Network monitoring with Nagios and OpenBSD

Author: Daniele Mazzocchio
Last update: May 24, 2007
Latest version: http://www.kernel-panic.it/openbsd/nagios/

Table of Contents

1. Introduction

So our OpenBSD-based network now includes redundant firewalls, domain name servers, a mail gateway and a web proxy cache. All the services provided by these machines are particularly critical and can't afford even minimal downtime. Redundancy may give us the time to recover a failure before having angry users trying to knock down our door, but it doesn't free us from the responsibility to detect and solve ongoing problems.

In sum, it's time to think about monitoring our network! And the following are the perfect ingredients for implementing a full-featured, secure and reliable network monitoring system:

OpenBSD
: the operating system for the security paranoid, with "*only two remote holes in the default install, in more than 10 years!*";

Nagios
: the most popular "*open source host, service and network monitoring program*";

Apache
: the "*secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards*".

My pick goes to Nagios for its ease of use, flexibility and extensibility. It also features a very clean and straightforward design, as it is structured into three basic building blocks:

- a daemon process, running periodic checks on specific hosts and services and managing notifications when problems arise;
- an optional web interface, to access current status information, historical logs and reports via a simple web browser;
- a set of external plugins, i.e. the (possibly custom) scripts executed by the daemon process to actually perform the checks and send out notifications.

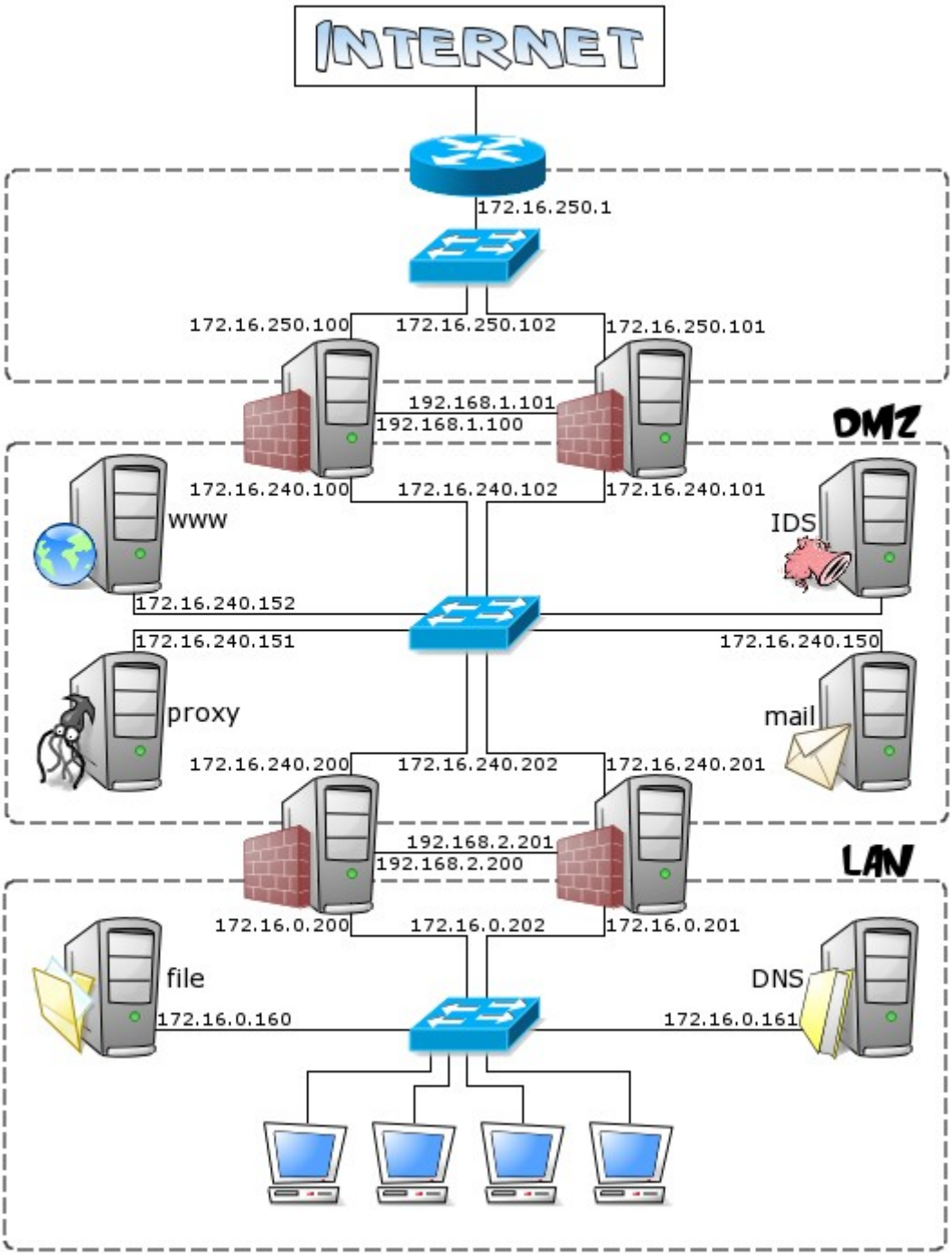
Furthermore, these basic components can be easily extended with external modules, thus making it easy for Nagios to meet even your most demanding needs! Therefore, after the installation and configuration of the Nagios' core components, we will take a brief look at some of its most popular and useful addons:

- NRPE, the Nagios Remote Plugin Executor, which allows you to execute local plugins on remote hosts;
- NSCA, the Nagios Service Check Acceptor, which sends passive service checks from a host to the Nagios server;
- NagVis, the Nagios Visualization Addon, which allows you to deeply customize how Nagios data is displayed;

A good knowledge of OpenBSD is assumed, since we won't delve into system management topics such as base configuration or packages/ports installation.

2. Installation and base configuration

Before delving straight into the details of Nagios installation and configuration, let's take a brief look at the layout of the network that we're going to monitor.



It's a very simple and small network, made up of:

- a LAN (172.16.0.0/24), containing clients and servers not accessible from the public internet (e.g. file server, DHCP server);
- a DMZ (172.16.240.0/24), containing the servers that must access the internet (e.g. mail, web and proxy servers);
- a router, in a small subnet (172.16.250.0/24), connecting the DMZ to the internet.

Our network monitoring system is a security-critical host and won't need to directly access the internet, so it

will perfectly fit in the internal LAN.

The OpenBSD installation procedure is documented in full detail in the official FAQ, so we won't linger on it here. Nagios doesn't have particular requirements and a standard OpenBSD installation will do just fine: according to the documentation, Nagios makes do with just a machine running Linux (or UNIX variant). That doesn't sound so fussy, does it?

2.1 Packages installation

Nagios installation only requires adding a few packages:

- libltdl-*x.x*.tgz
- libiconv-*x.x.x*.tgz
- expat-*x.x.x*.tgz
- gettext-*x.x.x*.tgz
- nagios-plugins-*x.x*.tgz
- nagios-*x.x*-chroot.tgz
- nagios-web-*x.x*-chroot.tgz

The installation procedure will automatically create the user and group that the monitoring daemon will drop its privileges to (`_nagios`). The `chroot` flavor will install Nagios in a way suited for chrooted `httpd(8)`, i.e. with the CGIs statically linked and all the configuration and log files stored inside the `/var/www` directory. By the way, Nagios has a particular directory structure that you will have to become familiar with:

`/var/www/nagios/`
: this directory contains the static HTML pages for the web interface and the online documentation;

`/var/www/cgi-bin/nagios/`
: contains the dynamic CGI pages of the web interface, which actually retrieve and display the current status of the monitored objects;

`/var/www/etc/nagios/`
: you should put all your Nagios configuration files in this directory: we will examine them one by one in a moment;

`/var/www/var/log/nagios/`
: this is the directory where Nagios will create the log, status and retention files;

`/var/www/var/log/nagios/archives/`
: Nagios log files are periodically rotated and moved to this directory;

`/var/www/var/nagios/rw/`
: contains the external command file;

`/usr/local/libexec/nagios/`
: contains the standard plugins.

2.2 Configuration overview

Nagios configuration may look overly complicated at first glance; even the documentation warns that Nagios is quite powerful and flexible, but unfortunately it's not very friendly to newbies. Anyway, don't despair! Once you've figured out the underlying logic of its "object-oriented" configuration, you will appreciate Nagios' flexibility and clean design. For the first tests, you can start by tweaking the sample configuration files contained in the `/usr/local/share/examples/nagios/` directory, customizing them to your needs.

The syntax of Nagios configuration files follows a few basic rules:

- comments start with a "#" character and span to the end of the line;
- variable names must begin at the start of the line (i.e. no indentation allowed);
- variable names are case sensitive;

- no spaces are allowed around the "=" sign.

