Chapter 5

Add-ons and Enhancements

Solutions in this chapter:

- **Checking Private Services when SNMP Is Not Allowed**
- **Visualization**
- **PNP—PNP Not PerfParse**
- **Cacinda**
- **NLG—Nagios Looking Glass**
- **SNMP Trap Handling**
- **SNMPTT**
- **NagTrap**
- **Text-to-Speech for Nagios Alerts**

☑ **Summary**

Introduction

Thanks to a very enthusiastic user/developer base, there is a large and continually growing set of add-ons and enhancements available for Nagios. These projects manipulate, massage, ingest, and display data in ways that extend far beyond the capabilities of the core Nagios system. In this chapter we discuss a number of enhancements and add-ons we feel are very useful in a larger environment.

Checking Private Services when SNMP Is Not Allowed

NRPE

Download it from: http://www.nagios.org/download/addons/

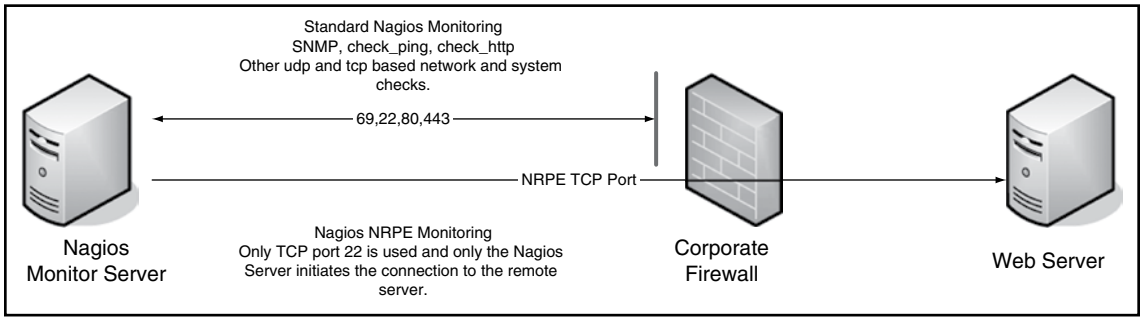
Windows version: http://www.miwi-dv.com/nrpent/

Monitoring and security requirements often clash when the monitoring server sits in one security zone and the managed server sits in a different security zone. For example, web servers are often hosted in DMZ zones. With most DMZs, SNMP traffic is not allowed, yet IT staff is still expected to monitor systems in the DMZ and identify issues early. Outages to servers in a high-visibility security zone like a DMZ means trouble for most companies as quite often corporate web servers are located in a DMZ.

DMZs and Network Security

NRPE allows Nagios to monitor systems that sit in security zones where traffic is only allowed to flow either from the client to the Nagios server or from the Nagios server to the managed client. Security policies generally dictate that systems in high-visibility security zones (like a DMZ) may not initiate communications with systems sitting in more trusted zones; however, often a monitoring system in the more trusted security zone will be allowed to communicate with clients in the DMZ for monitoring purposes. These same policies also often prohibit the use of SNMP v1 or SNMP v2c as the two are not encrypted and depend on UDP as their layer 4 transport. NRPE uses TCP and can encrypt traffic between the server and client using SSL (Figure 5.1).

Figure 5.1 NRPE Information Flow across a Security Zone



In order to use NRPE in your network, you should make the following changes to the firewall that sits between the Nagios server and the DMZ host being monitored:

1. Create a firewall ACL that allows the Nagios host to communicate with the DMZ host over TCP port 5666, which is the default port the NRPE client uses to listen for incoming connections requests.
2. Connections initiated from the DMZ Web server to the internal network are not permitted.
3. All traffic between the Nagios server and Web server is encrypted using TLS (SSL) encryption.
4. On the DMZ Web server, tcp_wrappers should be configured to only accept NRPE connections from the internal Nagios server or firewall IP address (if NAT is in use) on TCP port 5666.

Security Caveats

If NRPE is run with the configuration parameter *dont_blame_nrpe* set to 1, it will allow remote clients to send arbitrary command arguments to the NRPE daemon.
If NRPE is allowed to run in this mode it essentially acts as a remote shell for any server that can connect to the NRPE client. We highly recommend that you do not enable this mode of NRPE.

NRPE Details

NRPE allows the Nagios server to run Nagios plug-ins on a managed client. To install NRPE, you will need both the Nagios plug-ins and OpenSSL installed on the Nagios server. Install the OpenSSL libraries on the Nagios server to allow the *check_nrpe* plugin to communicate with remote NRPE clients using TLS / SSL. The NRPE documentation is quite good and from this point on we assume that you have an NRPE daemon running on a remote server and we assume that check_nrpe has been compiled and installed on your Nagios server using the default settings.

NRPE in the Enterprise

By default, NRPE is configured to monitor basic system metrics. We see many configurations where basic measures of system health are checked: disk usage, memory utilization and whether or not critical network-based system services are running (SSH, Apache)—but often administrators do not take the time to fully exploit NRPE to check higher-level application functionlity.

Scenario 1: The Internet Web Server

You have a small business Web server. You are running NRPE on the host and checking basic services based on your web_server template. This includes disk space utilization, memory utilization, number of active Apache processes, and number of active tomcat Java processes. With these settings in place you know you will be alerted when there is a critical service failure. Do not stop here; there are other types of failures that can mean big problems for your web server and NRPE can help you spot them.

