

Table of contents

1. Introduction	3
2. How to navigate through this guide	5
3. NOD32LS package installation	7
4. NOD32LS base configuration	11
5. NOD32LS as for Linux Mail Servers	15
5.1. Mail server in UNIX OS environment	16
5.2. Scanning of the inbound e-mail messages	17
5.2.1. Renaming the original MDA and its replacement by NOD32MDA	17
5.2.2. Setting of NOD32MDA (in MTA) as MDA	18
5.2.2.1. Setting Sendmail MTA	18
5.2.2.2. Setting Postfix MTA	19
5.2.2.3. Setting Qmail MTA	20
5.2.2.4. Setting MTA Exim version 3	21
5.2.2.5. Setting MTA Exim version 3 (more general)	22
5.2.2.6. Setting MTA Exim version 4	23
5.3. Scanning the outbound e-mail messages	24
5.4. Content filtering in MTA	26
5.4.1. Content filtering in MTA Postfix	26
5.4.2. Content filtering in MTA Sendmail	26
5.4.3. Content filtering in MTA Exim	27
5.4.4. Content filtering in MTA Qmail	28
5.5. Alternative methods of scanning e-mails	28
5.5.1. Scanning e-mail messages using AMaViS	28
5.5.1.1. amavis	28
5.5.1.2. amavisd	29
5.5.1.3. amavisd-new	29
6. NOD32LS as for Linux File Servers	31
6.1. On-demand scanner	32
6.2. On-access scanner	32
6.2.1. On-access scanner powered by Dazuko	33
6.2.1.1. Operation principle	33
6.2.1.2. Installation and configuration	33
6.2.1.3. Tips	34
6.2.2. On-access scanner using preload libc library	34
6.2.2.1. Operation principle	36
6.2.2.2. Installation and configuration	36
6.2.2.3. Tips	36
7. Sample submission system	37
8. NOD32 system update and maintenance	39
8.1. Basic concept of NOD32 system update	40
8.1.1. NOD32 modules organization	40
8.1.2. NOD32 mirror creation	40
8.1.3. Generation of NOD32 scanner loading modules	40
8.2. Subordinate mirrors creation	41
8.3. Automatic update of the virus definitions database	42
8.3.1. Structure and use of automatic update script	42
8.3.2. Periodic update of the virus definitions database	42
9. Tips and tricks	43
9.1. Dropping messages marked by NOD32 as deleted in MTA Postfix	42
9.2. NOD32LS and TLS support in MTA	42
10. Let us know	45
A. Installed content of NOD32LS package	47

NOD32 for Linux Server, First Edition

Published on 25th September 2005,

Last revised on 3th July 2006

Copyright © 2005 Eset, spol. s r. o.

All rights reserved.

NOD32 for Linux Server was developed by Eset, s. r. o. For more information visit www.nod32.com.

NOD32 for Linux Server was developed in co-operation with ProWeb Consulting. For more information visit www.pwc.sk.

All rights reserved. No part of this documentation may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise without a permission in writing from the author.

Eset, s. r. o. reserves the right to change any of the described application software without prior notice.

Revision History

Revision 2.51-1 (09/01/2006)

Revision 2.50-1 (01/01/2005)

Authorized transcription of NOD32 for Linux Servers (version 2.50-1) user's guide.

Eset, spol. s r. o.

Svoradova 1, 811 03 Bratislava, Slovak Republic

<http://www.eset.sk/en>

Chapter 1:

Introduction

Dear user, you have acquired NOD32 for Linux Servers – NOD32LS – probably the best antivirus system for servers running under the Linux operating system. As you will soon realize, the NOD32LS system using the state-of-the-art NOD32 scanning engine has unsurpassed scanning speed and detection rate, combined with a very small footprint that makes it the ideal choice for any Linux OS server.

The current version of the NOD32LS is, from the point of view of its functionality, compound from so-called NOD32 for Linux Mail Servers part (LMS) and from so-called NOD32 for Linux File Servers part (LFS). The following is a short summary of key features of the system.

NOD32 scanning engine algorithms provide both the highest detection rate and the fastest scanning times. The system is developed to run on the single-processor units as well as on the multi-processor units. It includes unique advanced heuristics for Win32 worms and back-doors. Inbuilt NOD32 archivers unpack archived objects without the need for any external programs. In order to increase speed and efficiency of the system, its architecture is based on the running daemon (resident program) where all the scanning requests are sent to. Six various levels of logging can be configured to get information about system activity and infiltrations. One of the major advantages is the fact that the system installation does not require external libraries or programs except for libc. The system can be configured to notify any person in case of detected infiltration. Moreover, information about infiltration can be configured to be written into an e-mail header, footer and subject (LMS only). The system (LMS only) is MTA-independent, i.e. the NOD32 for Linux Mail Servers solution does not depend on Mail Server used. The system (LMS) can be configured in order to define setting of the scanner individually for e-mail recipient address (resp. recipient domain address) for user whom e-mail is locally delivered.

To run efficiently, the NOD32LS requires just 11MB of hard-disk space and 8MB of RAM. The system runs smoothly under the 2.2.x, 2.4.x and 2.6.x Linux OS kernel versions. The system (LMS only) supports most popular server software including Sendmail, Postfix, Qmail, Exim, etc.

The NOD32LS runs under all of the common Linux distributions:

Fedora Core 1 and higher, Red Hat version 5.x and higher, Mandrake version 5.x and higher, Mandriva version 2005 and higher, 1 Chapter 1. Introduction SuSE version 6.x and higher, Debian version 2.x and higher. Ubuntu version 4.x and higher.

From lower-powered, small office servers to enterprise-class ISP servers with thousands of users, NOD32LS delivers the performance and scalability you expect from a UNIX based solution and the unequalled security of NOD32.

Chapter 2:

How to navigate through this guide

This guide is assumed to be the complex users' guide into the NOD32LS system. It covers information on configuration and maintenance of the system in order to run efficiently for various supported Linux OS distributions and various e-mail server systems.

Therefore, assuming you are a typical system administrator being too busy to read all these pages, here are some hints on how to surf through this guide with just catching the most relevant information necessary to setup and run.

First, look into the foreword to each chapter to get information related to relevance of the information stored in the chapter. Often you will get a hint to skip directly to the section that contains just the step by step statement information in order to suffice what is necessary.

Second, besides the information in this guide, there exists manual pages related to the individual components of the NOD32LS package. The pages are always a useful reference for getting actual information on the accurate components setting options that are not all mentioned here.

Third, note that the guide is divided in a following manner:

- Chapter 1 Introduction.
- Chapter 2 This text, that you have almost finished reading.
- Chapter 3 Leads you step by step to install the system.
- Chapter 4 Lists common configuration of the NOD32LS system.
- Chapter 5 Lists Linux Mail Server specific configurations.
- Chapter 6 Lists Linux File Server specific configurations.
- Chapter 7 Contains description of so-called samples submission system (Charon).
- Chapter 8 Will help you keep your system up to date.
- Chapter 9 Contains tips and tricks on configuration of NOD32LS.

Chapter 3:

NOD32LS package installation

Before further explanations concerned with NOD32LS, let's first install the whole thing. In order to do so, one has to download the appropriate packages from the NOD32 server. Use your favorite web browser to navigate to the NOD32 download page

```
http://www.nod32.com/download/download.htm
```

At this page you can see a set of NOD32LS packages listed for various UNIX OS distributions. Identify the appropriate distribution of the actual OS installed on your computer and download the appropriate package.

Note: For the download process to succeed, the user name and password for authentication against the NOD32 server is required. You will receive both from your vendor.

After successful download the binary file

```
nod32ls-x.xx-x.i386.ext.bin
```

is present within your download directory, where the 'ext' is the OS distribution dependent suffix of the package. To extract the installation package from this file one has to write the following statement

```
sh ./nod32ls-x.xx-x.i386.ext.bin
```

within the directory where the package file has been downloaded. As a result the User License Acceptance Agreement related with the product will be shown. It is the decision of the user whether agree or not with the acceptance, however, the installation package will be extracted into the current working directory only in case the acceptance has been confirmed.

Before further steps it is yet necessary to provide license file that has been obtained from vendor within the directory

```
/etc/nod32/license
```

i.e. if this is the first time installation, it is necessary to create this directory and copy the '*.lic' license file into it.

After successful extraction of the package file the whole installation relies on one command line statement. In case the OS distribution installed at your computer is one of the following (Fedora, Red Hat, SuSE, Mandrake, Mandriva or other Linux OS using Red Hat package manager) you need to write the statement:

```
rpm -i nod32ls-x.xx-x.i386.rpm
```

In case the NOD32LS is already installed on your computer you will receive the following or similar message in return.

```
package nod32ls-x.xx-x.i386.rpm is already installed
```

In this case you have to use a different switch -u to update the appropriate rpm package, so the statement will look like this

```
rpm -u nod32ls-x.xx-x.i386.rpm
```

The Debian Linux OS uses its own package manager for installation, so called dpkg (debian packager) and the files are also of different format (suffix deb). So the appropriate statement for installation and/or update of the package in this case will be as follows

```
dpkg -i nod32ls-x.xx-x.i386.deb
```

Once the NOD32LS package is installed, one can see at least two main NOD32LS scanning daemons nod32d running in the background. One of the daemons serves as the process and threads manager. The rest are responsible for objects scanning. The installation apparatus is done in a way that it starts the daemons automatically. One can also see that from now the nod32d daemons will also start immediately after a computer boot. To be absolutely certain, the fact that the daemon is running can be checked by typing the command

```
ps -C nod32d
```

that should return the following or similar message

PID	TTY	TIME	CMD
2226	?	00:00:00	nod32d
2229	?	00:00:00	nod32d

If this is not the case, then something is wrong and you should report this as a bug to the NOD32 support service. The most common reason for not proper daemon initialization will be missing license file.

As you will soon realize, there are also other daemons within a NOD32LS package that are required to run in some system configurations. For instance the daemon `nod32smtp` or `nod32smfi`. However, since the functionality of these daemons is not required in all cases, the installation script does not provide the automatic start-up of them at system boot. If necessary one has to do it by hand. Here is the description of how to do this.

In order to start, for instance, `nod32smtp` daemon under Red Hat or Mandrake, one has to write the following command

```
/sbin/service nod32smtp start
```

Under SuSE the appropriate command would be

```
/sbin/startproc /usr/bin/nod32smtp
```

Finally, for the Debian the equivalent command is

```
/sbin/start-stop-daemon --start --exec /usr/sbin/nod32smtp
```

Similarly, for stopping the appropriate service, one would write under RedHat or Mandrake the following command

```
/sbin/service nod32smtp stop
```

Under SuSE the appropriate command would be

```
/sbin/killproc /usr/bin/nod32smtp
```

and for Debian the equivalent command is

```
/sbin/start-stop-daemon --stop --name nod32smtp
```

Now we will describe how to provide the automatic start of the daemon at system boot. In order to do so, one has to write on Red Hat or Mandrake system the following commands

```
/sbin/chkconfig --add nod32smtp
```

On SuSE Linux the following command will be valid

```
/sbin/insserv /etc/init.d/nod32smtp
```

and finally for the Debian the statement

```
/usr/sbin/update-rc.d nod32smtp defaults
```

can be used. After using these commands, the system will create all the necessary settings for the automatic start of the `nod32smtp` daemon at system boot.