Configuration involves setting several parameters concerning the monitoring daemon, the CGIs and, of course, the hosts and services you want to monitor. All this information is spread among multiple files: we will now examine them in turn.

2.2.1 The main configuration file

The overall behaviour of the Nagios daemon is determined by the directives included in the main configuration file, `/var/www/etc/nagios/nagios.cfg`. Though this file contains several dozens of parameters, for most of them the default value is the most reasonable option and you will probably want to care about only very few of them (usually `cfg_file`, `cfg_dir` and `admin_email`). In any case, you can find a detailed description of each and every parameter in the official documentation.

`/var/www/etc/nagios/nagios.cfg`

```
# Path to main log file and log archive directory. All pathnames are relative
# to the chroot directory '/var/www/'
log_file=/var/log/nagios/nagios.log
log_archive_path=/var/log/nagios/archives

# Paths to files managed internally by the application
object_cache_file=/var/nagios/objects.cache
status_file=/var/nagios/status.dat
comment_file=/var/nagios/comments.dat
downtime_file=/var/nagios/downtime.dat
state_retention_file=/var/nagios/retention.dat
temp_file=/var/nagios/nagios.tmp
command_file=/var/nagios/rw/nagios.cmd
lock_file=/var/run/nagios/nagios.pid

# Object definitions (see next chapter) can be split across multiple files.
# You may either list files individually (using the 'cfg_file' parameter) or
# group them into directories (using the 'cfg_dir' parameter). In the latter
# case, Nagios will process all files with a '.cfg' extension found in the
# specified directories and their subdirectories
cfg_file=/etc/nagios/timeperiods.cfg
cfg_file=/etc/nagios/contacts.cfg
cfg_file=/etc/nagios/commands.cfg
cfg_file=/etc/nagios/generic-hosts.cfg
cfg_file=/etc/nagios/generic-services.cfg
cfg_dir=/etc/nagios/hosts
cfg_dir=/etc/nagios/services

# Path to the resource file, containing user-defined macros (see below). You can
# specify more than one resource file using multiple 'resource_file' statements
resource_file=/etc/nagios/resource.cfg

# User and group the Nagios process will run as
nagios_user=_nagios
nagios_group=_nagios

# Email address and pager number for the administrator of the local machine
admin_email=nagios@kernel-panic.it
admin_pager=xxx-xxx-xxxx

# Date format (available options: us, euro, iso8601 or strict-iso8601)
date_format=euro

# Enable checks, notifications and event handlers. Passive checks allow external
# applications to submit check results to Nagios. Event handlers are optional
# commands that are executed whenever a host or service state change occurs
```

```
execute_service_checks=1
accept_passive_service_checks=1
execute_host_checks=1
accept_passive_host_checks=1
enable_notifications=1
enable_event_handlers=1

# Checks freshness options. Enabling these options will ensure that passive
# checks are always up-to-date
check_service_freshness=1
service_freshness_check_interval=60
check_host_freshness=0
host_freshness_check_interval=60

# External commands allow the web interface and external applications (such as
# NSCA) to issue commands to Nagios. With a check interval of '-1', Nagios will
# check for external commands as often as possible
check_external_commands=1
command_check_interval=-1

# Various logging options
log_rotation_method=d
use_syslog=1
log_notifications=1
log_service_retries=1
log_host_retries=1
log_event_handlers=1
log_initial_states=0
log_external_commands=1
log_passive_checks=1

# Enable retention of state information between program restarts (refer to
# documentation for details)
retain_state_information=1
retention_update_interval=60
use_retained_program_state=1
use_retained_scheduling_info=0

# State flapping detection options (refer to documentation for details)
enable_flap_detection=0
low_service_flap_threshold=5.0
high_service_flap_threshold=20.0
low_host_flap_threshold=5.0
high_host_flap_threshold=20.0

# Miscellaneous tuning, performance and security options (refer to
# documentation for details)
interval_length=60
service_inter_check_delay_method=s
max_service_check_spread=30
service_interleave_factor=s
host_inter_check_delay_method=s
max_host_check_spread=30
max_concurrent_checks=0
service_reaper_frequency=10
auto_reschedule_checks=0
auto_rescheduling_interval=30
auto_rescheduling_window=180
sleep_time=0.25
service_check_timeout=60
host_check_timeout=30
event_handler_timeout=30
notification_timeout=30
```

```
ocsp_timeout=5
perfdata_timeout=5
use_aggressive_host_checking=0
process_performance_data=0
obsess_over_services=0
check_for_orphaned_services=0
aggregate_status_updates=1
status_update_interval=15
event_broker_options=-1
p1_file=/usr/local/bin/p1.pl
illegal_object_name_chars=`~!$%^&*|'"<>?,()=
illegal_macro_output_chars=`~$&|'"<>
use_regexp_matching=0
use_true_regexp_matching=0
daemon_dumps_core=0
```

2.2.2 The resource file

The resource file allows you to assign values to the user-definable macros `$USERn$` (where *n* is a number between 1 and 32 inclusive). Basically, in Nagios, macros are variables (starting and ending with a dollar sign, "`$`") that you can insert into command definitions and that will get expanded to the appropriate value immediately prior to the execution of the command. User-defined macros (and the several other macros Nagios makes available) allow you to keep command definitions generic and simple (see the next chapter for some examples).

User-defined macros are normally used to store recurring items in command definitions (like directory paths) and sensitive information (like usernames and passwords). It is recommended that you set restrictive permissions (`600`) on the resource file(s) in order to keep sensitive information protected.

```
/var/www/etc/nagios/resource.cfg
```

```
# Set $USER1$ to be the path to the plugins
$USER1$=/usr/local/libexec/nagios

# MySQL username and password
$USER2$=root
$USER3$=password
```

The next step is configuring object data, which is probably the trickiest part of the configuration. We will therefore devote the next chapter entirely to this topic.

3. Object data configuration

So now it's time to tell Nagios what to keep tabs on. Therefore, we must supply it with information about:

- *when* and *how* to perform checks and send out notifications;
- *whom* to notify;
- *which* hosts and services to monitor.

All this information is represented by means of objects, which are defined by a set of "`define`" statements, enclosed in curly braces and containing a variable number of newline-separated directives, in keyword/value form. Keywords are separated from values by whitespace and multiple values can be separated by commas; indentation within statements is allowed.

To summarize, the basic syntax of an object declaration can be represented as follows:

```
define object {
    keyword-1     value-1
    keyword-2     value-2,value-3,...
    [...]
    keyword-n     value-n
}
```

Object definitions can be split into any number of files: just remember to list them all in the main configuration file by using the `cfg_file` and/or `cfg_dir` directives.

3.1 Timeperiod definition

The `timeperiod` statement allows you to specify, for each day of the week, one or more time slots in which to run certain checks and/or notify certain people. Time intervals can't span across midnight and excluded days are simply omitted.

In the following example, all the `timeperiod` definitions are grouped together in a file named `timeperiods.cfg` stored in the `/var/www/etc/nagios/` directory.

`/var/www/etc/nagios/timeperiods.cfg`

```
# The following timeperiod definition includes normal work hours. The
# 'timeperiod_name' and 'alias' directives are mandatory. Note that weekend days
# are simply omitted
define timeperiod {
    timeperiod_name  workhours
    alias            Work Hours
    monday           09:00-18:00
    tuesday          09:00-18:00
    wednesday        09:00-18:00
    thursday         09:00-18:00
    friday           09:00-18:00
}

# The following timeperiod includes all time outside normal work hours. The
# time slot between 6 p.m. and 9 a.m. must be split into two intervals, to avoid
# crossing midnight
define timeperiod {
    timeperiod_name  nonworkhours
    alias            Non-Work Hours
    sunday           00:00-24:00
    monday           00:00-09:00,18:00-24:00
    tuesday          00:00-09:00,18:00-24:00
    wednesday        00:00-09:00,18:00-24:00
    thursday         00:00-09:00,18:00-24:00
    friday           00:00-09:00,18:00-24:00
```

```
    saturday         00:00-24:00
}

# Most checks will probably run on a continuous basis
define timeperiod {
    timeperiod_name always
    alias           Every Hour Every Day
    sunday          00:00-24:00
    monday          00:00-24:00
    tuesday         00:00-24:00
    wednesday       00:00-24:00
    thursday        00:00-24:00
    friday          00:00-24:00
    saturday        00:00-24:00
}

# The right timeperiod when you don't want to bother with notifications (e.g.
# on vacation or during testing)
define timeperiod {
    timeperiod_name never
    alias           No Time is a Good Time
}
```

3.2 Command definition

The next step is to tell Nagios *how* to perform the various checks and send out notifications; this is accomplished by defining multiple `command` objects specifying the actual commands for Nagios to run.