This is where we use NRPE as the eyes of our environment. For example, we can use *check_http* to check for a variety of problems with our web server. We can use the command *check_http -e -N <server-name>* to verify that the layer 7 firewall we use stops invalid HTTP requests from reaching our web server. We can use the -s option of check_http to verify that critical parts of our web site return the content we expect them to contain. HTTP content returned from the web server. The value of this type of checking is that we verify that our application is functioning as expected from the perspective of a customer.

When using NRPE on a DMZ Web server, it is best to identify standard text in the Web page near the bottom of the document (not in a header or footer) that you can check with check_http's -r or -s options. By checking for content on a web page, you can identify when problems occur in the application that impact your users.

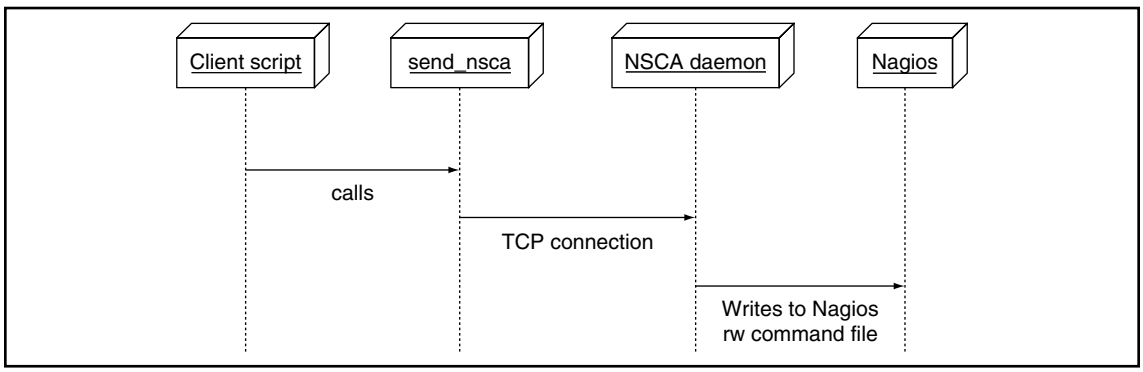
NSCA

Download it from: http://www.nagios.org/download/addons/

NSCA (Nagios Service Check Acceptor) allows you to configure devices and applications to send asynchronous events to Nagios. The event information can be encrypted in a variety of ways, and a password can be required in order for the server-side of NSCA to accept the incoming event. This framework is a terrific alternative or supplement to having devices send out SNMP traps and then ingesting them into Nagios using a framework like SNMPTT.

NSCA can be downloaded from the main Nagios site (www.nagios.org). The installation instructions are easy to follow and installation is simple. Once installed, the NSCA server can be run on the Nagios server as a daemon (the package comes with a SysV-style init script that can be placed in /etc/init.d/ to start and stop the NSCA daemon). The daemon will listen for incoming NSCA requests sent by the client-side send_nsca utility. When received, requests are authenticated using one of over a dozen encryption algorithms along with an optional administrator-configured password. The password chosen must be entered in the NSCA server configuration file and every NSCA client configuration file. If you are not using NSCA in a trusted environment, we highly recommend creating a complex password and using a strong encryption algorithm. We also recommend that in untrusted environments you use firewall rules to limit which client servers can communicate with your NSCA daemon. Once an NSCA client and server are configured, you can use the send_nsca utility to send events from the client for anything you can script—from hard disk errors, to application-level events, to security events (Figure 5.2).

Figure 5.2 NSCA Information Flow



Visualization

While Nagios has a very easy to read and well-designed network map front end, there are times when a logical view by device is not what your users want. For example, developers and application-specific support personnel might prefer a view that shows status organized logically and by service as that is the focus of their work—ensure the application they are responsible for is running and responding in a reasonable amount of time.

NagVis

Download it from: http://www.nagvis.org/

In larger, heterogeneous organizations, the number of system and host problems that occur in any given day can easily clutter up a screen, even a big one. There will also be staff members in your organization who do not want to see network maps or tables of problems, be that because they find the data intimidating or because they only care about application health as opposed to system and network health. Enter NagVis, another very useful add-on that lets you visualize data from Nagios in a variety of useful and creative ways. NagVis uses a PHP-based front and can either read host and service data directly from the Nagios CGIs (not recommended) or from a MySQL database that has been populated with Nagios status information using the NDO Utils package. In this section, we give you tips on installing and configuring this add-on, and show examples of how to make best use of NagVis.

We recommend (the NagVis team does, too) that you use the database back end for service and host data. For large groups, this greatly reduces the overhead of NagVis on a system and provides much better performance than the CGI-based back end does. Configuring NagVis to use a database involves the following steps:

1. Enable the event broker in Nagios.
2. Download and install NDO Utils.
3. Download, install, and configure NagVis.
4. We cover each of these steps in this section.