However, we still have to discuss one circumstance about the NOD32LS package installation. In case none of the above OS distributions correspond to your actual operating system, you can try to install NOD32LS from the `tgz` (tarred and gzipped) format. The rest of this chapter is a step by step guide to do so, therefore if this is not relevant to you, skip directly to the next section. In order to install NOD32LS from a generic `tgz` file, unpack the file `nod32ls-x.xx-x.i386.lnx.tgz` to the root directory:

```
tar -xvzf nod32ls-x.xx-x.i386.lnx.tgz -C /
```

See appendix A for complete NOD32LS package content installed at the proper location.

Note: After copying the NOD32LS components you have to adjust the initialization scripts for the daemon programs of NOD32LS. In the `/etc/init.d` directory you should see a short scripts, corresponding to the individual NOD32LS components and OS distributions by their names. For instance

```
/etc/init.d/nod32d.deb
```

is the initialization script for `nod32d` daemon used for Debian Linux distribution. Try adjusting one of the scripts for each daemon in order to make it able to start.

Chapter 4:

NOD32LS base configuration

All the necessary configuration files concerned with functionality of NOD32LS are stored within a directory `/etc/nod32`

In the following text all the files stored within this directory are going to be discussed.

`/etc/nod32/nod32.cfg`

This is a most important file, commonly known also as 'main NOD32 configuration file'. After exploring the file you can see that its structure is composed from various parameters of NOD32LS that are all distributed within sections (note the section names are always enclosed in square brackets – []). There is always one global and several special sections in the file. The configuration of the whole system (i.e. key features discussed within Introduction to this guide and much more) is provided by setting parameters used within individual sections. The parameters defined within a global section are valid for all modules (sometimes we call those modules 'agents') of NOD32LS. The parameters defined within a special section (also called 'agent' section), on the other hand, are valid only for an appropriate agent of NOD32LS. At this point we should describe the individual parameters meaning, however, we will not do it here as it is done in manual pages corresponding to the appropriate module. For instance the parameters that can be used in global and also in special sections are described in the manual page concerned with this file (`nod32.cfg(5)`). The manual page can be invoked using the following statement

```
man nod32.cfg
```

Similarly, the parameters that can be only used in the special sections are described in the manual pages concerned with individual modules, i.e. `nod32mda`, `nod32smtp`, `nod32smfi`, `nod32cli`, `nod32pipe`, `nod32pac`.

We recommend user to use these resources always when referring to the parameters of NOD32LS components. Refer to the appendix A to see a full list of manual pages installed.

Particularly, concerning main NOD32 configuration file we recommend user to read manual page `nod32.cfg(5)` in order to get detailed information on organization and philosophy of NOD32LS configuration.

`/etc/nod32/nod32.auth`

This is the so called 'user authentication file' that is used to authenticate the user against the NOD32 server when making an update of the virus signatures database. We will not discuss the meaning of the file here as it is done later in chapter 8 with more details.

`/etc/nod32/sig_*.html.example`

When browsing through the `/etc/nod32` directory you have probably noticed a set of files whose names are starting with same prefix 'sig' and ending with suffix 'html.example'. These are the html templates that can be used to define the text, inserted as a footnote, into the scanned e-mail messages (use your favorite editor to explore the files). In order to enable this feature, you have to set an appropriate option in the main configuration file (`write_to_emails`) that can be defined in a global section of the file as well as in the individual modules section of the file depending on what level of footnote writing we would like to achieve in appropriate individual modules (see `nod32d` manual page for detail). The other requirement to enable the custom html templates is to remove the suffix 'example' from all the file names of the templates. In this case the NOD32LS system will ignore the default footnote text and will use the text predefined in the html files instead. The meaning of the individual files can be explained directly in the structure of e-mail footnotes inserted. When NOD32 system has detected infiltration in the e-mail message, the following footnote template is written into the footnote when enabled.

```
e-mail header | From:
.             | To:
-----
e-mail body   |
.             |
.             | html text of the file sig_header_infected.html
.             | list of infiltrations found by the scanner
.             |
.             |
.             | html text of the file sig_footer_infected.html
```

When NOD32 system has not detected any infiltration in the e-mail message, the following footnote template is written into the footnote when enabled.

```
e-mail header | From:
.             | To:
-----
e-mail body   |
.             |
.             | html text of the file sig_header_clean.html
.             |
.             | list of objects scanned by the scanner
.             |
.             |
.             | html text of the file sig_footer_clean.html
```

When NOD32 system could not scan the complete e-mail message, the following footnote template is written into the footnote when enabled.

```
e-mail header | From:
.             | To:
-----
e-mail body   |
.             |
.             | html text of the file sig_header_not_scanned.html
.             |
.             | list of object scanned by the scanner
.             |
.             |
.             | html text of the file sig_footer_not_scanned.html
```

/etc/nod32/license

This is the license directory where all licenses related with individual products are stored. Before NOD32 scanner initialization daemon will always check the licenses found here and based on this information the appropriate part of the NOD32LS system will be enabled.

/etc/nod32/nod32d_license_warning_script

This script (if an appropriate functionality enabled) is executed last 30 days (once per day) before product license expiration. The script is used to send notification e-mail about the expiration status to a system administrator.

/etc/nod32/nod32d_script

There is one more file within /etc/nod32 directory. It is a NOD32LS notification script. In case NOD32 scanner has found an infiltration within a scanned message, this script (if enabled) is used to send a notification about the event to any person defined within a script. This functionality is enabled by logical parameter of the main configuration file (exec_script = yes). To get more information on this topic, read the section NOTIFICATION within the nod32d manual pages.

Chapter 5:

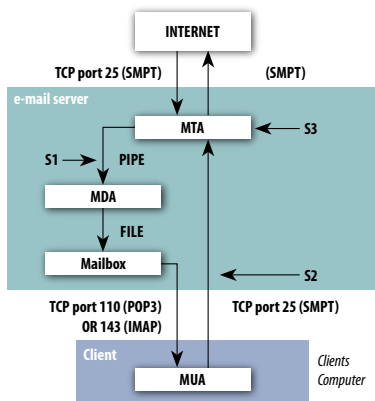
NOD32LS as for Linux Mail Servers

This chapter describes the process of the NOD32LMS part configuration.

5.1. Mail server in UNIX OS environment

This chapter is concerned with the basics of the e-mail messaging system, also commonly called e-mail server system, however, e-mail server is only part of the more complex messaging system. For better understanding of the NOD32LS operation, knowledge of the messaging system basic principles is of paramount importance. Therefore we do not recommend to skipping this section unless this knowledge is already acquired. The following diagram is a rough scheme of the UNIX OS e-mail messaging system.

Figure 5-1. Scheme of UNIX OS e-mail messaging system.



The meaning of abbreviations used in the scheme of figure 3-1 is as follows.

MTA (Mail Transport Agent)

A program (for instance sendmail, postfix, qmail, exim, etc.) receives e-mail messages from local and/or remote domains and forwards it for further delivery. Generally speaking, MTA is an agent providing mail transfer among other e-mail servers MTAs and/or MUAs (see below).

MDA (Mail Delivery Agent)

A program (maildrop, procmail, deliver, local.mail, etc.) providing delivery of an e-mail into a particular mailbox.

MUA (Mail User Agent)

An e-mail processing program (MS Outlook, Mozilla Mail, Eudora, etc.) that allows user to access and manage e-mail messages (i.e. read, compose, print them etc.).

MAILBOX

A file or a file structure on a disk serving as the storage space for e-mails.

Note: There are several formats of Mailboxes in Linux OS.

(e.g.: an old fashioned format where emails for each user are stored sequentially in one user appropriate file located in directory `/var/spool/mail`; MBOX (a bit newer but still an old format) with e-mails stored sequentially in one file located within user home directory; MAILDIR with e-mail stored in a separate file within a hierarchical directory structure.

Now the scheme in the figure 3-1 represents a typical e-mail gateway placed at an entrance to some local network. This means that the e-mail server receives data communication typically via TCP port 25 (SMTP – Simple Mail Transfer Protocol is used within this process). The message received is transferred by the local MTA either to another remote e-mail server system or the message is delivered by using local MDA into the appropriate MAILBOX (we assume that each user belonging to the local network has a corresponding MAILBOX located at the server). It is then a responsibility of the client's local MUA to provide download and/or correct interpretation of the message at the client's computer. To get the data from an e-mail server system the MUA uses typically TCP port 110 (POP3 – Post Office Protocol) or TCP port 143 (IMAP – Internet Message Access Protocol). On the other hand if a user at the client's computer would like to send an e-mail message to the Internet, it is again the responsibility of the local MUA to deliver the message via TCP port 25 (SMTP) to the local MTA (located at an entrance to the local network) that will take care of the further message delivery.

The operating principle of NOD32LS system is based on the idea of data communication interception at the various phases of its transfer and of scanning this communication by NOD32 scanning engine. Those locations are marked in the

figure 3-1 by symbols S1, S2 and S3. In the following text we will distinguish between three scenarios of e-mail message scanning which basically corresponds to the referred marks:

- * Scanning of inbound e-mail messages (We define the term „inbound message“ for e-mail message with the target address corresponding to the destination located at the local domain. Similarly the „outbound message“ will be a message bound to some remote domain via its target address.) marked in the figure by symbol S1, is used to protect e-mail messages delivered from the outside Internet to the local MAILBOX-es belonging to local users.
- * Scanning of outbound messages marked in the figure by symbol S2 is used to protect the e-mail messages sent by local user MUAs to the outside Internet.
- * Bi-directional scanning (usually known also as content filtering in MTA) marked by symbol S3 is devoted to check both directions (or even all directions) of the e-mail messages flow.

Note: Bi-directional scanning scenario is more complex than the other two methods previously discussed. The bi-directional scanning checks not only message flow directed from Internet to a 13 Chapter 5. NOD32LS as for Linux Mail Servers . local network or vice versa. Indeed, this scanning also takes care of the messages that will come to the gateway server and are further routed to the different remote domains. Therefore in a general case this kind of scanning can be used typically as a checkpoint between two different Internet networks. From this point of view, concerning our local e-mail gateway, this kind of scanning is not necessarily required, as this creates a much higher load on the server computer than previously discussed scenarios.

5.2. Scanning of the inbound e-mail messages

Scanning of the inbound e-mail messages is performed at the time of message transmission between MTA and MDA (as marked by symbol S1 in figure 3-1). The generic scheme of the process is shown in the figure 5-1.