Command definitions are pairs of short names and command lines (both mandatory) and can contain macros. As we mentioned before, macros are variables, enclosed in "`$`" signs, that will get expanded to the appropriate value immediately prior to the execution of a command; macros allow you to keep command definitions generic and straightforward. A simple example will make this clear.

Suppose you want to monitor a web server with IP address "1.2.3.4"; you could then define a command such as the following:

```
define command {
    command_name    check-http
    command_line    /usr/local/libexec/nagios/check_http -I 1.2.3.4
}
```

This definition is correct and will certainly do the job. But what if you later decide to add a new web server? Would you find it convenient to define a new (almost identical) command, with only the IP address changed? It is way more efficient to take advantage of macros by writing a single generic command such as:

```
define command {
    command_name    check-http
    command_line    $USER1$/check_http -I $HOSTADDRESS$
}
```

and leave Nagios the responsibility to expand the `$HOSTADDRESS$` macro to the appropriate IP address, obtained from the host definition (see below). As you'll remember from the previous chapter, the `$USER1$` macro holds the path to the plugins directory.

Now let's complicate things a bit! What if you want Nagios to check the availability of a particular URL on each web server? This URL may differ from server to server, so what we need now is a command definition that is still generic and yet server-specific! Though this may sound contradictory, once again Nagios solves this problem with macros: in fact, the `$ARGn$` macros (where *n* is a number between 1 and 32 inclusive) act as placeholders for service-specific arguments that will be specified later within service definitions (see

below for further details). Therefore, the above command definition would turn into:

```
define command {
    command_name    check-http
    command_line    $USER1$/check_http -I $HOSTADDRESS$ -u $ARG1$
}
```

In addition to the ones we have just seen, Nagios provides several other useful macros. Please refer to the documentation for a detailed list of all available macros and their validity context. Below is a sample set of command definitions.

/var/www/etc/nagios/commands.cfg

```
################################################################################
# Notification commands                                                        #
# There are no standard notification plugins; hence notification commands are  #
# usually custom scripts or mere command lines.                                #
################################################################################
define command {
    command_name    host-notify-by-email
    command_line    $USER1$/host_notify_by_email.sh $CONTACTEMAIL$
}

define command {
    command_name    notify-by-email
    command_line    $USER1$/notify_by_email.sh $CONTACTEMAIL$
}

define command {
    command_name    host-notify-by-SMS
    command_line    /usr/local/bin/sendsms $ADDRESS1$ "Nagios: Host $HOSTNAME$
($HOSTADDRESS$)is in state: $HOSTSTATE$"
}

define command {
    command_name    notify-by-SMS
    command_line    /usr/local/bin/sendsms $ADDRESS1$ "Nagios: Service $SERVICEDESC$
on $HOSTALIAS$ is in state: $SERVICESTATE$"
}

################################################################################
# Check commands                                                               #
# The official Nagios plugins should handle most of your needs for host and    #
# service checks. Anyway, should they not, we will discuss in a moment how to  #
# write custom plugins.                                                        #
################################################################################
define command {
    command_name    check-host-alive
    command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100%
-p 1
}

define command {
    command_name    check-ssh
    command_line    $USER1$/check_ssh $HOSTADDRESS$
}

define command {
    command_name    check-http
    command_line    $USER1$/check_http -I $HOSTADDRESS$ -u $ARG1$
}

define command {
```

```
    command_name    check-smtp
    command_line    $USER1$/check_smtp -H $HOSTADDRESS$
}

define command {
    command_name    check-imap
    command_line    $USER1$/check_imap -H $HOSTADDRESS$
}

define command {
    command_name    check-dns
    command_line    $USER1$/check_dns -s $HOSTADDRESS$ -H $ARG1$ -a $ARG2$
}

define command {
    command_name    check-mysql
    command_line    $USER1$/check_mysql -H $HOSTADDRESS -u $USER2$ -p $USER3$
}

[...]
```

3.3 Contact definition

`contact` objects allow you to specify people who should be notified automatically when the alert conditions are met. Contacts are first defined individually and then grouped together in `contactgroup` objects, for easier management.

For the first time, in the following definitions, we will refer to previously defined objects. In fact, the values of the `host_notification_period` and `service_notification_period` directives must be `timeperiod` objects; and the values of the `host_notification_command` and `service_notification_command` directives must be `command` objects.

/var/www/etc/nagios/contacts.cfg

```
define contact {
# Short name to identify the contact
    contact_name                    john
# Longer name or description
    alias                           John Doe

# Timeperiods during which the contact can be notified about host and service
# problems or recoveries
    host_notification_period        always
    service_notification_period     always

# Host states for which notifications can be sent out to this contact
# (d=down, u=unreachable, r=recovery, f=flapping, n=none)
    host_notification_options       d,u,r

# Service states for which notifications can be sent out to this contact
# (w=warning, c=critical, u=unknown, r=recovery, f=flapping, n=none)
    service_notification_options    w,u,c,r

# Command(s) used to notify the contact about host and service problems
# or recoveries
    host_notification_commands      host-notify-by-email,host-notify-by-SMS
    service_notification_commands   notify-by-email,notify-by-SMS

# Email address for the contact
    email                           jdoe@kernel-panic.it
```

```
# Nagios provides 6 address directives (named address1 through address6) to
# specify additional "addresses" for the contact (e.g. a mobile phone number
# for SMS notifications)
    address1                        xxx-xxx-xxxx
}

# The following contact is split in two, to allow for different notification
# options depending on the timeperiod
define contact {
    contact_name                    danix@work
    alias                           Daniele Mazzocchio
    host_notification_period        workhours
    service_notification_period     workhours
    host_notification_options       d,u,r
    service_notification_options    w,u,c,r
    host_notification_commands      host-notify-by-email
    service_notification_commands   notify-by-email
    email                           danix@kernel-panic.it
}

define contact {
    contact_name                    danix@home
    alias                           Daniele Mazzocchio
    host_notification_period        nonworkhours
    service_notification_period     nonworkhours
    host_notification_options       d,u
    service_notification_options    c
    host_notification_commands      host-notify-by-email,host-notify-by-SMS
    service_notification_commands   notify-by-email,notify-by-SMS
    email                           danix@kernel-panic.it
    address1                        xxx-xxx-xxxx
}

[...]

# All administrator contacts are grouped together in the "Admins"
# contactgroup
define contactgroup {
    contactgroup_name               Admins
    alias                           Nagios Administrators
    members                         danix@work,danix@home,john
}

[...]
```

3.4 Host definition

Now we have finally come to one of the most important facets of Nagios configuration: the definition of the hosts (servers, workstations, devices, etc.) that we want to monitor. This will lead us to introduce one of the most powerful features of Nagios configuration: object inheritance. Note that, though we are discussing it now first, object inheritance applies to all Nagios objects; however, it's in hosts and services definition that you can get the most out of it.

In fact, configuring a host requires setting up quite a few parameters; and the value of these parameters will normally be the same for most hosts. Without object inheritance, this would mean wasting a lot of time typing the same parameters over and over again and eventually ending up with cluttered, overweight and almost unmanageable configuration files.

But luckily, Nagios is smart enough to save you a lot of typing by allowing you to define special template objects, whose properties can be "inherited" by other objects without having to rewrite them. Below is a brief example of how a template is created:

```
define host {
    name                            generic-host-template  # Template name

    check_command                   check-host-alive
    check_period                    always
    max_check_attempts              5
    notification_options            d,u,r

    register                        0                      # Don't register it!
}
```

As you can see, a template definition looks almost identical to a normal object definition. The only differences are:

- every template must be assigned a name with the `name` directive;
- since this is not an actual host, you must tell Nagios not to register it by setting the value of the `register` directive to 0; this property doesn't get inherited and defaults to 1, so you won't need to explicitely override it in all "children" objects;
- a template object can be left incomplete, i.e. it may not supply all mandatory parameters.

To create an actual host object from a template, you simply have to specify the template name as the value of the `use` directive and make sure that all mandatory fields are either inherited or explicitely set:

```
define host {
    host_name                       hostname
    use                             generic-host-template
    alias                           alias
    address                         x.x.x.x
}
```

Well, now let's move from theory to practice and define two host templates for our servers. Note that the second one inherits from the first; this is possible because Nagios allows multiple levels of template objects.

