purpose as traffic from the corporate network to a server in DMZ can go unencrypted. However, it is convenient to have the same configuration for the 2 situations on the client side.
- Production interface: the SMTP traffic is allowed inbound and outbound between the mail relay Mercury and the mail server Jupiter. The mail coming from the Internet first flows through Mercury where several check are performed (anti-virus scanning, anti-spam…) before reaching Jupiter. Messages send from Jupiter to outside recipients have to pass through the mail relay Mercury before delivery (again virus scanning…).
- Production interface: The SMTP traffic is allowed inbound and outbound between the mail relay Jupiter and the internal mail server called Mars. This reduces the load on the mail relay by directly delivering the messages between the company users to the internal mail server.
- Management interface: Only SSH is allowed from the management network to the management interface of the server.
- Production interface: Outbound DNS and NTP traffic is allowed to the appropriate servers.
- Both interfaces: Inbound and Outbound ICMP is permitted for troubleshooting purpose.

IPtable was part of the initial OS installation. The initialization script looks for the IPtable configuration file in /etc/sysconfig/iptables to start the application. The chkconfig utility is used to start IPtable when the server boots. So we build the IPtable configuration file based on the ruleset detailed above and save it in /etc/sysconfig/iptables.

```
# more /etc/sysconfig/iptables

# Production interface eth0 configuration
# Authorized new connection to the server for SMTPS, IMAPS and POPS
/sbin/iptables -A INPUT -p tcp --dport 465 \
-m state --state NEW,ESTABLISHED -i eth0 --j ACCEPT
/sbin/iptables -A OUTPUT -p tcp --sport 465 \
-m state --state ESTABLISHED -o eth0 --j ACCEPT
```

```
/sbin/iptables -A INPUT -p tcp --dport 993 \
-m state --state NEW,ESTABLISHED -i eth0 --j ACCEPT
/sbin/iptables -A OUTPUT -p tcp --sport 993 \
-m state --state ESTABLISHED -o eth0 --j ACCEPT

/sbin/iptables -A INPUT -p tcp --dport 995 \
-m state --state NEW,ESTABLISHED -i eth0 --j ACCEPT
/sbin/iptables -A OUTPUT -p tcp --sport 995 \
-m state --state ESTABLISHED -o eth0 --j ACCEPT


# Authorized SMTP traffic between Jupiter and Mercury
/sbin/iptables -A INPUT -p tcp --dport 25 -s 192.168.2.5 \
-m state --state NEW,ESTABLISHED -i eth0 --j ACCEPT
/sbin/iptables -A OUTPUT -p tcp --dport 25 -d 192.168.2.5 \
-m state --state NEW,ESTABLISHED -o eth0 --j ACCEPT


# Authorized SMTP traffic between Jupiter and Mars
/sbin/iptables -A INPUT -p tcp --dport 25 -s 10.1.1.5 \
-m state --state NEW,ESTABLISHED -i eth0 --j ACCEPT
/sbin/iptables -A OUTPUT -p tcp --dport 25 -d 10.1.1.5 \
-m state --state NEW,ESTABLISHED -o eth0 --j ACCEPT


# Authorized outbound DNS request
/sbin/iptables -A OUTPUT -p udp -dport 53 -d 192.168.2.10 \
-m state --state NEW,ESTABLISHED -o eth0 --j ACCEPT
/sbin/iptables -A INPUT -p udp -sport 53 -d 192.168.2.10 \
-m state --state ESTABLISHED -i eth0 --j ACCEPT


# Authorized outbound NTP request
/sbin/iptables -A OUTPUT -p udp -dport 123 -d 192.168.2.10 \
-m state --state NEW,ESTABLISHED -o eth0 --j ACCEPT
/sbin/iptables -A INPUT -p udp -sport 123 -d 192.168.2.10 \
-m state --state ESTABLISHED -i eth0 --j ACCEPT

# Log and Drop rules
/sbin/iptables -A INPUT -j LOG -i eth0 \
--log-prefix "Iptable eth0 IN Packet"
/sbin/iptables -A INPUT –i eth0 -j DROP

/sbin/iptables -A OUTPUT -j LOG -o eth0 \
--log-prefix "Iptable eth0 OUT Packet"
/sbin/iptables -A OUTPUT –o eth0 -j DROP

# Configuration for the Management interface eth1
# Allow inbound SSH connection from the management LAN
/sbin/iptables -A INPUT -p tcp --dport 22 -s 10.10.10.0/24 \
-m state --state NEW,ESTABLISHED -i eth0 --j ACCEPT
/sbin/iptables -A OUTPUT -p tcp --sport 22 -d 10.10.10.0/24 \
-m state --state ESTABLISHED -o eth0 --j ACCEPT

# Log and Drop rules
/sbin/iptables -A INPUT -j LOG -i eth1 \
```

```
--log-prefix "Iptable eth1 IN Packet"
/sbin/iptables -A INPUT –i eth1 -j DROP
/sbin/iptables -A OUTPUT -j LOG -o eth1 \
--log-prefix "Iptable eth1 OUT Packet"
/sbin/iptables -A OUTPUT –o eth1 -j DROP
```

4.4.6 Syslog configuration

The syslog daemon listens for syslog messages and store them according to its configuration file /etc/syslog.conf. Syslog is a great tool for monitoring and performing accounting tasks on a system. The syslog daemon is configured to log messages based on a facility (auth for authentication messages for example) and a severity (from info to emergency). Based on these 2 values, we implement several entries in the configuration file to store log messages in the appropriate file (/var/log/auth.log of authentication message for example).