Enable the Event Broker in Nagios

If you did not enable the event broker functionality in Nagios 3 when you installed it, you can re-run *configure* in the Nagios 3 distribution directory and pass it the switch *--enable-event-broker*. After running *configure*, re-run *make* and then *make install-base* and *make install-commandmode* to re-install just the Nagios daemon with the event

broker capability enabled. You then must tell Nagios to broker everything through the event broker by setting the following parameter in your nagios.cfg file:

```
event_broker_options=-1
```

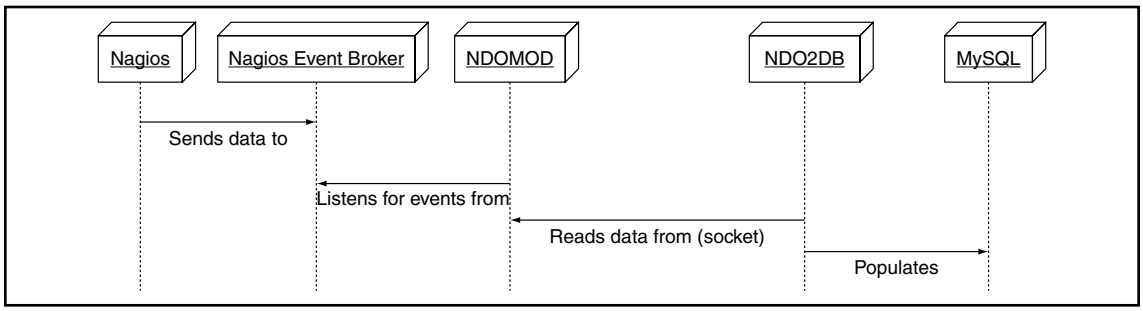
Finally, you must restart Nagios so your new event-broker enabled Nagios daemon is running.

Install the NDO Utils Package

Download it from: http://www.nagios.org/download/addons/

The NDO Utils package captures service and host configuration information and host and service events from Nagios and populates a database with the information. It is available for download from the main Nagios site under the Downloads section. While the module developers warn in the README that it is experimental/beta quality, we have not experienced any problems with this module in a production environment. NDO consists of two parts: the NDOMOD event broker module that receives events from Nagios and then makes the events available to remote clients over a TCP/IP socket or appends data to a file on the Nagios server, and a data ingestion program that processes the data. NDO comes with two programs that read events produced by the NDOMOD module; FILE2SOCK reads the NDOMOD data from the file output of NDOMOD and sends it over the network to a remote instance of NDO2DB, and NDODB, which can receive data from FILE2SOCK or NDOMOD directly and populate a database with the information. If you have the good fortune of having a separate machine to use just for visualization, make use of FILE2SOCK and offload the data ingestion and visualization functions onto the second machine. We will assume for the rest of this section that you are running Nagios and NagVis on the same host, as that is likely to be the most common situation Nagios integrators will encounter (Figure 5.3).

Figure 5.3 How Nagios, NDOMOD, and NDO2DB Interact to Populate a Database with Nagios Events



Follow the instructions in the README that comes with NDO Utils carefully and you will have NDOMOD and NDO2DB installed and running in no time. If you have multiple versions of MySQL installed on your host or have MySQL installed in a nonstandard location, use the command *mysql_config --libs* to determine which directory your MySQL libraries are installed in and then pass that directory to the configure script as *--with-mysql-lib=/path/to/mysql/lib*. also keep in mind that if you do not have Nagios installed in the default location (*/usr/local/nagios*), you need to pass the base path of your installation to the NDODB configure script using the *--prefix* option of configure. Finally, if you are not using the user and group *nagios* for your Nagios daemon, pass the names of the group and user to configure using the *--with-ndo2db-user* and *--with-ndo2db-group* switches.

After you make NDO, install the NDOMOD and NDO2DB files according to the instructions in the README. The README does not currently include an init script, so here is one you can use on Redhat-like Linux systems to control the NDO2DB daemon:

```
#!/bin/bash

# chkconfig: - 50 50

# description: NDO2DB - Nagios NDO to database daemon
#
# processname: ndo2db
# pidfile: /var/run/ndo2db.pid

NBASE=/home/nagios
ndo2db=$NBASE/bin/ndo2db

# source function library
. /etc/init.d/functions

case "$1" in
 start)
     echo -n $"Starting ndo2db: "
     daemon $ndo2db -c $NBASE/etc/ndo2db.cfg
     RETVAL=$?
     Echo
 ;;
stop)
   echo -n $"Stopping ndo2db: "
   killproc $ndo2db
   RETVAL=$?
   echo
;;

restart)
   echo -n $"Restarting ndo2db: "
```

```
   $0 stop
   sleep 30
   $0 start
;;
status)
   status ndo2db
   RETVAL=$?
;;
*)
   echo $"Usage: $0 {start|stop|status|restart}"
   RETVAL=1

Esac

exit $RETVAL
```

After installing NDO2DB and NODMOD by following the README file, restart Nagios to activate the NDOMOD module, and then start NDO2DB to activate the NDO database daemon:

```
/etc/init.d/ndo2db start
```

If everything is installed and configured properly, you will see a line like the following in your /var/log/messages file:

```
Jan 6 17:44:31 hostname nagios: ndomod: Successfully flushed 117 queued
items to data sink.
```

Download and Install NagVis, Configure It to Use the Database Back End You Set up with NDO

Now we will install and configure NagVis. Download the latest stable distribution from http://www.nagvis.org and follow the instructions in the INSTALL file to install NagVis. Note that when the instructions mention "Move the nagvis directory tree," they mean the distribution directory, not the nagvis directory you see under the distribution directory. Following the instructions, configure NagVis to use the database you set up when installing and configuring NDO utils on your system. Finally, integrate links to the project into the Nagios GUI to make it easy for users to find by adding code like the following to <path-to-nagios>/share/side.html:

```
  <br/>
  <br/>
```

```
<table width="150">
 <tr>
  <td>
   <table width="100%" class="NavBarTitle" cellspacing="0">
    <tr>
   <td class="NavBarTitle">Add-Ons</td>
  </tr>
   </table>
  </td>
 </tr>
</table>


<br/>


<table width="150" border="0" cellpadding=0 cellspacing=0>
      <tr>
     <td width=13><img src="images/greendot.gif" width="13" height="14"
         name="config-dot"></td>
     <td nowrap width=134><a href="/nagios/nagvis/" target="main"
         onMouseOver="switchdot('nagvis-dot',1)"
         onMouseOut="switchdot('nagvis-dot',0)"
         class="NavBarItem">NagVis Maps</a></td>
      </tr>
</table>
```