/var/www/etc/nagios/generic-hosts.cfg

```
# The following is a template for all hosts in the LAN
define host {
# Template name
    name                            generic-lan-host

# Command to use to check the state of the host
    check_command                   check-host-alive

# Contact groups to notify about problems (or recoveries) with this host
    contact_groups                  Admins

# Enable active checks
    active_checks_enabled           1
# Time period during which active checks of this host can be made
    check_period                    always
# Number of times that Nagios will repeat a check returning a non-OK state
    max_check_attempts              3

# Enable the event handler
    event_handler_enabled           1

# Enable the processing of performance data
    process_perf_data               1

# Enable retention of host status information across program restarts
    retain_status_information       1
# Enable retention of host non-status information across program restarts
```

```
    retain_nonstatus_information    1

# Enable notifications
    notifications_enabled           1
# Time interval (in minutes) between consecutive notifications about the
# server being _still_ down or unreachable
    notification_interval           120
# Time period during which notifications about this host can be sent out
    notification_period             always
# Host states for which notifications should be sent out (d=down,
# u=unreachable, r=recovery, f=flapping, n=none)
    notification_options            d,u,r

# Don't register this definition: it's only a template, not an actual host
    register                        0
}

# DMZ hosts inherit all attributes from the generic-lan-host by means of the
# 'use' directive. The only difference is that Nagios has to go through the
# internal (CARP) firewalls to reach the DMZ servers, thus requiring the
# additional 'parents' directive.
define host {
    name                            generic-dmz-host

# The 'use' directive specifies the name of a template object that you want
# this host to inherit properties from
    use                             generic-lan-host

# This directive specifies the hosts that lie between the monitoring host
# and the remote host (more information here)
    parents                         fw-int

# This too is a template
    register                        0
}
```

Now we can take advantage of our templates to define the actual hosts in a few lines. The `hostextinfo` directive allows you to specify some additional information about each host; in any case, this infomation has no effect on monitoring and is completely optional.

/var/www/etc/nagios/hosts/servers.cfg

```
# Configuration for host dns1.lan.kernel-panic.it
define host {
    use                             generic-lan-host
    host_name                       dns1
    alias                           LAN primary master name server
    address                         172.16.0.161
}

define hostextinfo {
    host_name                       dns1
    notes                           This is the internal primary master name server
(Bind 9.3.4)
# URL with more information about this host
    notes_url                       http://www.kernel-panic.it/openbsd/dns/
# Image associated with this host in the status CGI; images must be placed in
# /var/www/nagios/images/logos/
    icon_image                      dns.png
# String used in the 'alt' tag of the icon_image
    icon_image_alt                  [dns]
# Image associated with this host in the statusmap CGI
    statusmap_image                 dns.gd2
```

```
}

# Configuration for host mail.kernel-panic.it
define host {
    use                             generic-dmz-host
    host_name                       mail
    alias                           Mail server
    address                         172.16.240.150
}

define hostextinfo {
    host_name                       mail
    notes                           This is the Postfix mail server (with IMAP(S)
and web access)
    notes_url                       http://www.kernel-panic.it/openbsd/mail/
    icon_image                      mail.png
    icon_image_alt                  [Mail]
    statusmap_image                 mail.gd2
}

# Configuration for host proxy.kernel-panic.it
define host {
    use                             generic-dmz-host
    host_name                       proxy
    alias                           Proxy server
    address                         172.16.240.151
}

define hostextinfo {
    host_name                       proxy
    notes                           This is the Squid proxy server
    notes_url                       http://www.kernel-panic.it/openbsd/proxy/
    icon_image                      proxy.png
    icon_image_alt                  [Proxy]
    statusmap_image                 proxy.gd2
}

[...]
```

/var/www/etc/nagios/hosts/firewalls.cfg

```
# Configuration for host fw-int.kernel-panic.it
define host {
    use                             generic-lan-host
    host_name                       fw-int
    alias                           Internal firewalls' CARP address
    address                         172.16.0.202
}

define hostextinfo {
    host_name                       fw-int
    notes                           Virtual CARP address of the internal firewalls
    notes_url                       http://www.kernel-panic.it/openbsd/carp/
    icon_image                      fw.png
    icon_image_alt                  [FW]
    statusmap_image                 fw.gd2
}

# Configuration for host mickey.kernel-panic.it
define host {
    use                             generic-lan-host
    host_name                       mickey
    alias                           Internal Firewall #1
```

```
    address                         172.16.0.200
}

define hostextinfo {
    host_name                       mickey
    notes                           Internal firewall (first node of a two-nodes
CARP cluster)
    notes_url                       http://www.kernel-panic.it/openbsd/carp/
    icon_image                      fw.png
    icon_image_alt                  [FW]
    statusmap_image                 fw.gd2
}

[...]
```

Hosts can optionally be grouped together with the `hostgroup` statement, which has no effect on monitoring, but simply allows you to display the hosts in groups in the CGIs.

/var/www/etc/nagios/hosts/hostgroups.cfg

```
# Domain Name Servers
define hostgroup {
    hostgroup_name                  DNS
    alias                           Domain Name Servers
    members                         dns1,dns2,dns3,dns4
}

# Firewalls
define hostgroup {
    hostgroup_name                  firewalls
    alias                           CARP Firewalls
    members                         mickey,minnie,donald,daisy,fw-int,fw-ext
}

# Web servers
define hostgroup {
    hostgroup_name                  WWW
    alias                           Web Servers
    members                         www1,www2
}
```

3.5 Service definition

Configuring the services to monitor is much like configuring hosts: object inheritance can save you a lot of typing, you may provide additional information with the `serviceextinfo` statement and you can group services together with the optional `servicegroup` statement. Below is the definition of our service template:

/var/www/etc/nagios/generic-services.cfg

```
define service {
# Template name
    name                            generic-service

# Services are normally not volatile
    is_volatile                     0

# Contact groups to notify about problems (or recoveries) with this service
    contact_groups                  Admins

# Enable active checks
```

```
    active_checks_enabled           1
# Time period during which active checks of this service can be made
    check_period                    always
# Time interval (in minutes) between "regular" checks, i.e. checks that
# occur when the service is in an OK state or when the service is in a non-OK
# state, but has already been re-checked max_check_attempts number of times
    normal_check_interval           5
# Time interval (in minutes) between non-regular checks
    retry_check_interval            1
# Number of times that Nagios will repeat a check returning a non-OK state
    max_check_attempts              3
# Enable service check parallelization for better performance
    parallelize_check               1
# Enable passive checks
    passive_checks_enabled          1

# Enable the event handler
    event_handler_enabled           1

# Enable the processing of performance data
    process_perf_data               1

# Enable retention of service status information across program restarts
    retain_status_information       1
# Enable retention of service non-status information across program restarts
    retain_nonstatus_information    1

# Enable notifications
    notifications_enabled           1
# Time interval (in minutes) between consecutive notifications about the
# service being _still_ in non-OK state
    notification_interval           120
# Time period during which notifications about this service can be sent out
    notification_period             always
# Service states for which notifications should be sent out (c=critical,
# w=warning, u=unknown, r=recovery, f=flapping, n=none)
    notification_options            w,u,c,r

    register                        0
}
```

Now, before moving to services definition, we should complete our discussion on passing service-specific arguments to commands by means of the `$ARGn$` macros. As you'll remember, these macros act as placeholders: they expand to the *n*th argument passed to the command in the service definition; for instance, a command definition such as the following expects to be passed two aguments:

```
define command {
    command_name                    some-command
    command_line                    $USER1$/check_something $ARG1$ $ARG2$
}
```

Therefore, to configure a service check to use the above command, we will need to assign the `check_command` variable a string containing the command's short name followed by the arguments, separated by "`!`" characters. E.g.:

```
define service {
    service_description             some-service
    check_command                   some-command!arg-1!arg-2
    [...]
}
```

Now we can proceed to the definition of the actual services:

/var/www/etc/nagios/services/services.cfg

```
# Secure Shell service
define service {
    use                             generic-service
    service_description             SSH
# Short name(s) of the host(s) that run this service. If a service runs on all
# hosts, you may use the '*' wildcard character
    host_name                       *
    check_command                   check-ssh
# This directive is a possible alternative to using the members directive in
# service groups definitions
    servicegroups                   ssh-services
}

define serviceextinfo {
    host_name                       *
    service_description             SSH
    notes                           Availability of the SSH daemon
    icon_image                      ssh.png
    icon_image_alt                  [SSH]
}

# Web service
define service {
    use                             generic-service
    service_description             WWW
    host_name                       www1,www2
    check_command                   check-http!/index.html
}

define service {
    use                             generic-service
    service_description             WWW
    host_name                       mail
    check_command                   check-http!/webmail/index.html
}

define serviceextinfo {
    host_name                       www1,www2,mail
    service_description             WWW
    notes                           Availability of the corporate web sites
    icon_image                      www.png
    icon_image_alt                  [WWW]
}

[...]
```

Just like hosts, services can be grouped together with the `servicegroup` directive:

/var/www/etc/nagios/services/servicegroups.cfg

```
define servicegroup {
    servicegroup_name               www-services
    alias                           Web Services
# The 'members' directive requires a comma-separated list of host and
# service pairs, e.g. 'host1,service1,host2,service2,...'
    members                         www1,WWW,www2,WWW,mail,WWW
}

define servicegroup {
    servicegroup_name               dns-services
```

```
    alias                           Domain Name Service
    members                         dns1,DNS,dns2,DNS,dns3,DNS,dns4,DNS
}

# The members of the following servicegroup are specified with the
# 'serviecegroups' directive in the 'SSH' service definition
define servicegroup {
    servicegroup_name               ssh-services
    alias                           Secure Shell Service
}

[...]
```

Well, the bulk of the work is over now: the last step is configuring the web interface and then we will finally be able to set our Nagios server to work!