Syslog is configured to log messages as follow:

```
# vi /etc/syslog.conf
<…skip…>
# Auth generated by SASL logged to /var/log/auth.log with auth facilities
auth.log                                                /var/log/auth.log

# IMAP logged to /var/log/imap.log with local6 facilities
local6.*                                                /var/log/imap.log

# All mail log are stored in /var/log/maillog
mail.*                                                  /var/log/maillog
<…skip…>
```

The files specified in the syslog configuration should be created prior restarting the syslog daemon. All space in the configuration file should be TAB.

Once the configuration is finalized, it is time to restart the syslog daemon and make some testing with the logger command.

```
# pkill -HUP syslogd
# logger -p authpriv.info -t TEST-Syslog Test to authpriv facility with info
severity

# tail -f /var/log/secure
<…skip…>
Nov 10 07:04:39 mail TEST-Syslog: Test to authpriv facility with info
severity
```

The syslog messages should be both send to the local syslog server and to a remotely centralized syslog server. Another entry labeled “loghost” should be added in the /etc/hosts file for the centralized syslog server in addition to the loopback address 127.0.0.1. For the messages that are logged locally, they have to be archived or simply purged every week for example using a utility like logrotate (shipped with the Linux distribution)

4.4.7 Tripwire configuration

Tripwire is a file integrity checker that aids in the detection of unauthorized changes of files. Tripwire enables system administrators to monitor any changes that might happen on a set of crucial files to the system.

The way it works is rather simple:

- Generate an initial fingerprint of critical files and directories on the system and store this information inside a local database.
- Get the same fingerprint of the system on a daily basis and compare it with the one initially generated.
- Look for any discrepancies between the 2 fingerprints.
- In case of difference between the 2 files, notify the administrator for further analysis.

The first step is to generate a local and site keys that are used to protect the Tripwire files using digital signature. The local key is used to sign the database file containing the fingerprint of the system. The site passphrase is used to protect Tripwire information that can be shared between machines. This can be for example the Tripwire configuration file, which can be the same for a large number of machines of the same company.

These 2 keys are protected using a passphrase that will be asked by Tripwire whenever it needs access one of these keys.

So let us generate these keys and protect them with 2 passphrases. We store these keys in the /etc/tripwire.

```
# twadmin --generate-keys --local-keyfile /etc/tripwire/tw-local.key \
--local-passphrase local-pass --site-keyfile /etc/tripwire/tw-site.key \
-- site-passphrase site-pass

Using supplied passphrases.
Generating site key: /etc/tripwire/tw-site.key
Generating key (this may take several minutes)...Key generation complete.
Generating local key: /etc/tripwire/tw-mail.key
Generating key (this may take several minutes)...Key generation complete.
```

The second step includes the creation of the Tripwire configuration and policy files that will be signed using the previously generated site key. We edit manually the clear text version of these 2 files as they serve as inputs in the generation of the digitally signed version of these files (configuration + policy).

The configuration file “tw.cfg” contains several variables required for Tripwire to run properly. This includes the location of the keys, database files to store reports…

The policy file “tw.pol” contains the files that need to be inspected by Tripwire. Every time it runs, Tripwire captures the fingerprint of these files, so this file has to be configured correctly. The default Tripwire policy file that is shipped with the RedHat distribution is a good starting point for building the new policy.

In particularly, this includes:

- Binaries located under /bin, /sbin, /usr/local/bin.
- Root profile /.profile.

- Configuration files located under /etc. This includes the configuration files for Cyrus IMAP, Postfix, Stunnel.
- Log file in /var/log.
- Devices files.

```
# twadmin --create-cfgfile --cfgfile tw.cfg --site-keyfile tw-site.key --
site-passphrase site-pass /etc/tripwire/twcfg.txt
Wrote configuration file: /etc/tripwire/tw.cfg

# twadmin --create-polfile -v --cfgfile tw.cfg --polfile tw.pol --site-
keyfile tw-site.key --site-passphrase site-pass twpol.txt
Opening configuration file: /etc/tripwire/tw.cfg
This file is encrypted.

Opening key file: /etc/tripwire/tw-site.key
Using plaintext policy file: /etc/tripwire/twpol.txt
Opening key file: /etc/tripwire/tw-site.key
Wrote policy file: /etc/tripwire/tw.pol
```

Step 3: once we have our policy file, we can initiate the database

```
# tripwire --init -v --cfgfile tw.cfg --local-passphrase local-pass

<skip>

--- Generating information for: /dev/tty6
Processing:  /dev/urandom
--- Generating information for: /dev/urandom
Processing:  /dev/initctl
--- Generating information for: /dev/initctl
Wrote database file: /etc/tripwire/tw.db
The database was successfully generated.
```

Based on the policy file that was used, some tuning may be required. For example, if some files included in the policy file are not present on the machine, this can generate some non-useful alert.