The incoming e-mail is scanned by nod32mda module supported by daemon nod32d. Scanned e-mail is afterward forwarded to the original MDA for further delivery to the MAILBOX.

As shown in the figure 5-1, the virus scanning feature can be enabled by appropriate configuration setting of the MTA agent and nod32mda module. It is also apparent from the figure that the solution is MDA independent.

Note: The majority of mail servers use procmail or maildrop (MDA). The nod32mda module supports any MDA. The following MDAs were tested: procmail, maildrop, deliver and local. mail.

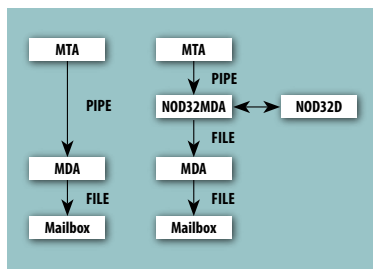
In general, the virus scanning feature can be enabled in two ways described below.

5.2.1. Renaming the original MDA and its replacement by NOD32MDA

This is a simple approach even without a need to make any changes in the agent MTA. But still, prior starting with the proper setup modification, the user is required to know exactly what MDA agent is used by the system. This information can be grabbed only by exploring the MTA configuration file. For illustration purposes, we will use the MDA procmail.

In the next step we have to find the location of the MDA. In order to do so we use the following command:

Figure 5-2. The scheme of the inbound e-mail message delivery without (left part of the figure) and with (right part of the figure) the NOD32 scanning.



which `procmail`

that will give as the full path to agent MDA in return. Now the most important part of the modification is to rename the original `procmail` binary file to `procmail.real`:

```
mv /usr/bin/procmail /usr/bin/procmail.real
```

and create the soft link to module `nod32mda` with the name 'procmail' located in the directory where the original MDA `procmail` was found, i.e. with the name '/usr/bin/procmail':

```
ln -s /usr/bin/nod32mda /usr/bin/procmail
```

With the above modifications, we have ensured that all the messages originally sent by the MTA for delivery to MAILBOX (by using an original MDA) will now be primarily sent to module `nod32mda`. Still there remains the second part of the modification to provide that all messages processed by `nod32mda` will be sent to the original `procmail` MDA (at this time renamed to `procmail.real`). In order to do so, we have to modify parameter 'mda_path' within the section [mda] of the main NOD32LS configuration file (/etc/nod32/nod32.cfg) as follows:

```
mda_path = "/usr/bin/procmail.real"
```

After all of the above modifications one has to restart the daemon `NOD32d` in order to reread the new configuration of `NOD32LS`.

Note that in the above example we have used `procmail` as an MDA, however, the whole procedure can be done with any other known MDA which is a great advantage of this approach. On the other hand, there is a potential problem within this configuration that may occur in circumstances of MDA software upgrade. Indeed, once we will decide to upgrade the MDA software (for instance `procmail` in our case), the upgrading mechanism will provide the new binary file `/usr/bin/procmail` that will cancel the link to `nod32mda` scanning module created by the above procedure and the configuration will be broken. Thus, even if the approach suggested in this section works in all cases, it is always necessary keep in mind that the problem of the link cancellation may occur and one has always, after the MDA software upgrade, to repeat the whole procedure described above.

5.2.2. Setting of NOD32MDA (in MTA) as MDA

This section contains a more rigorous approach to provide scanning of inbound messages than the one described in the previous section. Even if the approach described here requires modification of the MTA configuration, we recommend it due to the fact that it does not bring the potential problems, described at the end of the previous section, to the system.

Note: Some MTA modules may be configured to not need an MDA component to deliver emails. In such cases, it may be necessary to configure the MTA to use any, by `NOD32LS` supported, external MDA (for instance `procmail` or `maildrop`).

5.2.2.1. Setting Sendmail MTA

An MTA `Sendmail` stores all its configurations in one file `/etc/mail/sendmail.cf` (in older operating systems it can be stored in `/etc/sendmail.cf`). The procedure described here will therefore be done only with this one file and of course with the main `NOD32LS` configuration file (`/etc/nod32/nod32.cfg`).

First one has to grab information on the MDA used by `Sendmail`. This can be found in the `Sendmail` configuration file (`/etc/mail/sendmail.cf`) in the section called 'Local and Program Mailer specification' at the parameter 'P' (in the sentence starting with word 'Mlocal'). Yet, before performing any changes to the `Sendmail` configuration file, please, make a back-up of this file (for example with the name `sendmail.cf.orig`). Now let's return to the description of the `sendmail.cf` parameters. The parameter 'P' represents the full path to the MDA used. In order to involve 'nod32mda' module into the message processing, one has to replace this path by the appropriate 'nod32mda' module path (i.e. /

usr/bin/nod32mda) . At the same time the appropriate change has to be done for parameter 'A' (located at the same sentence of Sendmail configuration file). Note that the 'A' parameter contains the file name of the original MDA component followed by some switches. These switches represent macros used by Sendmail when processing a message. Therefore none of the switches may be changed when modifying the name of the original MDA component at the parameter 'A'. For instance, if the original MDA component path is /usr/bin/procmail, the appropriate sentence in the Sendmail configuration file can look like this.

```
Mlocal, P=/usr/bin/procmail
      F=lsDFMAw5:|@qSPhnu9
      S=EnvFromL/HdrFromL
      R=EnvToL/HdrToL
      T=DNS/RFC822/X-Unix
      A=procmail -t -Y -a $h -d $u
```

After the modifications have been performed, the appropriate sentence will look like this.

```
Mlocal, P=/usr/bin/nod32mda
      F=lsDFMAw5:|@qSPhnu9
      S=EnvFromL/HdrFromL
      R=EnvToL/HdrToL
      T=DNS/RFC822/X-Unix
      A=nod32mda -t -Y -a $h -d $u
```

Note: Please, in case you are reading the ASCII form of this guide, do not drag and drop the above sentence, since it may not work. Indeed, it can be that the switches used here as an example will not work in your case as they are dependent on the version of Sendmail MTA.

With the above modifications we have ensured that all the messages originally sent by the MTA for delivery to the MAILBOX (by using an original MDA) will now be primarily sent to module nod32mda.

Note that in case user would like to pass command line parameter to the nod32mda agent it is necessary to define the parameter A in different way. Agent nod32mda will accept command line parameters only in case the sentence '--' is added before them. So for instance to pass command line option --user the nod32mda agent the A parameter of the sendmail configuration file has to be defined as follows.

```
A=nod32mda -t -Y -a $h -d $u -- --user $u
```

In this case agent 'nod32mda' will instruct daemon 'nod32d' to search special section [user] within the configuration file.

Yet remains the second part of the modification to provide that all messages processed by nod32mda will be sent to the original MDA. In order to do so, we have to modify the parameter 'mda_path' within the section [mda] of the main NOD32LS configuration file (/etc/nod32/nod32.cfg) to the original MDA component. The original MDA component can be, at this stage of the procedure, still found in the back-up file (/etc/mail/sendmail.cf.orig) you have created. Thus assuming your original MDA component is (/usr/bin/procmail) the appropriate parameter of the main NOD32LS configuration file will look like this:

```
mda_path = "/usr/bin/procmail"
```

Note (for Debian users): The Debian distribution uses the following MDA: /usr/lib/sm.bin/sensible-mdm. Note that this is not a full-blown MDA, it is rather a wrapper. However, the mda_path, in this case has the following format:

```
mda_path = "/usr/lib/sm.bin/sensible-mdm"
```

To accomplish the whole procedure, one has to restart both the MTA Sendmail and the daemon nod32d.

5.2.2.2. Setting Postfix MTA

An MTA Postfix stores its configuration in several files. However, the modifications necessary to be described in this section will be done only with the main Postfix configuration file (`/etc/postfix/main.cf`) and of course with the main NOD32LS configuration file (`/etc/nod32/nod32.cfg`).

First one has to grab information on the MDA used by Postfix. This can be found by exploring the main Postfix configuration file (`/etc/postfix/main.cf`). To make all things easier, the authors of Postfix have written a helper utility to work with this file, so called 'postconf', so all the necessary steps will be done by using this utility. Also, before any further steps, please make a back-up of the original main Postfix configuration file (for instance, with the name `/etc/postfix/main.cf.orig`).

Now let's assume that the original MDA component is `/usr/bin/maildrop`. Then by using a command `postconf mailbox_command` we will receive the following information on return:

```
mailbox_command = maildrop -d "$USER" "$EXTENSION"
```

In the next step we will use the 'which' command `which maildrop`

that will give us the full path to the binary file by return. Now, in order to involve module `nod32mda` into the message processing, we have to replace the MDA `maildrop` with the module `nod32mda`. Write the following command in order to do so:

```
postconf -e 'mailbox_command = nod32mda -d "$USER" "$EXTENSION"
```

Note: Please keep in mind that the switches used in 'mailbox_command' after the MDA component must stay the same as in the original configuration file, otherwise the current configuration may not work properly.

Note that in case user would like to pass command line parameter to the `nod32mda` agent it is necessary to define `mailbox_command` parameter in a different way. Agent `nod32mda` will accept command line parameters only in case the sentence `'--'` is added before them. So for instance to pass command line option `--user` the `nod32mda` agent the `mailbox_command` parameter has to be defined as follows.

```
postconf -e 'mailbox_command = nod32mda -d "$USER" "$EXTENSION" -- --user "$USER"
```

In this case agent `nod32mda` will instruct daemon `nod32d` to search special section `[user]` within the configuration file.

With the above modifications we have ensured that all the messages originally sent by MTA for delivery to MAILBOX (by using an original MDA) will now be primarily sent to module `nod32mda`. Still there is the second part of modification to provide that all messages processed by `nod32mda` will be sent to the original MDA. In order to do so, we have to modify the parameter `mda_path` within the section `[mda]` of the main NOD32LS configuration file (`/etc/nod32/nod32.cfg`) to the original MDA component. The original MDA component can be at this stage of the procedure still found in the back-up file (`/etc/postfix/main.cf.orig`) you have created. Thus assuming your original MDA component is (`/usr/bin/maildrop`) the appropriate parameter of the main NOD32LS configuration file will look like this:

```
mda_path = "/usr/bin/maildrop"
```

To accomplish the whole procedure, one has to restart both the MTA Postfix and the daemon `nod32d`.

5.2.2.3. Setting Qmail MTA

The delivery options in the Qmail program are configured via the command line parameters at the time of program execution or by using short scripts where the appropriate command line statements are stored. In the following we assume that the Qmail is installed in a `/var/qmail` directory and there is a Qmail's starting script called usually `'rc'` within this directory that is always executed at the start-up of Qmail by running the standard statement

```
qmailctl start
```

In a simplest case the script (`/var/qmail/rc`) is of the following content:

```
#!/bin/sh
exec env - PATH="/var/qmail/bin:$PATH" qmail-start ./Maildir/ splogger qmail
```

Note: The first `qmail-start` argument tells him where to deliver messages to. This time it is a special directory structure locating in `./Maildir` within the local user home directory. There are, indeed, more possibilities to be written at the place of the first command line argument (for instance `./Mailbox` to deliver the locally delivered messages to the MAILBOX file located at the local user home directory; `|preline procmail` to use the procmail MDA as a local deliver agent, etc. See the `dot-qmail(5)` manual page).