4. Setting up the web interface

Nagios doesn't have a specific client application to access the monitoring information; instead, it relies on the Apache web server to provide a very simple yet powerful web interface, accessible to any browser and allowing users to access current status information, browse historical logs, create reports and, if so configured, issue commands to the monitoring daemon.

4.1 CGIs configuration

Nagios' web interface relies on a series of CGI programs written in C. The CGIs read their configuration information from two files: the main configuration file and `cgi.cfg`, located, by default, in the `/var/www/etc/nagios/` directory.

Below is a sample configuration file; pay particular attention when setting the `authorized_for_*` directives, because they allow you to assign special privileges to authenticated users and are, therefore, highly security critical. In the next section, we will review how to create users in Apache.

`/var/www/etc/nagios/cgi.cfg`

```
# Path to the main configuration file (relative to the chroot)
main_config_file=/etc/nagios/nagios.cfg
# Path to the directory where the HTML files reside (relative to the chroot)
physical_html_path=/nagios
# Path portion of the URL used to access the web interface
url_html_path=/nagios

# Disable context-sensitive help
show_context_help=0

# Path to the program used to check the status of the Nagios process
nagios_check_command=/usr/local/libexec/nagios/check_nagios /var/nagios/status.dat 5
'/usr/local/sbin/nagios'

# Enable authentication for the CGIs
use_authentication=1
# Uncomment the following directive to set a default user for unauthenticated
# sessions (strongly discouraged)
#default_user_name=guest

# The 'authorized_for_*' directives define a comma-separated list of
# authenticated web users who can:
# - view system/process information in the extended information CGI:
authorized_for_system_information=nagiosadmin,operator
# - view configuration information in the configuration CGI:
authorized_for_configuration_information=nagiosadmin,operator
# - issue system/process commands via the command CGI:
authorized_for_system_commands=nagiosadmin
# - view status and configuration information for all services
authorized_for_all_services=nagiosadmin,operator
# - view status and configuration information for all hosts
authorized_for_all_hosts=nagiosadmin,operator
# - issue commands for all services via the command CGI:
authorized_for_all_service_commands=nagiosadmin
# - issue commands for all hosts via the command CGI:
authorized_for_all_host_commands=nagiosadmin

# Options for the Status Map and Status World CGIs
statusmap_background_image=smbackground.gd2
default_statusmap_layout=5
default_statuswrl_layout=4
statuswrl_include=myworld.wrl
```

```
# Command to use when attempting to ping a host from the WAP interface
ping_syntax=/sbin/ping -n -c 5 $HOSTADDRESS$

# Time interval (in seconds) between page refreshes
refresh_rate=90

# List of audio files to play in the browser in case of problems. These files
# are assumed to be in the /var/www/nagios/media/ directory
host_unreachable_sound=hostdown.wav
host_down_sound=hostdown.wav
service_critical_sound=critical.wav
service_warning_sound=warning.wav
service_unknown_sound=warning.wav
#normal_sound=noproblem.wav
```

4.2 Apache configuration

The web interface holds particularly sensitive information about network and services and may even allow the execution of commands that directly affect the monitoring daemon. As a consequence, it is strongly recommended that you configure authentication for accessing the CGIs.

User authentication files are managed with the `htpasswd(1)` utility. Note that the first time you run this command, you must supply the "`-c`" option to create the password file:

```
# htpasswd -c /var/www/users/nagios.passwd nagiosadmin
New password: password
Re-type new password: password
Adding password for user nagiosadmin
# htpasswd /var/www/users/nagios.passwd danix@work
New password: password
Re-type new password: password
Adding password for user danix@work
#
```

An authenticated user whose username matches the short name of a contact definition is called an authenticated contact and is automatically granted access to information and commands for those hosts and services for which he is contact (please refer to the documentation for further details about authentication in the CGIs).

Well, now that we have Apache requiring users to authenticate, we should also configure SSL to avoid sending passwords in clear text. Below are the `openssl(1)` commands to create a self-signed certificate (a more detailed discussion about certificate management can be found here).

```
# openssl genrsa -des3 -out server.3des-key 1024
Generating RSA private key, 1024 bit long modulus
..............++++++
.++++++
e is 65537 (0x10001)
Enter pass phrase for server.3des-key: passphrase
Verifying - Enter pass phrase for server.3des-key: passphrase
# openssl rsa -in server.3des-key -out server.key
Enter pass phrase for server.3des-key: passphrase
writing RSA key
# openssl req -new -key server.key -x509 -out server.crt -days 365
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

```
-----
Country Name (2 letter code) []: IT
State or Province Name (full name) []: State
Locality Name (eg, city) []: Locality
Organization Name (eg, company) []: kernel-panic.it
Organizational Unit Name (eg, section) []: Information Technology
Common Name (eg, fully qualified host name) []: nagios.kernel-panic.it
Email Address []: nagios@kernel-panic.it
# chmod 600 server.key
# rm server.3des-key
# mv server.crt /etc/ssl/
# mv server.key /etc/ssl/private/
```

The last step is configuring Apache to actually require authentication and encryption to access the Nagios interface by adding the following lines to the `/var/www/conf/httpd.conf` configuration file:

/var/www/conf/httpd.conf

```
ScriptAlias /cgi-bin/nagios "/var/www/cgi-bin/nagios"

<Directory "/var/www/cgi-bin/nagios">
    SSLRequireSSL

    Options ExecCGI

    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /users/nagios.passwd
    Require valid-user

    Order deny,allow
    Deny from all
    Allow from 127.0.0.1 172.16.0.13
</Directory>

Alias /nagios "/var/www/nagios"

<Directory "/var/www/nagios">
    SSLRequireSSL

    Options None
    AllowOverride None

    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /users/nagios.passwd
    Require valid-user

    Order deny,allow
    Deny from all
    Allow from 127.0.0.1 172.16.0.13
</Directory>
```

4.3 Running Nagios

Well, it looks like we're done with the configuration for now! Then we can make Nagios evaluate our hard work by invoking it with the "`-v`" option:

```
# /usr/local/sbin/nagios -v /var/www/etc/nagios/nagios.cfg

Nagios 2.5
```

```
Copyright (c) 1999-2006 Ethan Galstad (http://www.nagios.org)
Last Modified: 07-13-2006
License: GPL

Reading configuration data...

Running pre-flight check on configuration data...

[...]

Total Warnings: 0
Total Errors:   0

Things look okay - No serious problems were detected during the pre-flight check
#
```

If no errors were detected, then the long-awaited moment has arrived: we are ready to start Nagios! Though not before having created the directory for the lock file (*Note*: if you haven't rebooted since installing the Nagios packages, the `/var/run/nagios/` directory should already exist).

```
# apachectl startssl
/usr/sbin/apachectl startssl: httpd started
# mkdir /var/run/nagios
# chown _nagios:_nagios /var/run/nagios
# /usr/local/sbin/nagios -d /var/www/etc/nagios/nagios.cfg
```

You can check if everything is working fine by connecting to the web interface (`https://`*`your.server.here`*`/nagios/`) or taking a look at the logs.

To finish up, we have to configure the system to start both Apache and Nagios at boot time, by setting the `httpd_flags` variable in the `/etc/rc.conf.local` file:

`/etc/rc.conf.local`

```
httpd_flags="-DSSL"
```

and by adding the following lines to the `/etc/rc.local` file:

`/etc/rc.local`

```
if [ -x /usr/local/sbin/nagios ]; then
    [ -d /var/run/nagios ] || mkdir /var/run/nagios
    chown _nagios:_nagios /var/run/nagios
    /usr/local/sbin/nagios -d /var/www/etc/nagios/nagios.cfg
    echo -n ' nagios'
fi
```

In the next chapter we will take a look at how to extend Nagios with some of its most popular addons.

5. Nagios addons

One of Nagios' key features is its extensibility; new functionality can be easily added thanks to its plugin-based architecture, the external command interface or the Apache web server. In this chapter, we will take a look at a few common issues that can be addressed with some of the most popular addons for Nagios.

5.1 NRPE

Suppose you want Nagios to monitor local services on remote hosts, such as disk space usage, system load or the number of users currently logged in. These are not network services, so they can't be directly checked out with standard plugins: what we would need is some kind of agent to install on remote systems and that Nagios could periodically query for the status of local services.

Well, that's exactly what the Nagios Remote Plugin Executor (NRPE) does: it allows you to execute local plugins on remote hosts! It is made up of two components:

- an agent, running (either standalone or under `inetd(8)`) on the monitored host, which waits for incoming connections, executes the requested checksand returns the status of the local services;
- a plugin, "`check_nrpe`", used by Nagios to query the remote agents.

Both the agent and the plugin are available from the following package:

- nrpe-*x.x.x*.tgz

In addition, you may want to install the Nagios plugins package on the monitored host: this will allow the NRPE agent to take advantage of the standard Nagios plugins to perform local checks. The package installation automatically creates the `_nrpe` user and group that the daemon will run as and copy a sample `nrpe.cfg` configuration file in `/etc/`:

`/etc/nrpe.cfg`