Step 4: we can then compare the state of our system with the baseline that we capture in the initialization of the database.

```
# tripwire --check -v -- cfgfile /etc/tripwire/tw.cfg
<…skip…>
```

You would receive some messages from Tripwire telling that the file is trying to access does not exist. All these non-existing files that generate errors when running an integrity check must be remove from the twpol.txt file. Then a new initialization of the database and a new integrity test would solve the problem.

This will generate a report called Jupiter-date.twr in /var/tripwire/report digitally signed using the local key file.

Step 5: Print the report generated

```
# twprint --print-report --twrfile /var/tripwire/report/Jupiter-20031126-
073252.twr | more

Tripwire(R) 2.3.0 Integrity Check Report

Report generated by:          root
Report created on:            Wed 26 Nov 2003 07:54:24 AM GMT+2
Database last updated on:     Never

================================================================================
===
Report Summary:
================================================================================
===

Host name:                    Jupiter.example.com
Host IP address:              127.0.0.1
Host ID:                      None
Policy file used:             /etc/tripwire/tw.pol
Configuration file used:      /etc/tripwire/tw.cfg
Database file used:           /var/tripwire/twdbfile.twd
Command line used:            tripwire --check --cfgfile tw.cfg

================================================================================
===
Rule Summary:
================================================================================
===

--------------------------------------------------------------------------------
---
  Section: Unix File System
--------------------------------------------------------------------------------
---

  Rule Name                       Severity Level    Added    Removed
Modified
  ---------                       --------------    -----    -------  ------
--
  Invariant Directories           66                0        0        0
  Temporary directories           33                0        0        0
  Tripwire Data Files             100               0        0        0
  Critical devices                100               0        0        0
  User binaries                   66                0        0        0
  Tripwire Binaries               100               0        0        0
  Critical configuration files    100               0        0        0
  Libraries                       66                0        0        0
<…skip…>
```

This type of reports displays the modifications that might appear to the files and directories that are monitored using Tripwire.

5 Ongoing maintenance

The maintenance operation helps keeping the system in an appropriate working state. After a fresh installation, the system is correctly patched and the applications that we are running do not present any known vulnerabilities. This is likely to change during the life of the system, so an active monitoring of the vulnerabilities being discovered for the OS and the applications is required to keep the system up to date.

Another major point for the maintenance is the day-to-day operation system administrators need to perform on the system. This includes –but is not restricted- to backup policy, log analysis, Tripwire check verification, periodic scan.

5.1 System review

Looking for new patches for a RedHat system brings to RedHat Errata web site (https://rhn.redhat.com/errata/rh9-errata-security.html), where the latest advisories related to new discovered vulnerabilities are published. It is always a good thing to spend some time here and seek for information.

Registering to a few Security Alert mailing lists can greatly help when it comes to patch management. It is a good idea to register to the CERT advisories (www.cert.org), SANS web site (www.sans.com).

We can choose between 2 solutions for keeping our RedHat 9 distribution at the most current patch level. The first one is downloading the new patches from the errata web site and apply them to the server. The procedure for applying new patches is detailed in 4.2.1. A test server –if available-- where we can install a patch before implementation on the production server can be a good practice. The second way for keeping the OS up to date is to register the machine to the RedHat Network at www.rhn.com. This enables to use the up2date utility that comes with the RedHat distribution and enables the automatic download of the OS patches. This service comes for free for individual, but you will have to pay for it for the computer of your company. As this is a convenient way to keep the system up-to-date, we think it is worth paying for the RedHat Network service.

Note that RedHat has announced that the support for RedHat 9 will end in the middle of 2004. As we start working on the project before this announcement, we keep on working on this version of RedHat. For any other project, it should be interesting to spend some time on the Enterprise version of the RedHat distribution or or another distribution of Linux.

Now that our RedHat distribution is properly patched, we have to keep our applications up to date as well. In addition to the mailing list mentioned above, the application web site and mailing list for Postfix, IMAP and Stunnel assist in deciding whether of not an application upgrade is required.

The bottom line is to be sure that both the OS and the applications installed are reviewed on a regular basis.

5.2 Backup requirement

As this system contains user mailboxes with potentially valuable data, we should pay a special attention to the backup policy. The 2 things that we have to

define are: what are the data that we have to be archived on the machine and what is the frequency to perform the backup operations.

The backup policy recommends archiving all the configuration files and user data. The OS and applications binaries can be rebuilt during a new installation of a new system. As a consequence, here is a list of what has to be backup for our mail server:

- /var/imap and /var/spool/imap: IMAP server configuration and user data.
- /etc/postfix: Postfix configuration files.
- /etc/cyrus.conf ; /etc/imap.conf : Cyrus IMAP configuration files.
- /etc/stunnel : Stunnel configuration files and trusted certificates repository.
- /var/sasldb2 : database for user credentials information
- /etc/tripwire : Tripwire configuration file and database
- /etc/ssh: ssh configuration and keys. These may be regenerated after a new install of the system
- /etc/syslog.conf: syslog configuration file
- /etc/hosts.allow, /etc/hosts.deny : TCP Wrapper policy file

Due to the criticality of the data on the system, the backup operation is run daily after peak hours. The routine operation is performed every day at midnight and sends all the data to archive to a tape drive. This ensures that in case of a hardware crash, users will not loose a great amount of data.