In order to involve the module `nod32mda` into the message delivery process the user is required to prepend `!/usr/bin/nod32mda` to the first command line argument of `qmail-start`, i.e. the starting script (`var/qmail/rc`) will look as follows:

```
#!/bin/sh
exec env - PATH="/var/qmail/bin:$PATH" \
qmail-start '!/usr/bin/nod32mda' ./Maildir/ splogger qmail
```

Note: There is a space before the second `'` and no space after it. This configuration is independent of the first `qmail-start` argument, just be sure the original argument is still at the end of the new.

With the above modifications, we have provided that all messages originally processed by the MTA for delivery will now be primarily sent to module `nod32mda`. Still there is the second part of the modification to provide that all messages processed by `nod32mda` will be passed back to Qmail's delivery program, that will take care about the message successful delivery. For this we have to modify the parameter `mda_path` located in the section `[mda]` of the `nod32` configuration file to use a custom script

```
mda_path = "/var/qmail/bin/qmail-nod32mda"
```

that you should create with this contents and run `chmod a+x` on it:

```
#!/bin/sh
exec qmail-local -- "$USER" "$HOME" "$LOCAL" "" "$EXT" "$HOST" "$SENDER" "$1"
```

In this configuration, in case of infection or temporary error, `nod32mda` will automatically (based on present environment variables) use Qmail's recognized exit codes (100 or 111, see the manual page of `qmail-command(8)`). If you want to pass some command line arguments to `nod32mda` itself, you have to specify them in the first `qmail-start` argument by appending the text `'--user "$USER"'` to it. Our example `rc` file will then look like this:

```
#!/bin/sh
exec env - PATH="/var/qmail/bin:$PATH" \
qmail-start '!/usr/bin/nod32mda' ./Maildir/ -- --user "$USER" splogger qmail
```

5.2.2.4. Setting MTA Exim version 3

An MTA Exim version 3 stores its configuration in the file `/etc/exim/exim.conf` (resp. `/etc/exim.conf`). The procedure described here will therefore be done on ly with this one `_le` and of course with the main `NOD32LS` configuration file (`/etc/nod32/nod32.cfg`). First one has to grab information on MDA used by Exim. This can be found in the main Exim configuration file (`/etc/exim/exim.conf`) in the section usually called `TRANSPORTS CONFIGURATION`. However, first before any further steps, please make a back-up of the original main Exim configuration file (for instance with the name `/etc/exim/exim.conf.orig`). Now let's look inside the content of the appropriate section responsible for local delivery. This can, for instance, look like the following, where we assume the `procmail` be the original MDA component:

```
# TRANSPORTS CONFIGURATION
procmail_pipe:
  driver = pipe
```

```

command = /usr/bin/procmail -d $local_part
return_path_add
delivery_date_add
envelope_to_add
check_string = "From"
escape_string = ">From"
user = $local_part
group = mail
# DIRECTORS CONFIGURATION
procmail:
  driver = localuser
  transport = procmail_pipe

```

As can be seen in this screenshot, the original MDA component is referenced in the configuration file by parameter 'command'.

Note: Please note that besides the section related to transport there should always be defined the section usually called 'DIRECTORS CONFIGURATIONS' defined in the configuration file where the appropriate parameter transport is referenced. In the next step we will use the 'which' command

```
which procmail
```

to get the full path of this MDA component. Now, in order to involve the module nod32mda into the message processing, we have to replace the MDA procmail by the module nod32mda. In order to do so, we are required to replace the appropriate line in the main Exim configuration file so it will look as follows:

```
command = /usr/bin/nod32mda -d $local_part
```

Note: Please keep in mind that the switches used in 'command' after the MDA component has stayed the same as in the original configuration file, otherwise the current configuration may not work properly. Note that in case you would like to pass command line parameter to the nod32mda agent it is necessary to define parameter command in a different way. Agent nod32mda will accept command line parameters only in case the sentence '--' is added before them. So for instance to pass command line option --user the nod32mda agent the parameter 'command' has to be defined as follows.

```
command = /usr/bin/nod32mda -d $local_part -- --user $local_part
```

In this case agent 'nod32mda' will instruct daemon 'nod32d' to search special section [user] within the configuration file.

With the above modifications we have ensured that all the messages originally sent by the MTA for delivery to the MAILBOX (by using an original MDA) will now be primarily sent to module nod32mda. Still there remains the second part of the modification to provide that all messages processed by nod32mda will be sent to the original MDA. In order to do so, we have to modify the parameter 'mda_path' within the section [mda] of the main NOD32LS configuration file (/etc/nod32/nod32.cfg) to the original MDA component. The original MDA component can be, at this stage of the procedure, still found in the back-up file (/etc/exim/exim.conf) you have created. Thus assuming your original MDA component is (/usr/bin/procmail) the appropriate parameter of the main NOD32LS configuration file will look like this:

```
mda_path = "/usr/bin/procmail"
```

To accomplish the whole procedure, one has to restart both the MTA Exim and the daemon nod32d.

5.2.2.5. Setting MTA Exim version 3 (more general)

An MTA Exim version 3 stores its configuration in the file /etc/exim/exim.conf (resp. /etc/exim.conf). The procedure described here will therefore be done only with this one file and of course with the main NOD32LS configuration file (/etc/nod32/nod32.cfg).

Please, note that the configuration described here is MDA independent by means it can be used in cooperation with any MDA already working with Postfix MTA. First before any further steps, please make a back-up of the original main Exim configuration file (for instance with the name `/etc/exim/exim.conf.orig`). Now let's look inside the content of the exim configuration file. It is typically compound from the `TRANSPORTERS` and `DIRECTORS` sections. In order to configure exim to use NOD32LS one has to define special `DIRECTOR` section

```
# DIRECTORS CONFIGURATION
nod32_director:
  driver = smartuser
  condition = "${if eq {$received_protocol}{virus-scanned} {0}{1}}"
  transport = nod32_transport
  verify = false
```

The above section has to be placed as a first entry of the `DIRECTORS CONFIGURATIONS` section.

One has also define appropriate `TRANSPORTER` section responsible to deliver e-mail messages to nod32mda agent.

```
# TRANSPORTS CONFIGURATION
nod32_transport:
  driver = pipe
  command = /usr/bin/nod32mda -oMr virus-scanned $local_
part@$domain
  user = mail
  group = mail
```

Be sure that the 'user' (usually 'mail') used in the above setting is listed in a 'trusted_users' list for this parameter. Also be sure that the option 'qualify_domain' is undefined or set to your fully qualified domain name. Note that in case user would like to pass command line parameter to the nod32mda agent it is necessary to define parameter command in a different way. Agent nod32mda will accept command line parameters only in case the sentence '--' is added before them. So for instance to pass command line option --user the nod32mda agent the parameter 'command' has to be defined as follows.

```
command = /usr/bin/nod32mda -oMr virus-scanned $local_part@$domain -- --user $local_part
```

In this case agent 'nod32mda' will instruct daemon 'nod32d' to search special section [user] within the configuration file. With the above setting we have ensured that all the e-mail messages sent to user belonging to local domain will now be primarily sent to module nod32mda. Still there remains the second part of the modification to provide that all messages processed by nod32mda will be sent to the appropriate MAILBOX. In order to do so, we have to modify the parameter 'mda_path' within the section [mda] of the main NOD32LS configuration file (`/etc/nod32/nod32.cfg`) in a following way.

```
mda_path = "/usr/sbin/exim"
```

Above we have assumed that the exim binary file is located within a directory `/usr/sbin`. If this is not the case, please modify the above parameter as necessary. To accomplish the whole procedure, one has to restart both the MTA Exim and the daemon nod32d.

5.2.2.6. Setting MTA Exim version 4

An MTA Exim version 4 stores its configuration in the file `/etc/exim4/exim4.conf`. The procedure described here will therefore be done only with this one file and of course with the main NOD32LS configuration file (`/etc/nod32/nod32.cfg`). Please, note that the configuration described here is MDA independent by means it can be used in cooperation with any MDA already working with Postfix MTA. First before any further steps, please make a back-up of the original main Exim configuration file (for instance with the name `/etc/exim4/exim4.conf.orig`). Now let's

look inside the content of the `exim` configuration file. It is typically compound from the `TRANSPORTERS` and `ROUTERS` sections. Usually there should be always `ROUTER` section responsible for messages local delivery (usually it is called 'localuser'). In order to configure `exim` to use `NOD32LS` one has to define special `ROUTER` section

```
# ROUTER CONFIGURATION
nod32_router:
    driver = accept
    domains = +local_domains
    condition = "${if eq {$received_protocol}{virus-scanned} {0}{1}}"
```

and place this section as a first entry of the `ROUTERS CONFIGURATION` section.

One has also define appropriate `TRANSPORT` section responsible to deliver e-mail messages to `nod32mda` agent.

```
# TRANSPORTS CONFIGURATION
nod32_transport:
    driver = pipe
    command = /usr/bin/nod32mda -oMr virus-scanned $local_part@$domain
    user = mailnull
    group = mail
```

Be sure that the 'user' (usually 'mailnull') is the value of 'exim_user' or pick a name from the list 'trusted_users' for this parameter.

Also be sure that the option 'qualify_domain' is undefined or set to your fully qualified domain name.

Note that in case user would like to pass command line parameter to the `nod32mda` agent it is necessary to define parameter command in a different way. Agent `nod32mda` will accept command line parameters only in case the sentence '--' is added before them. So for instance to pass command line option --user the `nod32mda` agent the parameter 'command' has to be defined as follows.

```
command = /usr/bin/nod32mda -oMr virus-scanned $local_part@$domain -- --user $local_part
```

In this case agent 'nod32mda' will instruct daemon 'nod32d' to search special section [user] within the configuration file.

With the above setting we have ensured that all the e-mail messages sent to user belonging to local domain will now be primarily sent to module `nod32mda`. Still there remains the second part of the modification to provide that all messages processed by `nod32mda` will be sent to the appropriate MAILBOX. In order to do so, we have to modify the parameter 'mda_path' within the section [mda] of the main `NOD32LS` configuration file (`/etc/nod32/nod32.cfg`) in a following way.

```
mda_path = "/usr/sbin/exim"
```

Above we have assumed that the `exim` binary file is located within a directory `/usr/sbin`. If this is not the case, please modify the above parameter as necessary. To accomplish the whole procedure, one has to restart both the MTA `Exim` and the daemon `nod32d`.