Now that you are done with installation, the sky is the limit. The NagVis home page has many examples of the kinds of visualization you can do with NagVis. Many people use NagVis to show network and system paths in a manner that is more visually appealing than the default Nagios status map; however, we have found it works very well for showing high-level application and service status as well.

Example one: You work in a software development shop and find yourself monitoring multiple development and integration environments. In these environments, it is typical for services to go up and down regularly and for a number of people to want to know the status of services in each environment at-a-glance. Your users are most likely not going to appreciate receiving large numbers of emails about service and host outages, as they are expecting services and hosts to be relatively instable. This is the type of situation in which NagVis shines; at a glance, developers, system administrators, and system integrators can easily see the status of multiple environments from

one place. As with any enhancement to Nagios, talk to your users and customers when you install this add-on; educate them on what it can do and solicit their feedback on how it can be used to best meet their needs.

Example two: Network Operations Center (NOC). In many NOCs, staffing is 24×7 and staff are hired and quit on a regular basis. Educating and training staff about applications on a network or set of networks quickly is very important. Using NagVis, you can create maps that show the status of applications and make it visually very easy for staff to learn about the relationships between the pieces of the applications without having to spend hours reading system and network documentation (most NOC staff do not ever get the time to do that). Additionally, this has the benefit of making it easy for NOC staff to communicate to application- or network-specific personnel what has gone wrong and where that piece sits within an application.

PNP—PNP Not PerfParse

Download it from: http://www.pnp4nagios.org/pnp/

PNP is a gem of an add-on; it allows a Nagios administrator to easily add RRD-style graphs and efficient long-term trending (four years per metric by default) capabilities to Nagios. Like the rest of the Nagios configuration framework and much like Cacti (at least one piece of PNP is borrowed from Cacti), this graphing framework makes extensive use of templates and is easy to customize. It consists of a PHP-based front end that lives under the Nagios web document directory and a Perl/C based back end that processes performance data produced by Nagios using RRDtool (www.rrdtool.org). RRDtool manipulates RRD files efficiently, and includes a powerful graphing language. The RRD files PNP produces are space efficient; a single metric takes up approximately 400k of space for four years' worth of trending. If you are familiar with Cacti, once you have PNP installed and integrated with Nagios you will find it very easy to use and very useful. We highly recommend you choose the *Batch Mode + NPCD* of operation (explained later).

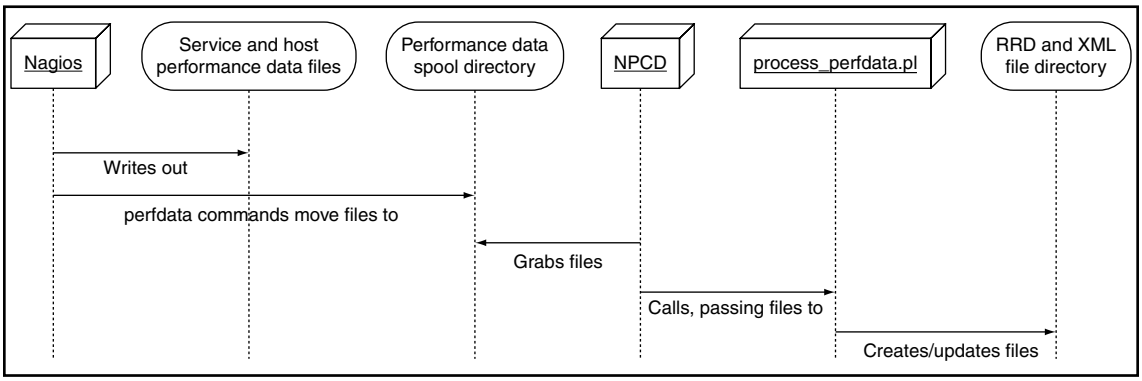
In this section, we provide recommendations on the installation and configuration of PNP. We then describe how to add performance data to your scripts in a format Nagios and PNP can read. We finish with an example of a custom PNP template to make an easy-to-read, useful graph from the output of a custom Nagios check (CPU utilization in this case). Examples in this section all assume PNP is using the batch mode and NPCD daemon mode of operation.

There are three ways to configure Nagios and PNP to get data from Nagios into PNP:

- **Default mode** Configure Nagios to directly call the PNP RRD (round-robin database) file creation script (process_perfdata.pl) with performance data from Nagios every N seconds.
- **Batch mode** Configure Nagios to create host and service performance data files, and add process_perfdata.pl commands to the service and host file processing command directives of Nagios. process_perfdata.pl will then be called by Nagios to read and process the service and host performance files and update the RRD and XML files the PNP Web interface reads.
- **Batch mode + NPCD** Configure Nagios to create host and service performance data files, and to move those configuration files to a spool directory every N seconds. Then, compile and run npcd, a C-based daemon included with the PNP package. The daemon reads the performance files from the spool directory, and then spawns N instances of the graphing script at a time to ingest the files into RRD databases.