5.3 System analysis

5.3.1 Log analysis: daily

A common task for system administrator is an active analysis of the logs generated by the system. A specific attention should be paid to authentication logs.

5.3.2 Tripwire report:

We automate the check performed by Tripwire on a daily basis. An alert generated by Tripwire may result of a system compromise or a modification performed without notification. In any case, this has to be investigated every time Tripwire pops up for an alert.

5.3.3 Periodic scan: monthly

After the initial setup of the system is complete, we run a scan to check that only the appropriate ports are opened on the machine. We should run the same type of scan every month to check if no other port has been turn on by a potential intruder.

6 Testing the configuration

6.1 Cyrus IMAP

We need to test that our IMAP server authenticate properly the users and enables them to retrieve their message.

6.1.1 Cyradm utility

The cyradm utility helps creating and managing users mailboxes. We need to modify the cyradm perl script to indicate the path for the Perl modules.

```
# vi /usr/local/bin/cyradm
<…skip…>
x) exec perl -I/usr/local/lib/perl5/site_perl/5.8.0/i386-linux-thread-multi/
-MCyrus::IMAP::Shell -e shell -- ${1+"$@"} ;;
*) exec perl -I/usr/local/lib/perl5/site_perl/5.8.0/i386-linux-thread-multi/
-MCyrus::IMAP::Shell -e shell -- "$@"
;;
<…skip…>
```

6.1.2 User mailboxes creation

We can create a mailbox for our “testmail” user. To do so, we login as the “admin1” user (this admin1 user is defined in /etc/imapd.conf) to the cyrus server. The credentials for the “admin1” users are validated against the sasldb2 credentials database.

The command “cm user.testmail” creates a brand new mailbox for the “testmail” user. We can take the opportunity to limit the size of this mailbox to 50MBytes.

```
$ cyradm --user admin --auth plain 192.168.2.7
Password: <enter admin password here>
IMAP Password: <enter admin password here>
192.168.2.7> cm user.tesmail
192.168.2.7> setquota user.testmail 50000
```

6.1.3 IMAP functionning

We can now test that the “testmail” user can authenticate using its SASL password “testmail” to the Cyrus mail server.

```
$ imtest -a testmail -w testmail -v -m login 192.168.2.7
S: * OK Jupiter.example.com Cyrus IMAP4 v2.1.14 server ready
C: C01 CAPABILITY
S: * CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ MAILBOX-REFERRALS
NAMESPACE UIDPLUS ID NO_ATOMIC_RENAME UNSELECT CHILDREN MULTIAPPEND SORT
THREAD=ORDEREDSUBJECT THREAD=REFERENCES IDLE
S: C01 OK Completed
C: L01 LOGIN testmail {8}
S: + go ahead
```

```
C: <omitted>
S: L01 OK User logged in
Authenticated.
Security strength factor: 0
```

6.2 Postfix

This part of the test includes the testing of the authentication of the users before sending a message, the proper delivery of message for local recipients and the proper relaying of messages to outside recipients.

We intend to only test here the Postfix configuration. However, we need to connect to the Postfix server using the port 465 used for SMTP over SSL instead of the classical port 25. We detail this part later in the document when testing the configuration of Stunnel.

6.2.1 Sender authentication

When sending his credentials to the Postfix server for authentication purpose, the server expects the base 64 endoded form of the user password instead of its clear text form “testmail”. (The message is not encrypted but just encoded, no security is added by the use of a 64 encoded message). The-mail client program performs this encoding operation prior to send user credential to the server. However, as we are doing some manual testing here, we need to create the base 64 encoded form of the password using a perl module.

```
$ perl -MMIME::Base64 -e 'print
encode_base64("testmail\0testmail\0testmail")'
dGVzdG1haWwAdGVzdG1haWwAdGVzdG1haWw=
```

We can now connect to the Postfix server and verify that only a plain authentication mechanism is provided as configured. A first test with a wrong password returns an error. Providing the encoded version of the appropriate password for the “testmail” user ensures a successful authentication.

After a successful authentication, the Postfix server properly accepts the message for delivery to a local recipient (ie: tesmail1@branch1.example.com).

```
$ telnet 192.168.2.7 465
220 jupiter.example.com mail server
ehlo test@example.com
250-jupiter.example.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-AUTH PLAIN
250 8BITMIME
auth plain BadTest
535 Error: authentication failed
auth plain dGVzdG1haWwAdGVzdG1haWwAdGVzdG1haWw=
```

```
235 Authentication successful
mail from: testmail@branch1.example.com
250 Ok
rcpt to: testmail1@ branch1.example.com
250 Ok
Data
354 End data with <CR><LF>.<CR><LF>
From: testmail@ branch1.example.com
To: testmail1@ branch1.example.com
This is mail sent after been successfully authenticated.
.
250 Ok: queued as 29E90FA48
```