5.3. Scanning the outbound e-mail messages

Scanning of the outbound e-mail messages is performed at the time of message transmission between the local MUA and the MTA (as marked by symbol S2 in figure 3-1). A more detailed scheme of the process is shown in the figure 5-2.

The most important part of scanning the outbound messages is done by the `nod32smtp` filter. This filter is a resident program (daemon) that performs in general three functions:

- * receives data via the INET socket,
- * extracts e-mail/s and feeds nod32d (scanning daemon) to scan it,
- * forwards the e-mail to another port or computer.

The nod32smtp uses the SMTP protocol for its communication. As discussed in the section 4, the configuration of nod32smtp daemon is specified in section [smtp] of the main NOD32LS configuration file /etc/nod32/nod32.cfg. The following parameters are defined:

listen_addr – specify the smtp server address where the daemon listens to

listen_port – specify the smtp server port where the daemon listens to

server_addr – specify the smtp server address where the communication will be redirected to

server_port – specify the smtp server port where the communication will be redirected to

The operation principle of scanning an outbound e-mail message is based on the following idea: We configure a nod32smtp daemon to listen to communication incoming to port 2525 of the email server computer and forward the scanned communication to port 25 of the same computer where, typically, the MTA daemon listens to. In order to do so, use the following values of the aforementioned parameters:

```
listen_addr = "localhost"
listen_port = 2525
server_addr = "localhost"
server_port = 25
```

Once again, an e-mail received via port 2525 will be processed by nod32d (scanning daemon) and afterward sent to port 25 for further processing by agent MTA of the e-mail messaging system. So far this is just half of the job. The second part that has to be done is the automatic redirection of all the packets arriving on port 25 of the server computer to port 2525. If we do not this then no packet of e-mail communication will go through the nod32smtp daemon as the whole communication from the LAN (Local Area Network) will pass by through port 25.

Thus, in order to do proper rerouting one has to write the following rules for ipchains (resp. iptables):

Kernel 2.2.X:

```
ipchains -I INPUT -p tcp -s 192.168.1.0/24 -d 0.0.0.0/0 25 -j REDIRECT 2525
```

Kernel 2.4.X:

```
iptables -I PREROUTING -t nat -p tcp -s 192.168.1.0/24 --dport 25 -j REDIRECT --to-ports 2525
```

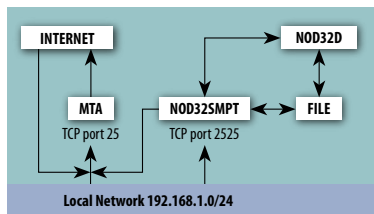
where ipchains (iptables) for kernel 2.2.X (2.4.X) have been used. After applying these rules and starting the nod32smtp daemon

```
/sbin/service nod32smtp start
```

One can see that all the communication arrives to the nod32smtp daemon and everything works correctly, however, with this setting there is still one unwanted effect. One can easily realize that the port 2525 with this setting provides an open relay. The point is that now there is a nod32smtp daemon that will accept all the packets that arrive at the port 2525, this also means packets arriving from outside the local network. The daemon nod32smtp will forward this traffic to port 25. This process will be interpreted by MTA as a local communication on the so called loop-back interface and therefore will not be rejected by MTA rules.

This problem, however, can be solved by ensuring that all communication with the port 2525 will be disabled with the exception of the local network. In order to do so we use the following commands:

Figure 5-3. The scheme of the scanning of outbound e-mail messages by using nod32smtp module.



Kernel 2.2.X:

```
ipchains -I INPUT -p tcp -s ! 192.168.1.0/24 -d 0.0.0.0/0 2525 -j REJECT
```

Kernel 2.4.X:

```
iptables -I INPUT -p tcp -s ! 192.168.1.0/24 --dport 2525 -j DROP
```

5.4. Content filtering in MTA

Content filtering method is currently a well known method used to screen and/or exclude certain defined information from the Internet or its part. Concerning an e-mail server system the best place to implement content filtering method is the MTA agent as an e-mail communication traffic nod. The advantage of such an implementation is that it allows one to scan e-mails inbound as well as outbound in the same implementation algorithm. On the other hand the content filtering method is MTA dependent. Taking into account the number of different MTAs working in the present UNIX OS environment, the method is not universal. NOD32LS comes with three content filters built for most common MTA, i.e. MTA Sendmail, Postfix, Exim and Qmail, described in the following sections.

5.4.1. Content filtering in MTA Postfix

The `nod32smtp` filter can also serve as a content filter for MTA Postfix providing the following changes are implemented:

In section `[smtp]` of the configuration file: `/etc/nod32/nod32.cfg` set:

```
listen_port = 2526
server_addr = "localhost"
server_port = 2525
```

With these settings `nod32smtp` will listen on port 2526 and will forward all communication from this port to the local port 2525.

In the next step, modify the `/etc/postfix/master.cf` file by adding the following specification into the file:

```
localhost:2525 inet n - n - - smtpd
-o content_filter=
-o myhostname=nod32.yourdomain.com
```

The value of the parameter `myhostname` must differ from that set in postfix (`postconf myhostname`), otherwise a loop is detected (by postfix) and the e-mail is rejected.

Finally, add

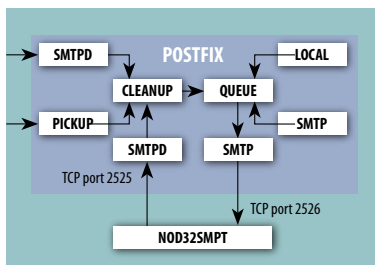
```
postconf -e "content_filter = smtp:localhost:2526"
```

into the postfix configuration file (`/etc/postfix/main.cf`). The entire process is illustrated in figure 5-3.

5.4.2. Content filtering in MTA Sendmail

The `nod32smfi` serves as a content filter for MTA Sendmail. It is a third-party program that accesses all the mail messages being processed by MTA Sendmail and therefore can be used in the bidirectional scanning scenario. In order to enable filtering, please provide the following line in the section `[smfi]` of the configuration file: `/etc/nod32/nod32.cfg` set:

Figure 5-4. Bidirectional scanning scheme of an `nod32smtp` module working as a content filter.



```
smfi_sock_path = "/var/run/nod32smfi.sock"
```

With these settings nod32smfi will communicate with the MTA Sendmail via unix socket /var/run/nod32smfi.sock. In the next step, modify the /etc/mail/sendmail.cf file by adding the following specification into the section MAIL FILTER DEFINITIONS:

```
Xnod32smfi, S=local:/var/run/nod32smfi.sock, F=T, T=S:2m;R:2m;E:5m
```

With this setting sendmail will communicate with the nod32smfi daemon via local (i.e. unix) socket /var/run/nod32smfi.sock. Flag F=T will result in temporary fail connection if the filter is unavailable. Flag T=S:2m defines timeout 2 minutes for sending information from MTA to a filter. Flag T=R:2m defines timeout 2 minutes for reading reply from the filter. Flag T=E:5m means overall timeout 5 minutes between sending end-of-message to filter and waiting for the final acknowledgment.

Note: In case the timeouts for the nod32smfi filter are set too small, Sendmail can temporarily reject the message which will attempt to pass through at a later time. This will lead to the continuous rejection of one and the same message later. In order to avoid the problem, the timeouts have to be set properly. In order to do this, one has to get into account 'confMAX_MESSAGE_SIZE' parameter defined in a sendmail.mc file that will provide not accepting messages bigger than the appropriate parameter value (given in bytes). Taking into account this value and the maximum time for processing of this amount of data by MTA (this can be measured) one can evaluate the appropriate timeouts for nod32smfi filter. Finally, uncomment and modify the following line in the /etc/mail/sendmail.cf file:

```
O InputMailFilters=nod32smfi
```

Since nod32smfi filter can modify the content of the e-mail message body, in case of multiple Sendmail filters, it is good to put the definition of the nod32smfi filter at the end of the filter chain.

5.4.3. Content filtering in MTA Exim

Configuration of the NOD32LS as an Exim content filter uses similar approach as discussed in the sections devoted to scanning inbound messages in Exim (general case). Indeed, in order to configure NOD32LS as content filter it is first necessary to follow the rules from section 5.2.2.5 (in case of Exim 3) or follow the rules from section 5.2.2.6 (in case of Exim 4). With this initial setting the NOD32LS is configured to scan every e-mail coming to the local recipient. Yet it is necessary to provide the scanning of the e-mail messages routed to another domains. In case of Exim 3 it is only necessary to define new ROUTER CONFIGURATIONS section

```
nod32_router:
  driver = domainlist
  route_list = "** localhost byname"
  condition = "${if eq {$received_protocol}{virus-scanned} {0}{1}}"
  transport = nod32_transport
  verify = false
```

This section has to be placed as first section among all ROUTER CONFIGURATIONS sections. In case of Exim 4 it is only necessary to comment out the line

```
domains = +local_domains
```

from already existing 'nod32_router' ROUTERS CONFIGURATIONS section defined by rules in section 5.2.2.6. of this manual. To accomplish the whole procedure, one has to restart both the MTA Exim and the daemon nod32d.

5.4.4. Content filtering in MTA Qmail

Nod32pipe can serve as a content-filter for Qmail. However, you need to download and compile the qmail-qfilter program, version 2.0 or newer. Following instructions don't require you to use the QMAILQUEUE patch. We assume your qmail is installed in /var/qmail. First create the script /var/qmail/bin/qmail-nod32pipe with these contents and run `chmod a+x` on it:

```
#!/bin/sh
export QMF_QMAILQUEUE=/var/qmail/bin/qmail-queue.nod32save
exec qmail-qfilter /usr/bin/nod32pipe
```

Next, rename the file /var/qmail/bin/qmail-queue to /var/qmail/bin/qmailqueue.nod32save and create a soft link /var/qmail/bin/qmail-queue to the new script /var/qmail/bin/qmail-nod32pipe. In this configuration, in case of infection or temporary error, nod32pipe will automatically (based on present environment variables) use Qmail's recognized exit codes (31, 71 or 99, see the manual page of qmail-queue(8) and qmail-qfilter(1)). It will also ask nod32d to use the user specific configuration based on the first envelope recipient, like is done in nod32smf and nod32smtp content filters. To accomplish the whole procedure, one has to restart the MTA Qmail.

5.5. Alternative methods of scanning e-mails

Although mechanisms described in previous sections are concerned to be the basic mechanisms of the e-mail messages scanning, there exists yet other possibilities that are all described in this section.

5.5.1. Scanning e-mail messages using AMaViS

AMaViS – A Mail Virus Scanner is a tool that interfaces your MTA and several anti-virus scanners. It supports Sendmail, qmail, Postfix, Exim and comes in three branches:

amavis – for low/medium mail volume

amavisd – for higher mail volume, daemonized version of amavis

amavisd-new – for higher mail volume, Anti-Spam, ISP features, ...