Option three provides the best scalability for large organizations and will allow your Nagios server to process the most performance data at a time without impacting the operation of Nagios, as in this mode, all Nagios does is create performance data files and periodically move them to a spool directory (Figure 5.4).

Figure 5.4 How PNP Processes Nagios Performance Data



For best RRD creation and update performance, make sure you have the RRDs Perl module (comes with the source distribution of RRDtool) installed. You can check to see if you have this module by running the command:

```
perl -MRRDs -e 1
```

If you see no output from Perl when you run this command, the RRDs module is installed. If you see an error message, you might need to download and compile RRDtool from source yourself; the options *--with-perl* and *--with-perl-site-install* will enable the RRDs module and place it in your site-wide Perl library directory.

Configuring Nagios to create the performance data files the PNP npcd daemon reads and setting up base PNP configuration files is fairly easy. The documentation on the PNP site is very well done, so we will not repeat basic installation instructions here. We do recommend you follow the documentation carefully, as the process will fail if you miss any steps. Especially critical are the host and service performance data templates; make sure you enter those into your configuration file correctly.

Once you have PNP installed and configured correctly, you need to understand how to modify your scripts to produce performance data output in a manner Nagios understands. Nagios 3 parses any data in the output of a service check that follows a pipe "|" symbol as performance data. Performance data can include multiple metrics, and each metric can contain discreet warning, critical, minimum, and maximum values. From the Nagios perfdata documentation (http://nagiosplug.sourceforge.net/developer-guidelines.html#AEN203):

```
| 'label1'=value;[warn];[crit];[min];[max] … 'labelN'=value;[warn];[crit];[min];[max]
```

Sample output from a service check that uses this format:

```
ROBOTIC_TEST OK: 340 results in 3.034 seconds |'Retrieve home page'=1s;3,5;;;
'Perform search'=15s;20;30;;
```

From the output of this check, we know:

- The test performed acceptably and involved a Web search that returned 340 results.
- Home page retrieval took 1 second, has a warning threshold of 3 seconds, a critical threshold of 5 seconds, and no min or max output values.
- Logout took 2 seconds, has a warning threshold of 5 seconds, a critical threshold of 15 seconds, and no min or max output values.

NOTE

Your performance data can include data that goes well beyond the single status your check returns, yielding a very nice separation of fault management data vs. service performance trending data.

Now that we have our performance data in a format Nagios and PNP will recognize, we need to configure the service so Nagios processes performance data from it by setting the service parameter *process_perf_data* to 1 and restarting Nagios. We then create a PNP graph template so we can see all the performance data from our Nagios check in one place. PNP templates are PHP files that generate a custom command line that will be passed to rrdtool along with a string containing the RRD tool language necessary to create your graph. For more information on the RRD tool and RRD language, see www.rrdtool.org/.

PNP first looks for templates under <nagios-share>/pnp/templates.dist, and then under <nagios-share>/pnp/templates for a PHP template file to use to display a custom graph for a service. Custom template files should be named for the service with which they are associated. If no custom template file is found, the default template default.php will be used.

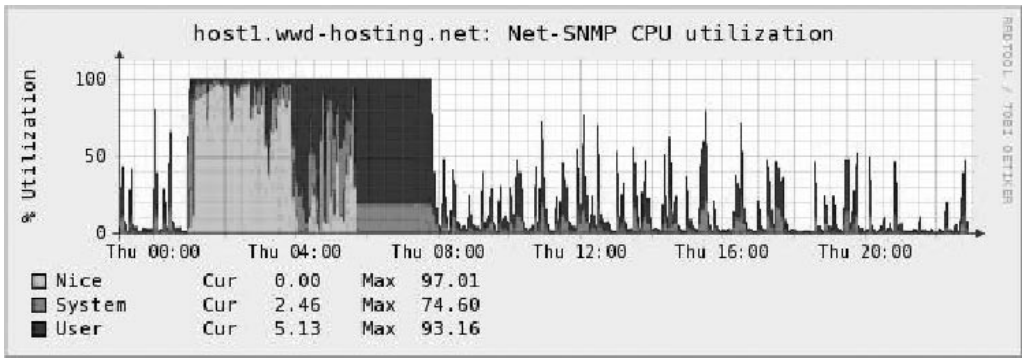
We will place our custom check graph template in the directory <nagios-share>/pnp/templates. An easy way to get started on a custom template is to find an existing template under the templates.dist directory, copy it to the templates directory, and then modify it to meet your needs. For this check, we copied the check_load.php template, as it shows multiple RRD sources on a single graph. Here is the template we created based on the Net-SNMP CPU check shown in Chapter 4 in this book:

```
<?php
$opt[1] = "--vertical-label '% Utilization' " .
     "--title '$hostname: $NAGIOS_SERVICEDESC' -X 0 -M";

$colors = array(
   'idle' => '00FF00',
   'user' => '0000FF',
   'system' => 'FF0000',
   'kernel' => '999999',
   'interrupt' => '999900',
   'wait' => 'FF9900',
   'nice' => '00FF00'
);

$graph = "";

$i = 1;

foreach ($DS as $d) {
   if ($NAME[$d] == 'idle') {
       continue;
   }

   $type = 'STACK';
```

```
   if ($i == 1) {
       $type = 'AREA';

   }
   $i++;

   $label = sprintf("%-9s", ucfirst($NAME[$d]));
   $graph .= <<<EOF
DEF:var$d=$rrdfile:$DS[$d]:AVERAGE
$type:var$d#{$colors[$NAME[$d]]}:"$label"
GPRINT:var$d:LAST:" Cur %6.2lf "
GPRINT:var$d:MAX:"Max %6.2lf \\n"

EOF;
   if ($i == 1) {
       $type = 'AREA';
   }
   $i++;

   $label = sprintf("%-9s", ucfirst($NAME[$d]));

   $graph .= <<<EOF
DEF:var$d=$rrdfile:$DS[$d]:AVERAGE
$type:var$d#{$colors[$NAME[$d]]}:"$label"
GPRINT:var$d:LAST:" Cur %6.2lf "
GPRINT:var$d:MAX:"Max %6.2lf \\n"

EOF;

}

$def[1] = preg_replace('/[\r\n]/', ' ', $graph);

?>
```