6.2.2 Reject message sent by a non-authenticated user

A user that has not being authenticated should not be able to send a message using the Postfix server. Let's test this using the same sender/recipient couple as previously but without providing credentials.

```
$ telnet 192.168.2.7 465
Trying 192.168.2.7...
Connected to 192.168.2.7.
Escape character is '^]'.
220 jupiter.example.com mail server
ehlo me
250-jupiter.example.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-AUTH PLAIN
250 8BITMIME
mail from: testmail@branch1.example.com
250 Ok
rcpt to: testmail1@ branch1.example.com
554 <testmail1@example.com>: Recipient address rejected: Relay access denied
quit
221 Bye
Connection closed by foreign host.
```

6.2.3 Delivery capability

The Postfix server should sent messages to non-local recipients to the corporate mail relay Mercury 192.168.2.5.

```
$ telnet 192.168.2.7 465
Trying 192.168.2.7...
Connected to 192.168.2.7.
Escape character is '^]'.
220 jupiter.example.com mail server
ehlo me
250-jupiter.example.com
250-PIPELINING
```

```
250-SIZE 10240000
250-VRFY
250-ETRN
250-AUTH PLAIN
250 8BITMIME

<…skip authentication part…>

mail from: tesmail@branch1.example.com
250 Ok
rcpt to: outside-rcipient@yahoo.com
250 Ok
data
354 End data with <CR><LF>.<CR><LF>
From: testmail@branch1.example.com
To: outside-recipient@yahoo.com
This is a message destined to an outside recipient.
.
250 Ok: queued as 1B453FA48
```

We monitor the connection opened from the mail server using the netstat command.

```
$ netstat -an | grep 25
<…skip…>
tcp        0      0 192.168.2.7:32778       192.168.2.5:25        CONNECTED
<…skip…>
```

6.3 Stunnel

When testing Stunnel, we need to be sure that the only the clients that we considered as trusted are authorized to initiate SSL connection to the mail server. In addition to this, the traffic between the user and the server should be encrypted.

6.3.1 Client configuration

Many mail clients provide SSL support. An alternative solution is to use Stunnel in client mode. In client mode, Stunnel accepts clear text data connection from the local machine, encrypts it and sends it to the remote server, where Stunnel is running in server mode.

For an SMTP connection, the mail client program sends in clear the SMTP flow locally to port 25. The Stunnel daemon listens locally on port 25, then gets the connection, encrypts the flow and sent it to the public IP of the mail server.

Here is an example of an stunnel.conf file set up on the client machines:

```
# more /etc/stunnel/stunnel.conf
key = /etc/postfix/stunnel/testmailkey.pem
cert = /etc/postfix/stunnel/testmailcert.pem

# Some debugging stuff
debug = 7
output = C:\stunnel.log
```

```
# Use it for client mode
client = yes

# Service-level configuration

[pop3s]
accept  = 110
connect = <mailserverpublicIP>:993
TIMEOUTclose=0

[imaps]
accept  = 143
connect = <mailserverpublicIP>:995
TIMEOUTclose=0

[smtps]
accept = 25
connect = <mailserverpublicIP>:465
TIMEOUTclose=0
```

6.3.2 Valid client certificate

As explained above when describing the configuration of Stunnel, we add in the "trustedcerts" directory the certificate for the "testmail" user; which should authorized him to connect to the mail server Jupiter using an SSL connection.

After initiating a telnet connection on port 465, we check the logs on the mail server. We highlight the part of the log related to the establishment of the successful verification of the client certificate that enables to pursue the SSL connection establishment