Amavis cooperates with NOD32LS by using its command line interface nod32cli (see the nod32cli manual page for details). Yet before we go into detailed explanation of the Amavis – NOD32LS configurations, we would like to discuss the impact of the method on the NOD32LS software functionality. First, one should note that Amavis does not allow modification of the body of scanned e-mail messages directly by NOD32LS software. Particularly, no infected e-mail message processed and delivered to the final recipient will be cleaned directly by NOD32LS software. Another consequence is that no NOD32 footnote will be written into the e-mail body. Another feature of the described method is that the modification of e-mail header is indirect from the point of view of NOD32LS software. Particularly, status dependent, header modification directly by NOD32LS is disabled. Taking into account the above statements we recommend the use of Amavis – NOD32LS configuration (described in the next sections) only in case the above discussed features of NOD32LS are not requested by the user. However, it is good to keep in mind that using Amavis, because of its anti-spam features, does not conflict with content filtering methods of NOD32LS (described in section 5.4).

5.5.1.1. amavis

Configuration of Amavis with NOD32LS is performed during the process of Amavis installation. For installation, first unpack the source `amavis-0.x.y.tgz` and overwrite the file `amavis/av/nod32cli` with this contents:

```
#
# ESET Software NOD32 Command Line Interface, Version 2.51
#

if ($nod32cli) {
    do_log(2,"Using $nod32cli");
    chop($output = '$nod32cli --subdir $TEMPDIR/parts');
    $errval = retcode($?);
    do_log(2,$output);
    if ($errval == 0) { # no errors, no viruses found
        $scanner_errors = 0;
    } elseif ($errval == 1 || $errval == 2) {
        # no errors, viruses discovered
        $scanner_errors = 0;
        @virusname = ($output =~ /virus="([\^"]+)"/g);
        do_virus();
    } else {
        do_log(0,"Virus scanner failure: $nod32cli (error code:
        $errval)");
    }
}
}
```

Note that the above modification provides accepting of the message only in case it was originally clean. The rest of non-error states returned by NOD32LS will be treated in a way that the message will be dropped. The messages resolved by NOD32LS as cleaned/deleted will be dropped as well as nod32cli module has no exclusive access to the message and therefore is not able to guarantee its cleaning/deletion. Above modification also treats the "error in archive" status (3) of nod32cli in a way that the message is rejected. Particularly, messages with the password protected attachment are treated in this way as NOD32LS cannot mark the messages with the "not scanned" status. In order to change these default settings user is free to modify the above text, however, it is not recommended unless he is sure about the consequences. Please, read the discussion at the end of foreword to this section to get more information on NOD32LS functionality when configured with Amavis. For successful installation you may need arc, unarj, unrar, zoo, make a symlink in /usr/bin from uncompress to gzip and create the user amavis in group amavis with home dir /var/amavis. Rename (at least for the amavis configuration) the file /usr/sbin/nod32 to /usr/sbin/nod32.bak. Now continue with the usual installation process (./configure, make, make install) and follow the rules README.mta according your mail server.

5.5.1.2. amavisd

Configuration of Amavisd with NOD32LS is performed during the process of Amavisd installation. Unpack the source amavisd-0.x.tgz and follow the rules for amavis described in previous section of this guide.

Note: After 'make install' you may need to move /usr/etc/amavisd.conf to /etc and do a 'make install' again. Don't forget to run amavisd as user amavis after finishing the installation.

5.5.1.3. amavisd-new

In order to install NOD32LS with Amavisd-new, unpack and install the source amavisd-new-2.x.y.tgz in your installation directory. Now to configure NOD32LS with newly installed Amavisd-new, delete the clause for 'ESET

Software NOD32' and replace the clause for 'ESET Software NOD32 – Client/Server Version' in file 'amavisd.conf' with the following one:

```
# http://www.nod32.com/  
['ESET Software NOD32 Command Line Interface v 2.51',  
 '/usr/bin/nod32cli', '--subdir {}',  
 [0], [1,2], qr/virus=" ([^"]+)"/ ],
```

Please, note the NOD32 scanning status values written within square brackets of the above setting. They are set to follow the same performance of Amavis – NOD32LS cooperation as defined by default in the section discussing Amavis configuration. User is free to modify the above text, however, it is not recommended unless he is sure about the consequences. Please, read end of the foreword to this section and end of the section discussing Amavis configuration in order to get more information. You may need to install additional perl modules Archive-Tar, Archive-Zip, BerkeleyDB, Compress-Zlib, Convert-TNEF, Convert-UUlib, IO-stringy, MailTools, MIME-Base64, MIME-tools, Net-Server and Unix-Syslog from www.cpan.org/modules. The procedure is by each as follows: perl Makefile.PL; make; make install. After configuring NOD32LS follow the recommendation for configuring Amavisd-new in README.mta located in Amavisd-new directory according your mail server.

Chapter 6:

NOD32LS as for Linux File Servers

This chapter describes process of configuration of NOD32LS system in order to provide an efficient protection from virus and worms infection of the file systems by using on-demand and on-access scanning techniques. The NOD32 for Linux File Servers part of the system is composed from the so called on-demand scanner 'nod32' and from two so-called on-access scanners 'libnod32pac.so', 'nod32dac'. All concerned components are described in the following sections.

6.1. On-demand scanner

On-demand scanner is scanner that can be invoked by privileged user (usually system administrator) using command line interface or by operating system using periodic command scheduler. This is also an explanation of the term 'on-demand' that the file system objects are scanned on user and/or system demand. Concerning NOD32 on-demand scanner there are not special requirements for its operation. After proper installation of the NOD32LS package and after valid license has been provided within '/etc/nod32/license' directory feel free to run on-demand scanner by using command line interface or scheduler tool. In order to run on-demand scanner from command line the following syntax is expected

```
/usr/sbin/nod32 [option(s)] INCL_SCAN_DIR -- -EXCL_SCAN_DIR
```

where INCL_SCAN_DIR (resp. EXCL_SCAN_DIR) is list of directories and/or files to be scanned (resp. excluded from scanning). The format of both arguments is colon separated list

```
"dir1:dir2:dir3:..."
```

where dir1, dir2, dir3, ... are absolute and/or relative paths to the directories and/or files included into (resp. excluded from) the list. Multiple command line options are implemented within NOD32 on-demand scanner. To get full list of them, please, read appropriate manual page for NOD32 on-demand scanner nod32(8). Tip: As indicated already it is good idea to use on-demand scanner on the regular basis to protect operating system. Next example shows how to set periodic command scheduler 'cron' for running on-demand scanner each day at 3AM. In order to do so one has to enter the following line into the 'cron' table (use statement 'crontab -e' for this purpose):

```
0 3 * * * /usr/sbin/nod32 [option(s)] INCL_SCAN_DIR -- -EXCL_SCAN_DIR
```

Please note that there is no configuration file interface supported for this module.

6.2. On-access scanner

On-access scanner is scanner invoked by predefined triggered access of user(s) and/or operating system to the file system objects. This also explains the term 'on-access' that the scanner is started on attempt to access selected file system object. NOD32LS contains two modules implementing on-access scanning technique. First is the onaccess technique based on interception of the relevant kernel calls (nod32dac). Second technique uses preload library to intercept relevant calls of the LIBC library (libnod32pac.so). Each of the techniques has its advantages and disadvantages. The technique based on kernel calls interception is powered by Dazuko (read da-tzu-ko) kernel module. Dazuko project is a Free software, by means that it is distributed as a free source code, in order to allow users compilation of the kernel module for their own custom kernels. Note that the Dazuko kernel module is not a part of the NOD32LS package and thus it must be compiled and installed into the kernel prior the NOD32 on-access scanner (nod32dac daemon) initialization. On the other hand the Dazuko technique make on-access scanning independent of used file system type. It is also suitable for controlling file system objects via Network File System (NFS), Nettalk and Samba. The additional instalation of the Dazuko module can be non-wished for system administrators which carry on the critical systems where source code and/or configuration file appropriate to the actually running kernel is not available or the kernel is rather monolithic than modular. In this case second discussed on-access scanning technique based on the preload LIBC library comes handy. Before we provide user with the detailed information related with the on-access scanner configuration and operation,

let us make few remarks concerned with the common functionality and use of the on-access scanner.

IMPORTANT: First, we would like to point out that both the on-access scanners are not assumed to provide protection of whole file system where installed. They have been developed and tested to protect primarily the file systems mounted externally. If this is not the case, one should count on exclusion of multiple directories from file access control to prevent system from hang-up. Typical directory to be excluded in this case is '/dev' directory or variables containing directories used by NOD32LS, however other directories could be problematic as well. **IMPORTANT:** Second, it is generally bad idea to try run both the described on-access scanners simultaneously, as the system may hang-up.

6.2.1. On-access scanner powered by Dazuko

This section contains information concerned with operation, installation and configuration of on-access scanner using Dazuko kernel module.

6.2.1.1. Operation principle

On-access scanner 'nod32dac' (NOD32 Dazuko powered file Access Controller) is a resident program (daemon) providing permanent monitoring and control over the file system. It runs as a daemon and can be executed at the system startup.

Scanning of each file system object is performed upon customizable file access event of the user and/or operating system. The following file access types are supported by the current version:

ON_OPEN events – This file access type is controlled once first bit of the integer parameter 'event_mask' in the main NOD32 configuration file (section [dac]) is 1. In this case ON_OPEN bit of Dazuko access mask is set on.

ON_CLOSE events – This file access type is controlled once second bit of the integer parameter 'event_mask' in the main NOD32 configuration file (section [dac]) is 1. In this case ON_CLOSE bit and ON_CLOSE_MODIFIED bit of Dazuko access mask is set on.

Note that some of the kernel versions does not support interception of the ON_CLOSE events. In this case problems could be detected when running nod32dac module.

ON_EXEC events – This file access type is controlled once third bit of the integer parameter 'event_mask' in the main NOD32 configuration file (section [dac]) is 1. In this case ON_EXEC bit of Dazuko access mask is set on.

By using this mechanism all opened, closed and executed regular files are scanned by daemon nod32d for viruses. Based on the result of this scanning the access to the files as denied or allowed.

6.2.1.2. Installation and configuration

On-access scanner 'nod32dac' (NOD32 Dazuko powered file Access Controller) is a resident program (daemon) providing permanent monitoring and control over the file system. It runs as a daemon and can be executed at the system startup. Scanning of each file system object is performed upon customizable file access event of the user or operating system. It has been already discussed that prior any 'nod32dac' initialization so-called Dazuko kernel module has to be compiled and installed within the running kernel. Note that the following text contains only brief description concerned with the Dazuko kernel module installation. Therefore it is highly recommended to read the Dazuko how-to-install documentation (see <http://www.dazuko.org/howto-install.shtml>) in order to compile and load the Dazuko module properly into the kernel. Download Dazuko tarball from (<http://www.dazuko.org/downloads.shtml>). Note that the NOD32 on-access scanner is compatible with the Dazuko kernel module whose version is equal or higher than 2.1.0. Therefore the proper behavior of NOD32 on-access scanner running with the older versions of Dazuko kernel module is not guaranteed. Before compilation of the Dazuko it is necessary to have a source code of actually running kernel stored in the kernel source code directory

```
/usr/src/linux*
```

Note that it is important to have a version of the kernel source code exactly matching the version of the kernel that is currently running on the operating system. Even if the kernel source code is present in the kernel source code directory, be certain that it is properly configured. Read the Dazuko FAQ related to this topic (<http://www.dazuko.org/faq.shtml>).