```
# Path to the pid file (ignored if running uder inetd)
pid_file=/var/run/nrpe.pid

# Address to bind to to avoid binding on all interfaces (ignored if running
# under inetd)
server_address=172.16.0.170
# Port to wait connections on (ignored if running under inetd)
server_port=5666

# User and group the NRPE daemon should run as (ignored if running under inetd)
nrpe_user=_nrpe
nrpe_group=_nrpe

# Comma-delimited list of IP addresses or hostnames that are allowed to connect
# to the NRPE daemon (ignored if running under inetd)
allowed_hosts=127.0.0.1,172.16.0.164

# Don't allow clients to specify arguments to commands that are executed
dont_blame_nrpe=0

# Uncomment the following option to prefix all commands with a specific string
# command_prefix=/usr/bin/sudo

# Don't log debugging messages to the syslog facility
debug=0

# Maximum length (in seconds) of executed plugins
command_timeout=60

# Command definitions are in the form
```

```
#
#   command[<command_name>]=<command_line>
#
# Thus, when the NRPE daemon receives a request to execute the command
# 'command_name', it will run the *local* script specified by 'command_line'.
# Note: macros are NOT allowed within command definitions
command[check_users]=/usr/local/libexec/nagios/check_users -w 5 -c 10
command[check_load]=/usr/local/libexec/nagios/check_load -w 15,10,5 -c 30,25,20
command[check_disk1]=/usr/local/libexec/nagios/check_disk -w 20 -c 10 -p /dev/wd0a
command[check_total_procs]=/usr/local/libexec/nagios/check_procs -w 150 -c 200
```

To run NRPE as a standalone daemon, simply type:

```
# /usr/local/sbin/nrpe -c /etc/nrpe.cfg -d
```

and add the following lines to `/etc/rc.local` to start it automatically after reboot:

/etc/rc.local/

```
if [ -x /usr/local/sbin/nrpe ]; then
    /usr/local/sbin/nrpe -c /etc/nrpe.cfg -d
    echo -n ' nrpe'
fi
```

Alternatively, you can run NRPE under `inetd(8)` by adding the following line in `/etc/inetd.conf(8)`:

/etc/inetd.conf

```
nrpe    stream  tcp     wait    _nrpe:_nrpe     /usr/local/sbin/nrpe    nrpe -c
/etc/nrpe.cfg -i
```

and by adding the `nrpe` service in `/etc/services(5)`:

/etc/services

```
nrpe    5666/tcp        # Nagios Remote Plugin Executor
```

and then send the `inetd(8)` daemon the hangup signal, instructing it to re-read its configuration:

```
# pkill -HUP inetd
```

Now, on the Nagios server, you can perform checks using NRPE simply by defining commands such as the following (only make sure that the command name passed to the "`-c`" option has a corresponding command definition in the `nrpe.cfg` file on the remote host!):

/var/www/etc/nagios/commands.cfg

```
define command {
    command_name    check-disk1-nrpe
    command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c check_disk1
}
```

5.2 NSCA

Now suppose you want to monitor the correct execution of a process on a remote host, like a scheduled backup or a crontab job. This is still a "local" service, but, unlike disk space usage or system load, it would probably sound more logical to make it the responsibility of the job itself to notify Nagios of its exit status. That's the perfect job for the Nagios Service Check Acceptor (NSCA), which is a daemon program designed

to accept passive service check results from clients.

NSCA is similar to NRPE in that it is made up of a daemon process and a client application, but now the roles are inverted: the daemon process runs on the Nagios server while remote hosts use the `send_nsca` utility to communicate their status to the daemon. NSCA then forwards the check results to Nagios through the external command interface (so make sure you have enabled external commands in the main configuration file).

5.2.1 Server configuration

NSCA can run either as a standalone daemon or under `inetd(8)`. To install the server component we need to add the following packages on the Nagios server:

- mhash-*x.x.x*.tgz
- libmcrypt-*x.x.x*.tgz
- nsca-*x.x*.tgz

Next, we need to edit the `/etc/nsca.cfg` configuration file:

`/etc/nsca.cfg`

```
# Address to bind to (optional)
server_address=172.16.0.164
# Port to wait connections on
server_port=5667

# User and group the NSCA daemon should run as (ignored if running under inetd)
nsca_user=_nagios
nsca_group=_nagios

# Don't log debugging messages to the syslog facility
debug=0

# Path to the command file
command_file=/var/www/var/nagios/rw/nagios.cmd
# File where to dump service check results if the command file does not exist
alternate_dump_file=/var/www/var/nagios/rw/nsca.dump

# Do not aggregate writes to the external command file
aggregate_writes=0
# Open the external command file in write mode
append_to_file=0

# Maximum packet age (in seconds)
max_packet_age=30

# Password to use to decrypt incoming packets
password=password
# Decryption method (16 = RIJNDAEL-256). It must match the encryption method
# used by the client
decryption_method=16
```

You should set restrictive permissions (600) on the configuration file in order to keep the decryption password protected. To run NSCA as a standalone daemon, simply type:

```
# /usr/local/sbin/nsca -c /etc/nsca.cfg
```

and add the following lines to `/etc/rc.local` to start it automatically after reboot:

`/etc/rc.local`

```
if [ -x /usr/local/sbin/nsca ]; then
```

```
    /usr/local/sbin/nsca -c /etc/nsca.cfg
    echo -n ' nsca'
fi
```

Alternatively, you can run it under `inetd(8)` by adding the following line in `/etc/inetd.conf(8)`:

/etc/inetd.conf

```
nsca    stream  tcp     wait    _nagios:_nagios /usr/local/sbin/nsca    nsca -c
/etc/nsca.cfg --inetd
```

and by adding the `nsca` service in `/etc/services(5)`:

/etc/services

```
nsca    5667/tcp        # Nagios Service Check Acceptor
```

and then send the `inetd(8)` daemon the hangup signal, instructing it to re-read its configuration:

```
# pkill -HUP inetd
```

5.2.2 Client configuration

On the client side, we need to install the following packages:

- mhash-*x.x.x*.tgz
- libmcrypt-*x.x.x*.tgz
- nsca-client-*x.x*.tgz

and edit the encryption parameters in the `/etc/send_nsca.cfg` configuration file:

/etc/send_nsca.cfg

```
# Password to use to encrypt outgoing packets
password=password
# Encryption method (16 = RIJNDAEL-256)
encryption_method=16
```

The `send_nsca` utility reads data from standard input and expects, for service checks, a tab separated sequence of *host name*, *service description* (i.e. the value of the `service_description` directive in the service definition), *return code* and *output*; e.g.:

```
echo "www1\tbackup\t0\tBackup completed successfully" | \
/usr/local/libexec/nagios/send_nsca -H nagios.kernel-panic.it
```

and, for host checks, a tab separated sequence of *host name*, *return code* and *output*; e.g.:

```
echo "router1\t2\tRouter #1 is down" | /usr/local/libexec/nagios/send_nsca -H \
nagios.kernel-panic.it
```

You can override the default delimiter (tab) with `send_nsca`'s "`-d`" option. Now, if everything is working fine, each message received by the NSCA daemon should produce a line like the following in the Nagios log file:

/var/www/var/log/nagios/nagios.log

```
[1167325538] EXTERNAL COMMAND: PROCESS_SERVICE_CHECK_RESULT;www1;backup;0;Backup
completed successfully
```

5.3 NagVis and NDO

NagVis is a visualization addon for Nagios; it can be used to give users a graphical view of Nagios data. It requires the installation of PHP and a few libraries:

- libxml-*x.x.x*.tgz
- t1lib-*x.x.x*.tgz
- jpeg-*x*.tgz
- png-*x.x.x*.tgz
- php5-core-*x.x.x*.tgz
- php5-gd-*x.x.x*-no_x11.tgz
- php5-mysql-*x.x.x*.tgz

Apache is already up and running, so we only need to copy the `php.ini` configuration file, activate the GD and MySQL modules:

```
# cp /usr/local/share/examples/php5/php.ini-recommended /var/www/conf/php.ini
# /usr/local/sbin/phpxs -s
[...]
# /usr/local/sbin/phpxs -a gd
Activating extension : gd
# /usr/local/sbin/phpxs -a mysql
Activating extension : mysql
```

uncomment the following line in `/var/www/conf/httpd.conf`:

```
/var/www/conf/httpd.conf
```

```
AddType application/x-httpd-php .php a httpd.conf
```

and restart Apache:

```
# apachectl restart
/usr/sbin/apachectl restart: httpd restarted
```

5.3.1 Installing NDO and MySQL

Prior to version 1.0, NagVis was able to pull data from Nagios directly from its web interface; now this is not supported anymore and NagVis expects monitoring data to be stored in a MySQL database, thus requiring the intallation of the *Nagios Data Output Utils* (NDOUTILS) addon.

The NDOUTILS addon "*allows you to move status and event information from Nagios to a database for later retrieval and processing*", thus providing the interface between Nagios and MySQL. This addon consists of several parts, but we will need only two of them:

- the NDOMOD event broker module, which is loaded by Nagios at startup and dumps all events and data from Nagios to a Unix or TCP socket;
- the NDO2DB daemon, which is a standalone daemon and reads the output produced by the NDOMOD module through the Unix or TCP socket and dumps it into the database.

First off, we need to install MySQL; the following is the list of the required packages:

- p5-Net-Daemon-*x.x*.tgz
- p5-PlRPC-*x.x*.tgz
- p5-DBI-*x.x*.tgz
- p5-DBD-mysql-*x.x*.tgz
- mysql-client-*x.x.x*.tgz
- mysql-server-*x.x.x*.tgz

Next, we need to download, extract and compile the NDOUTILStarball:

```
# tar -zxvf ndoutils-x.x.x.tar.gz
[ ... ]
# cd ndoutils-x.x.x
# ./configure --disable-pgsql --enable-mysql --with-mysql-lib=/usr/local/lib \
>  --with-mysql-inc=/usr/local/include
[ ... ]
# make
```

Note: if `make` returns errors while compiling the `dbhandlers.c` file, try the following patch (applies to version 1.4b3) by running the following command from outside the ndoutils source tree:

```
# patch -p0 < ndo-openbsd.patch
```

Now we can start MySQL, assign a password to the root account and create the appropriate database and user. The database creation script can be found in the `db/` directory of the extracted tarball.

```
# cp /usr/local/share/mysql/my-medium.cnf /etc/my.cnf
# /usr/local/bin/mysql_install_db
[ ... ]
# mysqld_safe &
Starting mysqld daemon with databases from /var/mysql
# mysqladmin -u root password root
# mysql -u root -p
password: root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.22-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database nagios;
mysql> use nagios;
mysql> \.  db/mysql.sql
[...]
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON nagios.* TO 'ndouser'@'localhost'
    -> IDENTIFIED BY 'ndopasswd';
mysql> \q
```

Now we need to manually copy the binaries and configuration files:

```
# cp src/ndomod-2x.o /usr/local/libexec/nagios/ndomod.o
# cp config/ndomod.cfg /var/www/etc/nagios/
# cp src/ndo2db-2x /usr/local/sbin/ndo2db
# cp config/ndo2db.cfg /var/www/etc/nagios/
```

and edit the NDOMOD configuration file:

`/var/www/etc/nagios/ndomod.cfg`

```
instance_name=default
output_type=unixsocket
output=/var/nagios/rw/ndo.sock
tcp_port=5668

output_buffer_items=5000
buffer_file=/var/nagios/rw/ndomod.tmp

file_rotation_interval=14400
file_rotation_timeout=60

reconnect_interval=15
reconnect_warning_interval=15
data_processing_options=-1
```

```
config_output_options=3
```

and the NDO2DB configuration file:

/var/www/etc/nagios/ndo2db.cfg

```
ndo2db_user=_nagios
ndo2db_group=_nagios

socket_type=unix
socket_name=/var/www/var/nagios/rw/ndo.sock
tcp_port=5668

db_servertype=mysql
db_host=localhost
db_port=3306
db_name=nagios
db_prefix=nagios_
db_user=ndouser
db_pass=ndopasswd

max_timedevents_age=1440
max_systemcommands_age=10080
max_servicechecks_age=10080
max_hostchecks_age=10080
max_eventhandlers_age=44640
```

Then we have to specify the event brokermodule that Nagios must load at startup, by adding the following line to the main configuration file:

/var/www/etc/nagios/nagios.cfg

```
broker_module=/usr/local/libexec/nagios/ndomod.o
config_file=/var/www/etc/nagios/ndomod.cfg
```

and, finally, we can start the NDO2DB daemon and restart Nagios:

```
# /usr/local/sbin/ndo2db -c /var/www/etc/nagios/ndo2db.cfg
# chmod 770 /var/www/var/nagios/rw/ndo.sock
# pkill nagios
# nagios -d /var/www/etc/nagios/nagios.cfg
```

Add the following lines to `/etc/rc.local` to start the NDO2DB daemon on boot:

/etc/rc.local

```
if [ -x /usr/local/sbin/ndo2db ]; then
    /usr/local/sbin/ndo2db -c /var/www/etc/nagios/ndo2db.cfg
    chmod 770 /var/www/var/nagios/rw/ndo.sock
    echo -n ' ndo2db'
fi
```

5.3.2 Configuring NagVis

Now that we have installed all the necessary prerequisites, we can download and extract the NagVis tarball:

```
# cd /var/www/nagios/
# tar -zxvf nagvis-x.x.x.tar.gz
[ ... ]
# mv nagvis-x.x.x nagvis
# chown -R www /var/www/nagios/nagvis/nagvis/etc/
```

Below is a sample NagVis configuration file:

/var/www/nagios/nagvis/nagvis/etc/config.ini.php

```
; <?php return 1; ?>

[global]
language              = "english"
rotatemaps            = 0
maps                  = "dmz,lan"
displayheader         = 1
headercount           = 3
; Use gdlibs (if set to 0 lines will not work, all other types should work fine)
usegdlibs             = 1
refreshtime           = 60

[defaults]
backend               = "ndomy_1"
; Default icons' size (icons can be found in
; /var/www/nagios/nagvis/nagvis/images/iconsets)
icons                 = "std_medium"
recognizeservices     = 1
onlyhardstates        = 0
backgroundcolor       = "#fff"

[wui]
autoupdatefreq        = 25

[paths]
base                  = "/nagios/nagvis/"
htmlbase              = "/nagios/nagvis/"
htmlcgi               = "/cgi-bin/nagios/"
htmldoku              = "http://luebben-home.de/nagvis-doku/nav.html?nagvis/"

[backend_ndomy_1]
backendtype           = "ndomy"
dbhost                = "127.0.0.1"
dbport                = 3306
dbname                = "nagios"
dbuser                = "ndouser"
dbpass                = "ndopasswd"
dbprefix              = "nagios_"
dbinstancename        = "default"
maxtimewithoutupdate = 180

[includes]
header                = "header.nagvis.inc"
```

5.3.3 Maps definition

Now we have to create the images for NagVis to use as the background for each map and put them in the `/var/www/nagios/nagvis/nagvis/images/maps/` directory. You can find a few examples here.

Once the map images are ready, we can tell NagVis where to place objects on the map by creating and editing the maps configuration files. Each map must have a corresponding configuration file (in `/var/www/nagios/nagvis/nagvis/etc/maps/`) with the same name, plus the "`.cfg`" extension. Below is a sample map configuration file; syntax is rather simple, so you can easily tweak it to include your own hosts and services (please refer to the documentation for further details).

/var/www/nagios/nagvis/nagvis/etc/maps/dmz.cfg

```
# The 'global' statement sets some default values that will be inherited by all
# other objects
define global {
# List of users allowed to view this map
    allowed_user=nagiosadmin,operator
# List of users allowed to modify this map via the web interface
    allowed_for_config=nagiosadmin
# Defaul iconset
    iconset=std_medium
# Background image
    map_image=dmz.png
# Backend ID
    backend_id=ndomy_1
}

# Display the status of our 'www1' web server
define host {
    host_name=www1
# Coordinates of the host on the map
    x=268
    y=166
# Set this to '1' if you want the host status to also include the status
# of its services
    recognize_services=0
}

# Display the status of the 'WWW' service on the 'www1' web server
define service {
    host_name=www1
    service_description=WWW
    x=588
    y=165
# As you can see, 'global' options can be overridden in subsequent objects
    iconset=std_small
}

# Display the worsest state of hosts in the 'WWW' hostgroup
define hostgroup {
    hostgroup_name=WWW
    x=298
    y=363
    recognize_services=1
}

# Display the worsest state of services in the 'www-services' servicegroup
define servicegroup {
    servicegroup_name=www-services
    x=609
    y=363
}

# Display the worsest state of objects represented in another NagVis map
define map {
    map_name=lan
    x=406
    y=323
}

# Draw a textfield on the map
define textbox {
# Text may include HTML
```

```
    text="This is the DMZ network"
    x=490
    y=394
    w=117
}
```

To allow the web interface to modify NagVis' configuration, make sure that all configuration files belong to, and are writable by, the `www` user.