$ tail –f /var/log/stunnel.log
2003.08.20 08:16:47 LOG7[1357:1075930272]: smtps accepted FD=9 from clienttest:1355
2003.08.20 08:16:47 LOG7[1357:1075930272]: FD 9 in non-blocking mode
2003.08.20 08:16:47 LOG7[1357:1084382400]: smtps started
2003.08.20 08:16:47 LOG5[1357:1084382400]: smtps connected from clienttest:1355
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): before/accept initialization
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 read client hello A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 write server hello A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 write certificate A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 write certificate request A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 flush data
2003.08.20 08:16:47 LOG7[1357:1084382400]: waitforsocket: FD=9, DIR=read
2003.08.20 08:16:47 LOG7[1357:1084382400]: waitforsocket: ok
2003.08.20 08:16:47 LOG5[1357:1084382400]: VERIFY OK: depth=1, /C=FR/ST=Ile-de-France/L=Paris/O=Example Company/CN=Certificate Authority Server/emailAddress=admin-ca@branch1.example.com
2003.08.20 08:16:47 LOG5[1357:1084382400]: VERIFY OK: depth=0, /C=FR/ST=Ile-de-France/L=Paris/O=Example Company/CN=testmail/emailAddress=testmail@branch1.example.com
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 read client certificate A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 read client key exchange A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 read certificate verify A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 read finished A

```
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 write change cipher spec A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 write finished A
2003.08.20 08:16:47 LOG7[1357:1084382400]: SSL state (accept): SSLv3 flush data
2003.08.20 08:16:47 LOG7[1357:1084382400]:    1 items in the session cache
2003.08.20 08:16:47 LOG7[1357:1084382400]:    0 client connects (SSL_connect())
2003.08.20 08:16:47 LOG7[1357:1084382400]:    0 client connects that finished
2003.08.20 08:16:47 LOG7[1357:1084382400]:    0 client renegotiatations requested
2003.08.20 08:16:47 LOG7[1357:1084382400]:    1 server connects (SSL_accept())
2003.08.20 08:16:47 LOG7[1357:1084382400]:    1 server connects that finished
2003.08.20 08:16:47 LOG7[1357:1084382400]:    0 server renegotiatiations requested
2003.08.20 08:16:47 LOG7[1357:1084382400]:    0 session cache hits
2003.08.20 08:16:47 LOG7[1357:1084382400]:    1 session cache misses
2003.08.20 08:16:47 LOG7[1357:1084382400]:    0 session cache timeouts
2003.08.20 08:16:47 LOG6[1357:1084382400]: Negotiated ciphers: AES256-SHA           SSLv3 Kx=RSA
Au=RSA  Enc=AES(256)  Mac=SHA1
2003.08.20 08:16:47 LOG7[1357:1084382400]: FD 10 in non-blocking mode
2003.08.20 08:16:47 LOG7[1357:1084382400]: smtps connecting 192.168.2.7:25
2003.08.20 08:16:47 LOG7[1357:1084382400]: remote connect #1: EINPROGRESS: retrying
2003.08.20 08:16:47 LOG7[1357:1084382400]: waitforsocket: FD=10, DIR=write
2003.08.20 08:16:47 LOG7[1357:1084382400]: waitforsocket: ok
2003.08.20 08:16:47 LOG7[1357:1084382400]: Remote FD=10 initialized
```

6.3.3 Invalid client certificate

For this test, we delete the certificate for the testmail user from the "trustedcerts" directory, which mean that we do not trust any longer the testmail user. As a consequence, he should not be authorized to open an SSL connection to the mail server, as the Stunnel daemon on the server side would not be able to verify his certificate.

```
$ tail –f /var/log/stunnel.log
2003.08.20 08:23:32 LOG7[1368:1075930272]: smtps accepted FD=9 from clienttest:1357
2003.08.20 08:23:32 LOG7[1368:1075930272]: FD 9 in non-blocking mode
2003.08.20 08:23:32 LOG7[1368:1084382400]: smtps started
2003.08.20 08:23:32 LOG5[1368:1084382400]: smtps connected from clienttest:1357
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): before/accept initialization
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): SSLv3 read client hello A
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): SSLv3 write server hello A
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): SSLv3 write certificate A
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): SSLv3 write certificate request A
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): SSLv3 flush data
2003.08.20 08:23:32 LOG7[1368:1084382400]: waitforsocket: FD=9, DIR=read
2003.08.20 08:23:32 LOG7[1368:1084382400]: waitforsocket: ok
2003.08.20 08:23:32 LOG5[1368:1084382400]: VERIFY OK: depth=1, /C=FR/ST=Ile-de-
France/L=Paris/O=Example Company/CN=Certificate Authority Server/emailAddress=admin-
ca@branch1.example.com
2003.08.20 08:23:32 LOG4[1368:1084382400]: VERIFY ERROR ONLY MY: no cert for /C=FR/ST=Ile-de-
France/L=Paris/O=Example Company/CN=testmail/emailAddress=testmail@branch1.example.com
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL alert (write): fatal: certificate unknown
2003.08.20 08:23:32 LOG3[1368:1084382400]: SSL_accept: 140890B2: error:140890B2:SSL
routines:SSL3_GET_CLIENT_CERTIFICATE:no certificate returned
2003.08.20 08:23:32 LOG7[1368:1084382400]: smtps finished (0 left)
```