Now we browse to http://example.org/pnp?host=myhost to see our new graph. Figure 5.5 is a screenshot of what the output looks like.

Figure 5.5 PNP-based Net-SNMP CPU Utilization Graph



The power of this framework is that this graph will be generated for any new hosts you add to Nagios that are associated with this Nagios check.

Finally, integrate PNP into the Nagios HTML GUI by adding some custom HTML code to <path-to-nagios>/share/side.html. This example adds HTML that will take you to the custom pages section of PNP when the "PNP Graphs" link is clicked (just remove the word *page* from the link to go to the default PNP index page). Add the code to the bottom of the file, before the ending </body> tag:

```
<br/>
<table width="150" border="0" cellpadding=0 cellspacing=0>
  <tr>
       <td width=13><img src="images/greendot.gif" width="13" height="14"
       name="config-dot"></td>
   <td nowrap width=134><a href="/nagios/pnp/index.php?page" target="main"
       onMouseOver="switchdot('pnp-dot',1)"
       onMouseOut="switchdot('pnp-dot',0)"
       class="NavBarItem">PNP Graphs</a></td>
  </tr>
</table>
```

Cacinda

Download it from: http://cacinda.sf.net/

If your shop is more Cacti-oriented and you use Nagios purely for service and status checks or you have a very large Cacti install and are just starting to use Nagios, Cacinda can help you create HTML-based dashboards that integrate the information from Cacti and Nagios along with live SNMP data from your devices. Warning: Cacinda is an alpha release; when it is stable, it will be released as a Cacti plug-in. Despite its new status, getting it set up and running is straightforward. Like most Nagios and Cacti add-ons, Cacinda uses templates to allow you to handle multiple device types easily and create customized HTML views for metrics from a variety of SNMP devices.

Installation of Cacinda is straightforward; just download the source code from http://cacinda.sf.net, untar it into a directory under your Web server's document root, and configure it by editing the config.pnp file that comes with the distribution. The first section of the Cacinda configuration file is used to configure Cacinda so it can select data from your Cacti database. The second section is to tell Cacinda the base URL for Nagios, and configure a username and password to use to log in to Nagios

if Nagios requires authentication. Finally, you may configure a search and replace string pair that will help you match Cacti host names to Nagios names. For example, if you use fully qualified host names in Cacti, but short names in Nagios, you can use this to strip the domain portion of each Cacti host name so the names will work with Nagios.

Cacti uses templates for each device type you wish to display a dashboard for; the current release supports Cisco devices, Microsoft SNMP agents, and Net-SNMP agents. Cacti checks the SNMP sysDescr.0 and sysObjectId.0 SNMP OIDs to try to match devices with templates; if a suitable template cannot be found, a descriptive error page will appear. A template can include graphs from Cacti, output from Nagios, and live SNMP data. Cacinda uses the PHP package Image_Graph for creating live graphs.

Once you have Cacinda properly configured, you can create useful and aesthetically pleasing dashboards for your users and administrators to use to view host status. Dashboard pages will refresh every five minutes to display data updates. Eventually, the author of this project plans to make Cacinda a Cacti plug-in to make it easier to install and configure (Figures 5.6 and 5.7).