Note: When installing dazuko module under 2.6.x Linux kernel, the following must be taken into account:

- * When running multiple security modules, make sure the „Enable different security modules” is enabled and the dazuko module is the first security module loaded into kernel. To get more information what has to be done during kernel configuration, please, read the Dazuko FAQ related to this topic (<http://www.dazuko.org/faq.shtml>).
- * The ‘commoncap’ kernel module has to be loaded into kernel prior ‘dazuko’ module.

Generate the Makefile by running:

```
./configure
```

within the Dazuko source code directory. Compile Dazuko code with:

```
make
```

which will create the ‘dazuko.o’ (‘dazuko.ko’ for 2.6.x Linux kernels) file. If any warnings or errors appeared as a consequence of the above steps, the Dazuko module was not compiled properly. In this case it is not recommended to go further over the next steps of this section but rather refer to the appropriate Dazuko documentation (<http://www.dazuko.org/faq.shtml>).

Insert the dazuko module into kernel by typing

```
/sbin/insmod dazuko.o
```

and check that the module is properly loaded. The check can be done for instance by using:

```
/sbin/lsmmod
```

or:

```
cat /proc/modules
```

In both cases there should be dazuko listed in the output of these commands. Type:

```
cat /proc/devices
```

to check whether the dazuko device has been started successfully. The Dazuko with its device major number should be listed again. Unless the device node is not created automatically, create it with:

```
mknod -m 600 /dev/dazuko c $(grep dazuko /proc/devices | sed "s/ .*//" ) 0 chown root:root /dev/dazuko
```

Read and edit [global] and [dac] sections of NOD32 main configuration file ‘/etc/nod32/nod32.cfg’. **IMPORTANT:** It is important to note that in order to run nod32dac simultaneously with nod32d scanning daemon it is necessary to register nod32d daemon with Dazuko kernel module. In order to do so one has to define configuration option

```
dazuko_support_enabled = yes
```

within a section [global] of the main NOD32 configuration file. Note also that for the proper run of on-access scanner it is necessary to define file system objects (i.e. directories and files) that are required to be under control of dazuko kernel module. This can be achieved via ‘ctl_incl’ and ‘ctl_excl’ configuration options defined within [dac] section of the configuration file. **IMPORTANT:** Please, don’t remove ‘/dev’ directory from the control of on-access scanner (e.g. ‘ctl_excl = “/dev”’) unless you know exactly what you are doing, otherwise system can hang up. Run nod32dac daemon by typing:

```
/etc/init.d/nod32dac start
```

6.2.1.3. Tips To provide

Dazuko module loading prior each nod32dac daemon initialization, follow the next steps: Provide copy of Dazuko module in some of the directories located within the directory reserved for the kernel modules

```
/lib/modules
```

Use the kernel utilities `depmod` and `modprobe` to handle the dependencies setting and proper loading of the newly added Dazuko module. Insert the following line into initialization script `/etc/init.d/nod32dac` before the `nod32dac` daemon initialization:

```
/sbin/modprobe dazuko
```

IMPORTANT: It is highly important to execute the individual steps above exactly in the order as they are written above. The reason is that in case of kernel module not located within the kernel modules directory, `modprobe` will not be able to handle properly loading of the module and the system can hang-up.

6.2.2. On-access scanner using preload libc library

This section contains information concerned with operation, installation and configuration of on-access scanner using preload library `'libnod32pac.so'`.

6.2.2.1. Operation principle

On-access scanner `'libnod32pac.so'` (NOD32 Preload library based file Access Controller) is a shared objects library that is used as a preload library of LIBC and can become functional during the system start-up. It is thus applicable for file system servers using LIBC calls, for instance ftp server, Samba server etc. Scanning of each file system object is performed upon customizable file access event of the user and/or operating system. The following file access types are supported by the current version:

ON_OPEN events – This file access type is controlled once first bit of the integer parameter `'event_mask'` in the main NOD32 configuration file (section `[pac]`) is 1. In this case all `'open'` or `'open64'` calls of the LIBC are intercepted.

ON_CLOSE events – This file access type is controlled once second bit of the integer parameter `'event_mask'` in the main NOD32 configuration file (section `[pac]`) is 1. In this case all `'close'`, `'dup'` and `'dup2'` calls of the LIBC are intercepted.

By using this mechanism all opened and closed descriptors tied to regular files are scanned by daemon `nod32d` for viruses. Based on the result of this scanning the access to the files is denied or allowed.

6.2.2.2. Installation and configuration

The `'libnod32pac.so'` installation is done using standard installation mechanism of the preload libraries. One has just to define the environment variable `'LD_PRELOAD'` with absolute path pointing to the `'libnod32pac.so'` library. Please refer also to the manual page `ld.so(8)` to get further information. **IMPORTANT:** It is important to note that the `'LD_PRELOAD'` environment variable has to be defined just for the network server daemon process (ftp, samba, etc.) we would like to have under control. Generally it is not recommended to preload LIBC calls in all operating system processes as for controlling the selected file system area it is not necessary and it can dramatically slow down the performance of the system or even cause the system hang-up. In this sense all mechanisms using configuration file `'/etc/ld.so.preload'` are not correct as well as mechanisms using statement `'export LD_PRELOAD'`. Both would override all relevant LIBC calls in the whole system that will lead to the system hang-up during its initialization. Thus in order to intercept just relevant file access calls related with just objects within selected file system area, one has to override an executable statement of an appropriate network file system server with the following line

```
LD_PRELOAD=/usr/lib/libnod32pac.so COMMAND 39
```

where `'COMMAND'` is the original executable statement. Note also that for the proper run of on-access scanner it is necessary to define file system objects (i.e. directories and files) that are required to be under control of the preload library. This can be achieved via `'ctl_incl'` and `'ctl_excl'` configuration options defined within `[pac]` section of the configuration file.

6.2.2.3. Tips

In order to provide on-access scanner functionality immediately after network file system server start-up, it is good to define environment variable 'LD_PRELOAD' directly within an appropriate network file server initialization script. EXAMPLE: Let's assume we would like to have on-access scanner catching all file system access events immediately after starting the samba server. Thus within the initialization script concerned with samba daemon (/etc/init.d/smb), we replace the statement

```
daemon /usr/sbin/smbd $SMBDOPTIONS
```

responsible for initialization of smbd daemon by the following line

```
LD_PRELOAD=/usr/lib/libnod32pac.so daemon /usr/sbin/smbd $SMBDOPTIONS
```

In this manner selected file system objects controlled by Samba will be checked immediately after Samba initialization, i.e. during the system start-up.

Chapter 7:

Sample submission system

Sample submission system is functionality that provides catching of the infected objects found by advanced heuristics method and delivering these objects to the sample submission system server. All virus samples caught by the sample submission system will be processed by the team of NOD32 virus laboratory department and consequently added into the NOD32 virus database, if necessary.

Note: According to our License Agreement, by enabling sample submission system You are agreeing to allow the computer and/or platform on which the nod32d is installed to collect data (which may include personal information about You and/or the user of the computer) and samples of newly detected viruses or other threats and send them to our virus lab. This feature is turned off by default. We will only use this information and data to study the threat and will take reasonable steps to preserve the confidentiality of such information.

In order to turn on this feature, set both parameters `samples_enabled` and `samples_send_enabled` in `/etc/nod32/nod32.cfg` to yes. Sample submission system is able to send infected samples also via http proxy server with basic authentication. See the nod32d manual page for details.

Chapter 8:

NOD32 system update and maintenance

In order to keep NOD32LS system effective, it is necessary to keep NOD32 virus definitions database up to date. In the following sections a concept of the NOD32 database update process is described with more in depth details which is in most cases not necessary to read. Therefore one may wish to skip directly to section 6.3 that can serve in this sense as a short 'how-to' document related to this topic.

8.1. Basic concept of NOD32 system update

Basic concept of the NOD32 system update is to download so called NOD32 modules from the NOD32 server and compile them in order to provide so called loading modules used by the NOD32 anti-virus scanner. First part of the process is provided by NOD32 Update Mirror Creator (`nod32umc`) system utility while compilation of the modules is done by NOD32 Update (`nod32upd`) system utility. Both the utilities are part of the NOD32LS system.

8.1.1. NOD32 modules organization

The NOD32 modules introduced above are located by default within

```
/var/lib/nod32/mirror
```

directory (so called local mirror directory or default local mirror directory). These modules are divided into two categories, e.g. engine category and component category (modules of component category are currently only for use on the Windows operating systems). Currently there are 4 types of engine category modules supported:

- * base scanning modules (prefix `engine`) containing virus signatures database,
- * archives support modules (prefix `archs`) supporting various file system archive formats,
- * advanced heuristics modules (prefix `advheur`) containing implementation of so called advanced heuristics method of virus and worm detection,
- * packed worm scanner modules (prefix `pwsan`). These modules are always necessary for proper running of any NOD32 anti-virus scanner based application and therefore are all downloaded by default at each download process.

On the other hand the component category modules are platform dependent and language localization dependent and thus the selection of them has to be specified by the user according to his needs. Download of component category modules is therefore optional.

Besides the NOD32 modules, the default mirror directory contains so called modules version control file (`update.ver`). This file contains relevant information concerning all modules stored in the local mirror directory and is used as a reference file for generation of further subordinate mirrors. The process of subordinate mirror creation is described in the section 6.2 in detail.

8.1.2. NOD32 mirror creation

As has already been told, NOD32 update mirror creator (`nod32umc`) utility is used to download and maintain the mirror of NOD32 modules storage. Minimal `nod32umc` configuration requires that you define authentication options and the absolute path to the directory where the mirror will be created. Authentication options are used to pass to `nod32umc` the user name and password (received from vendor) necessary to authenticate the end-user against NOD32 server where the modules are downloaded from. Several scenarios for user authentication against NOD32 server are supported:

- * direct user name and password setting into the command line.
- * setting via an authentication file,
- * setting via an old featured '.netrc' file.

Please refer also to the nod32umc manual pages for a detailed description of the related command line options. After proper authentication the modules will be downloaded into a defined mirror directory. There exists also the possibility to download and update NOD32 modules in the situation where all the communication of the local computer with the NOD32 server is mediated via http proxy server. Read more on this topic in nod32umc manual pages. In situations when user authentication is required against the appropriate proxy server, the minimal nod32umc configuration must include also the proxy authentication options. There are two supported scenarios for user authentication against proxy server:

- * direct proxy user name and proxy password setting into the command line
- * setting via a proxy authentication file.