```
# chown www /var/www/nagios/nagvis/nagvis/etc/maps/*.cfg
# chmod 644 /var/www/nagios/nagvis/nagvis/etc/maps/*.cfg
```

6. Writing your own Nagios plugins

Plugins are executable files run by Nagios to determine the status of a host or service. By default, Nagios comes with a very rich set of official plugins that should cover most people's needs; in addition, you can find lots of contributed plugins on the Nagios Exchange website, some of which are also available via OpenBSD's packages and ports system.

However, despite the abundance of plugins, there may be occasions in which no existing plugin is suitable for monitoring a particular service, thus forcing you to write a fully custom plugin, tailored to your exact needs. Luckily, this is a very simple task!

Nagios doesn't bind you to a specific programming language: plugins may be either compiled C programs or interpreted scripts, in Perl, shell or Python. Nagios doesn't mess with the internals of plugins; however, it asks developers to follow a few basic guidelines, just for standard's sake.

6.1 Command line options

A plugin's command line must follow some specific requirements:

- positional arguments are strongly discouraged;
- all plugins should provide a "`-V`" command-line option (and "`--version`" if long options are enabled) to display the plugin's revision number;
- the "`-?`" option, as well as any incorrect option, displays a short usage statement that should fit on a standard 80x25 terminal;
- the "`-h`", or "`--help`", option displays detailed help information;
- the "`-v`", or "`--verbose`", option adjusts the verbosity level; multiple "`-v`" options (up to 3) should increase the verbosity level, as described in the official guidelines;
- There are a few other reserved options that should not be used for other purposes:
    - "`-t`" or "`--timeout`" (plugin timeout);
- "`-w`" or "`--warning`" (warning threshold);
- "`-c`" or "`--critical`" (critical threshold);
- "`-H`" or "`--hostname`" (name of the host to check).

6.2 Plugin return codes

Nagios determines the status of a host or service based on the return code of the plugin. Valid return codes are:

Numeric value	Service/Host status	Service Status description	Host status description
0	Ok/Up	The plugin was able to check the service and it seemed to work correctly	The host is up and replied in acceptable time
1	Warning	The plugin was able to check the service, but it didn't seem to work correctly or it exceeded some "warning" threshold	The host is up, but some "warning" threshold was exceeded
2	Critical/Down	The service was not running or it exceeded some "critical" threshold	The host is down or some "critical" threshold was exceeded
3	Unknown	Invalid command line arguments were supplied or an internal error occurred	Invalid command line arguments were supplied or an internal error occurred

The warning and critical thresholds are usually set via command line options (see above).

6.3 A sample plugin script

Just a couple of notes before moving to a practical example:

- plugins can access macros as environment variables; such variables have the same name as the corresponding macros, with "NAGIOS_" prepended. For instance, the "$HOSTNAME$" macro will be accessible through the "NAGIOS_HOSTNAME" environment variable;
- always specify the full path of any system commands run from your plugins.

Well, so let's see, as an example, what a plugin to monitor the amount of free memory on the local machine could look like:

/usr/local/libexec/nagios/check_free_mem.sh

```
#!/bin/ksh

##############################################################################
# Sample Nagios plugin to monitor free memory on the local machine           #
# Author: Daniele Mazzocchio (http://www.kernel-panic.it/)                   #
##############################################################################

VERSION="Version 1.0"
AUTHOR="(c) 2007 Daniele Mazzocchio (danix@kernel-panic.it)"

PROGNAME=`/usr/bin/basename $0`

# Constants
BYTES_IN_MB=$(( 1024 * 1024 ))
KB_IN_MB=1024

# Exit codes
STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3

# Helper functions ############################################################

function print_revision {
   # Print the revision number
   echo "$PROGNAME - $VERSION"
}

function print_usage {
   # Print a short usage statement
   echo "Usage: $PROGNAME [-v] -w <limit> -c <limit>"
}

function print_help {
   # Print detailed help information
   print_revision
   echo "$AUTHOR\n\nCheck free memory on local machine\n"
   print_usage

   /bin/cat <<__EOT

Options:
-h
   Print detailed help screen
-V
   Print version information

-w INTEGER
   Exit with WARNING status if less than INTEGER MB of memory are free
```

```
-w PERCENT%
   Exit with WARNING status if less than PERCENT of memory is free
-c INTEGER
   Exit with CRITICAL status if less than INTEGER MB of memory are free
-c PERCENT%
   Exit with CRITICAL status if less than PERCENT of memory is free
-v
   Verbose output
__EOT
}

# Main ########################################################################

# Total memory size (in MB)
tot_mem=$(( `/sbin/sysctl -n hw.physmem` / BYTES_IN_MB))
# Free memory size (in MB)
free_mem=$(( `/usr/bin/vmstat | /usr/bin/tail -1 | /usr/bin/awk '{ print $5 }'` \
 / KB_IN_MB ))
# Free memory size (in percentage)
free_mem_perc=$(( free_mem * 100 / tot_mem ))

# Verbosity level
verbosity=0
# Warning threshold
thresh_warn=
# Critical threshold
thresh_crit=

# Parse command line options
while [ "$1" ]; do
   case "$1" in
       -h | --help)
           print_help
           exit $STATE_OK
           ;;
       -V | --version)
           print_revision
           exit $STATE_OK
           ;;
       -v | --verbose)
           : $(( verbosity++ ))
           shift
           ;;
       -w | --warning | -c | --critical)
           if [[ -z "$2" || "$2" = -* ]]; then
               # Threshold not provided
               echo "$PROGNAME: Option '$1' requires an argument"
               print_usage
               exit $STATE_UNKNOWN
           elif [[ "$2" = +([0-9]) ]]; then
               # Threshold is a number (MB)
               thresh=$2
           elif [[ "$2" = +([0-9])% ]]; then
               # Threshold is a percentage
               thresh=$(( tot_mem * ${2%\%} / 100 ))
           else
               # Threshold is neither a number nor a percentage
               echo "$PROGNAME: Threshold must be integer or percentage"
               print_usage
               exit $STATE_UNKNOWN
           fi
           [[ "$1" = *-w* ]] && thresh_warn=$thresh || thresh_crit=$thresh
           shift 2
```

```
            ;;
        -?)
            print_usage
            exit $STATE_OK
            ;;
        *)
            echo "$PROGNAME: Invalid option '$1'"
            print_usage
            exit $STATE_UNKNOWN
            ;;
    esac
done

if [[ -z "$thresh_warn" || -z "$thresh_crit" ]]; then
    # One or both thresholds were not specified
    echo "$PROGNAME: Threshold not set"
    print_usage
    exit $STATE_UNKNOWN
elif [[ "$thresh_crit" -gt "$thresh_warn" ]]; then
    # The warning threshold must be greater than the critical threshold
    echo "$PROGNAME: Warning free space should be more than critical free space"
    print_usage
    exit $STATE_UNKNOWN
fi

if [[ "$verbosity" -ge 2 ]]; then
    # Print debugging information
    /bin/cat <<__EOT
Debugging information:
  Warning threshold: $thresh_warn MB
  Critical threshold: $thresh_crit MB
  Verbosity level: $verbosity
  Total memory: $tot_mem MB
  Free memory: $free_mem MB ($free_mem_perc%)
__EOT
fi

if [[ "$free_mem" -lt "$thresh_crit" ]]; then
    # Free memory is less than the critical threshold
    echo "MEMORY CRITICAL - $free_mem_perc% free ($free_mem MB out of $tot_mem MB)"
    exit $STATE_CRITICAL
elif [[ "$free_mem" -lt "$thresh_warn" ]]; then
    # Free memory is less than the warning threshold
    echo "MEMORY WARNING - $free_mem_perc% free ($free_mem MB out of $tot_mem MB)"
    exit $STATE_WARNING
else
    # There's enough free memory!
    echo "MEMORY OK - $free_mem_perc% free ($free_mem MB out of $tot_mem MB)"
    exit $STATE_OK
fi
```

7. Appendix

7.1 References

- OpenBSD installation guide
- The OpenBSD packages and ports system
- Nagios official documentation
- NagiosExchange, the central repository for Nagios plugins
- mod_ssl F.A.Q. list
- NagVis official documentation
- Nagios plug-in development guidelines

7.2 Bibliography

- *Pro Nagios 2.0*, James Turnbull, Apress, 2006
- *Nagios System and Network Monitoring*, W. Barth, No Starch Press, 2006
- *FreeBSD and OpenBSD Security*, Y. Korff, P. Hope & B. Potter, O'Reilly, 2005