Figure 5.6 How Cacinda Retrieves Data from Nagios and Cacti

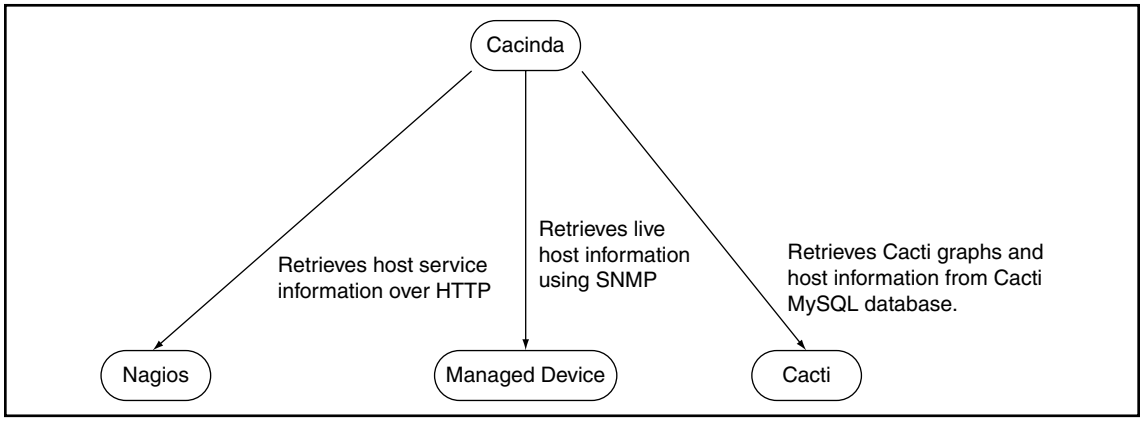
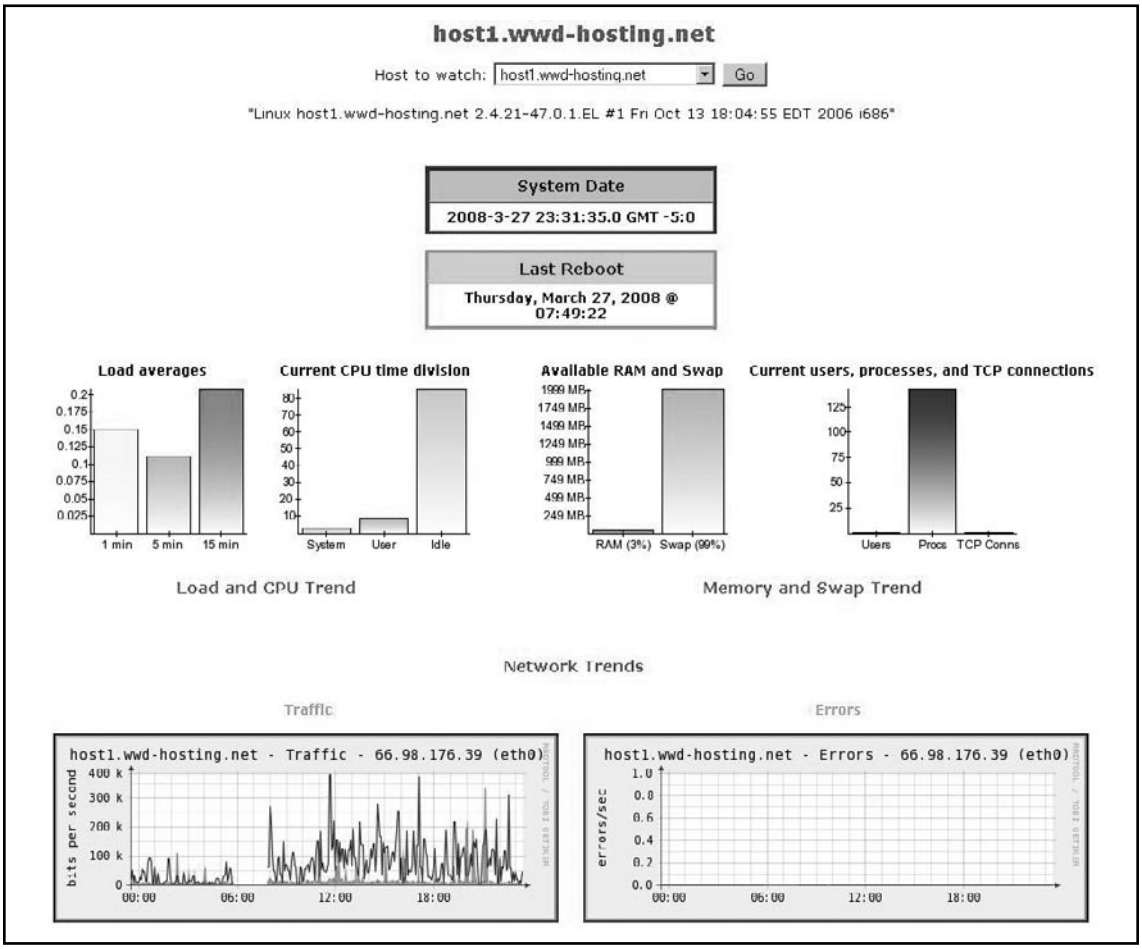


Figure 5.7 Cacinda Screenshot



NLG—Nagios Looking Glass

Download it from: http://www.nagioslookingglass.co.uk

Nagios Looking Glass (NLG) is a PHP-based project that allows you to set up a read-only status site that sits outside your Nagios security zone (DMZ for example). The software consists of two pieces: a client that resides on the system end users will access to view Nagios host and service status; and a server piece that sits on the Nagios server (under the *share* directory) that communicates with the client to retrieve Nagios status files. We tried version 1.10 b1 for the purposes of this book, as it supports Nagios 3.

The software requirements for the system are fairly standard; it does require PHP 5.1 or newer for both the client and server sides (we used php 5.2.5). The project

allows the administrator to customize the look, feel, and output of the NLG site. Even the thresholds for the front-end screen network and metric health bars can be customized. Most users will find the default look and feel aesthetically pleasing. NLG provides views that non-technical staff and managers will find especially pleasing, although technical staff will find much of the same detailed information available in the NLG Web GUI the core Nagios GUI produces (Figures 5.8 and 5.9).