6.4 Syslog configuration

We can check that the system properly logs messages from Postfix, IMAP, Stunnel in the adequate file.

```
$ more /var/log/maillog
Aug 20 00:14:55 mail postfix/smtpd[1417]: connect from client.example.com [10.0.0.5]
Aug 20 00:14:55 mail postfix/smtpd[1417]: > client.example.com[10.0.0.5]: 220 jupiter.example.com mail server
Aug 20 00:14:55 mail postfix/smtpd[1417]: watchdog_pat: 0x80785e0
Aug 20 00:15:11 mail postfix/smtpd[1417]: < client.example.com [10.0.0.5]: auth plain
dGVzdG1haWwAdGVzdG1haWwAdGVzdG1haWw=
Aug 20 00:15:11 mail postfix/smtpd[1417]: smtpd_sasl_authenticate: sasl_method plain, init_response
dGVzdG1haWwAdGVzdG1haWwAdGVzdG1haWw=
Aug 20 00:15:11 mail postfix/smtpd[1417]: smtpd_sasl_authenticate: decoded initial response testmail
Aug 20 00:15:11 mail postfix/smtpd[1417]: > client.example.com[10.0.0.5]: 235 Authentication successful
Aug 20 00:15:11 mail postfix/smtpd[1417]: watchdog_pat: 0x80785e0
Aug 20 00:16:10 mail postfix/postfix-script: refreshing the Postfix mail system
Aug 20 00:16:10 mail postfix/master[1284]: reload configuration
```

```
$ more /var/log/imap
Aug 28 01:13:56 mail master[2023]: about to exec /usr/local/cyrus/bin/imapd
Aug 28 01:13:56 mail imap[2023]: executed
Aug 28 01:13:56 mail imapd[2023]: accepted connection
Aug 28 01:13:56 mail imapd[2023]: login: client.example.com[10.0.0.5] testmail plaintext
Aug 28 01:13:56 mail imapd[2023]: seen_db: user testmail opened /var/imap/user/t/testmail.seen
Aug 28 01:13:56 mail imapd[2023]: accepted connection
Aug 28 01:13:56 mail imapd[2023]: login: client.example.com[10.0.0.5] testmail plaintext
Aug 28 01:13:56 mail imapd[2023]: open: user testmail opened INBOX
Aug 28 01:14:56 mail master[1610]: process 2023 exited, status 0
```

```
$ more /var/log/stunnel.log
2003.08.20 08:23:32 LOG7[1368:1075930272]: smtps accepted FD=9 from clienttest:1357
2003.08.20 08:23:32 LOG7[1368:1075930272]: FD 9 in non-blocking mode
2003.08.20 08:23:32 LOG7[1368:1084382400]: smtps started
2003.08.20 08:23:32 LOG5[1368:1084382400]: smtps connected from clienttest:1357
2003.08.20 08:23:32 LOG7[1368:1084382400]: SSL state (accept): before/accept initialization
```

6.5 Tripwire

For testing the Tripwire integrity check, we modify a “crucial” file like for example /etc/postfix/main.cf by adding a new parameter. The output below shows that Tripwire detects correctly the modification to the configuration file of the Postfix server.

```
<…skip…>
-------------------------------------------------------------------------------
  Section: Unix File System
-------------------------------------------------------------------------------
-
  Rule Name                       Severity Level    Added    Removed
Modified
  ---------                       --------------    -----    -------  ------
--
```

```
  Invariant Directories          66              0        0        0
  Temporary directories          33              0        0        0
  Tripwire Data Files            100             0        0        0
  Critical devices               100             0        0        0
  User binaries                  66              0        0        0
  Tripwire Binaries              100             0        0        0
* Critical configuration files   100             0        0        1
  Libraries                      66              0        0        0
 <…skip…>
```

6.6 Scan of the system

Before putting our system in production, we have to be sure that only the adequate ports are opened. This enables to validate the configuration of the host-based firewall IP-table.

6.6.1 Production network

A scan from the Public DMZ production network provide the following output.

```
$ nmap -sS 192.168.2.7

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2003-11-25 18:53 CET
Interesting ports on jupiter@example.com (192.168.2.7):
(The 1650 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
465/tcp open  smtps
993/tcp open  imaps
995/tcp open  pop3s

Nmap run completed -- 1 IP address (1 host up) scanned in 55.792 seconds
```

The scan reveals that only the ports enabling SSL to mail services SMTP, IMAP and POP are opened, which is the expected behavior of the server on the production network.

It is important to note that the standard port 25 for SMTP is not shown here. Indeed, only the corporate mail relay Mercury 192.168.2.5 can initiate SMTP connection towards our mail server. As a consequence, a scan launch from any IP in the Public DMZ production network is blocked by the host-based firewall.

6.6.2 Management network

A scan from the corporate management network reveals that only the ssh server is listening on this interface.

```
$ nmap -sS 192.168.10.7

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2003-11-25 19:05 CET
Interesting ports on jupiter@example.com (192.168.2.7):
(The 1650 ports scanned but not shown below are in state: closed)
```

```
PORT    STATE SERVICE
22/tcp  open  ssh

Nmap run completed -- 1 IP address (1 host up) scanned in 45,92 seconds
```

7 Conclusion

We present in this document a proper way to set an effective solution to provide remote users with access to their messaging resources. We use a Linux RedHat 9 distribution for the operating system. In addition, we select core applications: Cyrus IMAP server, Postfix and Stunnel that are known to be robust and deployed successfully at a large scale.

We pay particular attention to the way users are authenticated by the system. The first level is providing using digital certificate and the second level relies on traditional user and password. We use a central repository for user passwords storage that is used for access IMAP and SMTP resources. The protection of the data sent over the network is performed using Secured Socket Layer to provide encryption.