Please, refer also to the nod32umc manual pages for a detailed description of the related command line options.

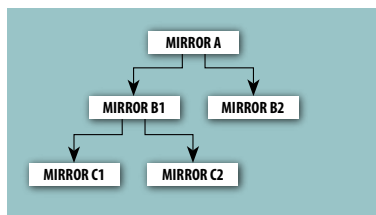
Note: For security reasons, please use authentication scenarios via authentication files rather than scenarios via typing the user name(s) and password(s) directly into the command line.

8.1.3. Generation of NOD32 scanner loading modules

In order to create loading modules from those downloaded by nod32umc one must use the NOD32 Update (nod32upd) system utility (see also nod32upd manual pages). Minimal configuration for nod32upd is to define full mirror directory path. The following NOD32 loading modules will be created: base module (nod32.000), archives support module (nod32.002), advanced heuristics module (nod32.003) and packed worm scanner module (nod32.004). One could specify the directory path where the loading module would be created (so called NOD32 base directory, i.e. directory where the NOD32 anti-virus scanner will load the modules from). One can choose an arbitrary directory for this purpose, however, this must contain nod32.000 module. There is no requirement given

on the version of the module. In case the module is not located in the directory chosen, error 105 will occur. It is recommended to set NOD32 base directory as `'/var/lib/nod32'` as this is exactly the default directory where the modules are loaded by NOD32LS anti-virus scanner from. In case the NOD32 base directory is not chosen by the user it is defined by default as the current working directory. In case the `'update.ver'` file located in the mirror directory is not signed by valid digital signature, error 107 will appear. One can force nod32upd to not check the digital signature of the file by choosing switch `'--no_signature'`. Please refer to the manual pages nod32upd to get detailed information on the command line options.

Figure 8-1. The scheme of subordinate NOD32 mirrors creation. With the presented hierarchy it is not necessary to perform update process of the mirror C1 by tasking the mirror A which preserves the appropriate server from the overload.



8.2. Subordinate mirrors creation

After download of NOD32 modules the nod32umc utility will create in the mirror directory the `'update.ver'` file containing the information about the modules currently stored in the newly created mirror. The newly created mirror is a fully functional mirror of NOD32 modules storage and can be used in the process of additional mirrors creation. This hierarchy allows the user to update all the modules in all the subsequently created mirrors without using, and thus overloading, one and the same mirror server (see figure 6-1 for details).

At this point we would like you to notice that the creation of the mirror file structure at some computers file system, is not enough to provide the fully featured NOD32LS mirror. For proper function of the newly created mirror there are

additional conditions to be fulfilled. First, as the utility `nod32umc` uses `http` protocol to download the NOD32 modules there must be a `http` server installed on the computer where the modules are going to be downloaded from. Second, the NOD32 modules to be downloaded by other computers have to be placed at the directory path

```
/http-serv-base-path/nod_upd
```

where `http-serv-base-path` is a base `http` server directory path, as this is the first place where `nod32umc` utility looks the NOD32 modules for.

8.3. Automatic update of the virus definitions database

The NOD32LS comes with the script

```
/usr/sbin/nod32_update
```

used to provide user with a comfortable and reliable service concerned with the process of NOD32 virus signatures database update.

8.3.1. Structure and use of automatic update script

The script is predefined to download NOD32 modules from the server

```
http://www.nod32.com
```

and to create the loading modules used by NOD32 scanner into

```
/var/lib/nod32
```

directory. This configuration can be, however, easily changed by simple editing the script in its configuration part. Even if the script does all the job in most cases without manual intervention into it, one has still to define the user name and password for NOD32 server authentication prior to using it. By default the update script reads this setting from the so called 'authorization file'

```
/etc/nod32/nod32.auth
```

that has to be adjusted by the user. The authorization file is an ASCII file with the following format.

```
username=your_nod32_username
```

```
password=your_nod32_password
```

Please, use your favorite editor to invoke the file and fill its appropriate parts by valid username and password that you have received from your vendor. From the security reason make sure that the authorization file is publicly unreadable. If it is not the case change its permission appropriately by typing

```
chmod 600 /etc/nod32/nod32.auth
```

into the command line.

8.3.2. Periodic update of the virus definitions database

To provide the highest security for the user, the NOD32 team collects the virus definitions continuously from all over the world. The new patterns can appear within the database in very short intervals. It is therefore useful, and also recommended, to trigger an update attempt (by means of executing the `nod32_update` script) on a regular basis via periodic scheduler. One hour interval is recommended. In order to do so, configure periodic scheduler (`cron`) on your system by entering the following line

```
0 * * * * /usr/sbin/nod32_update
```

into its configuration file (crontab). To add the above line into the crontab use command line statement

Chapter 9:

Tips and tricks

This chapter is devoted to describe tips and tricks concerned with configuration of NOD32LS. This means it describes configuration of NOD32LS in circumstances when for instance MTA is configured to use other software with similar functionality or with functionality that could normally lead to misconfiguration of NOD32LS.

9.1. Dropping messages marked by NOD32 as deleted in MTA Postfix

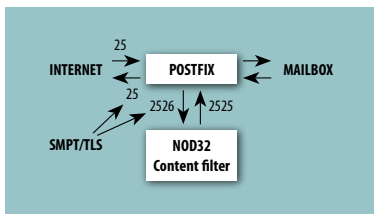
In an Internet has recently appeared nonnegligible increase of the number of the e-mail messages containing worms. In most cases the infected attachment of such messages cannot be cleaned but rather deleted and whole messages even does not contain any reasonable information. In such a cases it has sense to discard (or treat in special way) this kind of messages. Mechanism described in this section can be used to suppress messages marked by NOD32LS as deleted in MTA Postfix. First of all one has to add the following entry

```
write_to_header = 1
```

into section [smtp] of the main NOD32LS configuration file (/etc/nod32/nod32.cfg). This setting will result in a modification of each non-clean e-mail message by means the string 'XNOD32Result: status' is inserted into header of the message. Word 'status' of the string is replaced by actual status of the scanning process. In order to discard all messages that has been marked as 'deleted' one has to add following line

```
header_checks = regexp:/etc/postfix/header_checks
```

Figure 9-1. Scheme of content filtering in Postfix MTA with enabled TLS.



into the main Postfix configuration file (/etc/postfix/main.cf). At the same time one has to create file '/etc/postfix/header_checks' with the following content.

```
/^X-NOD32Result: deleted/ DISCARD
```

To accomplish the whole procedure, one has to restart MTA Postfix, daemon nod32d and daemon nod32smtp. Note that in older Postfix versions DISCARD functionality may not work. In this case warning message 'Postfix does not know the command DISCARD' will appear in the '/var/log/maillog' file. This can be only solved by update of the Postfix package.

9.2. NOD32LS and TLS support in MTA

Transport Layer Security (TLS) is a protocol guaranting data privacy in client/server communication over the Internet. The basic principle of TLS is based on the SSL encryption of data traveling between client and server (We have on our mind the SMTP communication between MTA client and server). This has of course nonnegligible consequences for scanning of this kind of communication by NOD32LS. For instance, once TLS support in MTA is enabled, the 'outbound messages scanning scenario' discussed in section 5.3 is impossible as the whole intercepted SMTP communication is encrypted at this stage. On the other hand, there is possibility to use data encryption in communication between local MTA and Internet and still use the NOD32LS as a content filter (discussed in section 5.4). In MTA Sendmail content filtering there is no problem with SMTP TLS support at all as the Sendmail Milter does not relay on the SMTP communication and content filtering is done rather internally. On the other hand the Postfix uses SMTP protocol for data communication between content filter and MTA. Therefore once the TLS is enabled in Postfix, the content filtering method fails as whole the SMTP communication is encrypted. Fortunately, this can be solved on the Postfix TLS configuration level. The situation is depicted in a figure 7-1.

As is shown in the figure above, once the TLS is enabled, all the SMTP communication channels including SMTP communication with content filter are affected. The only possibility in this case is to disable the TLS support for communication between client and server located within localhost. This can be achieved by adding the following line into the main Postfix configuration file.

```
smtp_tls_per_site = hash:/etc/postfix/smtp_tls_per_site
```

In addition it is necessary to create the above file with the following content

```
localhost NONE
```

and provide its appropriate hash table. In order to do so, execute the following statement from '/etc/postfix' directory.

```
postmap hash:smtp_tls_per_site
```

By using the above statement the '/etc/postfix/smtp_tls_per_site.db' file is created that is used by Postfix to enable TLS on per site basis. As far as we have disabled TLS for localhost the content filtering can be used and at the same time the SMTP communication between local MTA and Internet is encrypted.

Chapter 10:

Let us know

Dear user, this guide should have given you a good knowledge about how the NOD32LS works and how to configure it in order to protect your e-mail messaging system with highest efficiency. However, writing a documentation is a process that is never finished. There will always be some parts of the NOD32LS that can be explained better or are not even explained at all. Therefore, in case of bugs or inconsistencies found within this documentation, please report a problem to our support center

<http://www.nod32.com/support/support.htm>

or write an e-mail directly to linux development mailing list

<linux@nod32.com>.

Appendix A.

Installed content of NOD32LS package

```
/etc/init.d/nod32d
/etc/init.d/nod32dac
/etc/init.d/nod32smfi
/etc/init.d/nod32smtp
/etc/nod32/nod32.auth
/etc/nod32/nod32.cfg
/etc/nod32/nod32d_script
/etc/nod32/nod32d_license_warning_script
/etc/nod32/sig_footer_clean.html.example
/etc/nod32/sig_footer_infected.html.example
/etc/nod32/sig_footer_not_scanned.html.example
/etc/nod32/sig_header_clean.html.example
/etc/nod32/sig_header_infected.html.example
/etc/nod32/sig_header_not_scanned.html.example
/usr/bin/nod32mda
/usr/bin/nod32smfi
/usr/bin/nod32smtp
/usr/bin/nod32cli
/usr/sbin/nod32_update
/usr/sbin/nod32d
/usr/sbin/nod32umc
/usr/sbin/nod32upd
/usr/share/doc/nod32ls/copyright
/usr/share/doc/nod32ls/guide.us.txt.gz
/usr/share/man/man1/nod32mda.1.gz
/usr/share/man/man1/nod32smfi.1.gz
/usr/share/man/man1/nod32smtp.1.gz
/usr/share/man/man1/nod32cli.1.gz
/usr/share/man/man5/nod32.5.gz
/usr/share/man/man5/nod32.cfg.5.gz
/usr/share/man/man8/nod32d.8.gz
/usr/share/man/man8/nod32umc.8.gz
/usr/share/man/man8/nod32upd.8.gz
/var/lib/nod32/mirror
/var/cache/nod32
```