Annex A: Cyrus IMAP configuration details

```
# more /etc/cyrus.conf
# standard standalone server implementation

START {
  # do not delete this entry!
  recover       cmd="ctl_cyrusdb -r"

  # this is only necessary if using idled for IMAP IDLE
#  idled                cmd="idled"
}

# UNIX sockets start with a slash and are put into /var/imap/sockets
SERVICES {
  # add or remove based on preferences
  imap          cmd="imapd" listen="imap" prefork=0
  pop3          cmd="pop3d" listen="pop3" prefork=0

  # LMTP is required for delivery
  lmtpunix      cmd="lmtpd" listen="/var/imap/socket/lmtp" prefork=0

  # this is only necessary if using notifications
#  notify       cmd="notifyd" listen="/var/imap/socket/notify" proto="udp"
prefork=1
}

EVENTS {
  # this is required
  checkpoint    cmd="ctl_cyrusdb -c" period=30

  # this is only necessary if using duplicate delivery suppression
  delprune      cmd="ctl_deliver -E 3" at=0400

  # this is only necessary if caching TLS sessions
  tlsprune      cmd="tls_prune" at=0400
}
```

Annex B: OpenSSL configuration file

```
# This definition stops the following lines choking if HOME isn't
# defined.
HOME                    = /
RANDFILE                = $ENV::HOME/.rnd

# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids

# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir             = /ca           # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions        = crl_ext

default_days    = 365                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
```

```
default_md      = md5                   # which md to use.
preserve        = no                    # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
commonName              = supplied
emailAddress            = supplied

# For the example.com policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_example ]
countryName             = supplied
stateOrProvinceName     = supplied
localityName            = supplied
organizationName        = supplied
commonName              = supplied
emailAddress            = supplied

####################################################################
[ req ]
default_bits            = 1024
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
#attributes             = req_attributes
x509_extensions = v3_ca # The extentions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or UTF8Strings
# so use this option with caution!
string_mask = nombstr

# req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = FR
countryName_min                 = 2
countryName_max                 = 2
```

```
stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Ile-de-France

localityName                    = Locality Name (eg, city)
localityName_default            = Paris

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = Example Company

commonName                      = Common Name (eg, your name or your
server\'s h
ostname)
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_max                = 64

# SET-ex3                       = SET extension number 3

#[ req_attributes ]
#challengePassword              = A challenge password
#challengePassword_min          = 4
#challengePassword_max          = 20

#unstructuredName               = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType                    = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
```

```
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]


# Extensions for a typical CA


# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy
```

```
# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6
```

Annex C: Tripwire Policy example

Tripwire Policy file

```
# Tripwire Policy file

################################ #
#                              # #
# Critical configuration files # #
#                              ##
################################
(
  rulename = "Critical configuration files",
  severity = $(SIG_HI)
)
{
     /etc/crontab                      -> $(SEC_BIN) ;
     /etc/cron.hourly                  -> $(SEC_BIN) ;
     /etc/cron.daily                   -> $(SEC_BIN) ;
     /etc/cron.weekly                  -> $(SEC_BIN) ;
     /etc/cron.monthly                 -> $(SEC_BIN) ;
     /etc/default                      -> $(SEC_BIN) ;

<…skip…>

################### #
#                 # #
# Critical devices # #
#                 ##
###################
(
  rulename = "Critical devices",
  severity = $(SIG_HI),
  recurse = false
)
{
     /dev/kmem                         -> $(Device) ;
     /dev/mem                          -> $(Device) ;
     /dev/null                         -> $(Device) ;
     /dev/zero                         -> $(Device) ;
     /proc/devices                     -> $(Device) ;
```

Tripwire configuration file

```
ROOT                 =/usr/sbin
POLFILE              =/etc/tripwire/tw.pol
DBFILE               =/var/tripwire/jupiter.twd
REPORTFILE           =/var/tripwire/report/jupiter-$(DATE).twr
SITEKEYFILE          =/etc/tripwire/tw-site.key
LOCALKEYFILE         =/etc/tripwire/tw-local.key
EDITOR               =/bin/vi
LATEPROMPTING        =false
LOOSEDIRECTORYCHECKING =false
MAILNOVIOLATIONS     =true
EMAILREPORTLEVEL     =3
REPORTLEVEL          =3
MAILMETHOD           =SENDMAIL
```

```
SYSLOGREPORTING         =false
MAILPROGRAM             =/usr/sbin/sendmail -oi –t
```

References

[1] Red Hat: Red Hat Linux GPG keys
http://www.redhat.com/solutions/security/news/publickey.html

[2] Red Hat: Red Hat Linux Reference Guide
http://www.europe.redhat.com/documentation/rhl9/rhl-rg-en-9/

[3] Red Hat: Red Hat errata web site for Red Hat Linux 9
https://rhn.redhat.com/errata/rh9-errata-security.html

[4] Red Hat: Red Hat Linux 9 updates
http://updates.redhat.com/9/en/os/

[5] SANS institute: secure RedHat Linux SANS GCUX practicum

[6] Carnegie Mellon University: Cyrus SASL for system administrators
http://asg.web.cmu.edu/cyrus/download/sasl/sysadmin.html

[7] Postfix.org: Postfix basic configuration guide
http://www.postfix.org/basic.html

[8] Red Hat: SMTP authentication using Postfix and SASL
http://postfix.state-of-mind.de/patrick.koetter/smtpauth

[10] Carnegie Mellon University: Cyrus IMAP installation guide
http://asg.web.cmu.edu/cyrus/download/imapd/install.html

[11] Stunnel.org: using Stunnel with certificates
http://www.stunnel.org/faq/certs.html

[12] Carnegie Mellon: Software Engineering Institute
http://www.cert.org

[13] SANS Institute : InternetStormCenter
http://isc.sans.org/