Figure 5.8 Nagios Looking Glass Data Flow

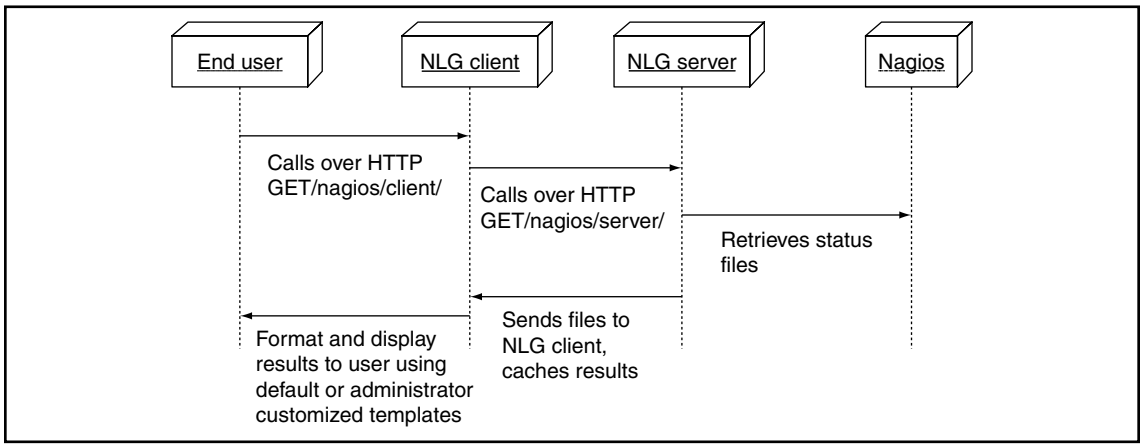
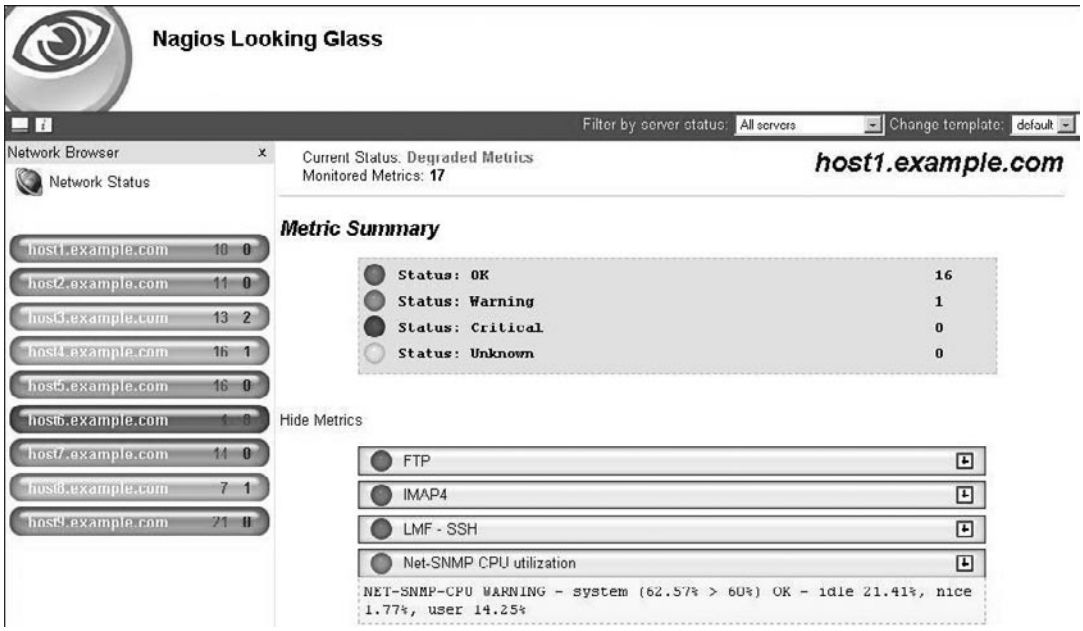


Figure 5.9 Nagios Looking Glass in Action



SNMP Trap Handling

Many open source monitoring packages lack the capability to ingest and process SNMP traps. While Nagios does not come "out of the box" with this capability, enabling it is fairly easy due to the open nature of Nagios. While configuring SNMP agents to send traps is sometimes a bit complex, the benefits of being able to receive traps are that events are received much closer to when they occur compared to waiting for polling to happen and those events take fewer system resources to process than do polled events. In this section, we give an overview of SNMPTT, a trap-handling program that hooks into Net-SNMP's snmptrapd. We then discuss how to make use of NagTrap, an open source trap viewer for Nagios.

Net-SNMP and snmptrapd

Download it from: http://net-snmp.sourceforge.net/

If you use SNMP with Nagios and are not familiar with Net-SNMP, you either are not managing Unix and Unix-like systems or you are fortunate enough to have a budget that allows you to use a commercial agent. Net-SNMP is an open source, extensible SNMP agent and suite of SNMP utilities. The only commercial agent we have used that exceeds its functionality is the CA (formerly Empire Technologies) SysEdge agent, which retails for over $3k per instance. Net-SNMP runs on a wide variety of Unix and Unix-like platforms and Windows. snmptrapd is a trap listener included with the Net-SNMP distribution; it will receive traps and then has a very flexible configuration that allows you to specify hooks to handle traps based on source IP, authentication, and/or OID. SNMPTT relies on snmptrapd, so make sure you download and install snmptrapd before installing SNMPTT.

SNMPTT

Download it from: http://www.snmptt.org/

SNMPTT (SNMP Trap Translator) is a Perl-based program developed by the Net-SNMP group. It translates traps from SNMP to more human-readable formats (or other computer-parsable formats). It can also populate databases or files with the traps. NagTrap reads SNMPTT translated traps from a MySQL database. Download and install SNMPTT and follow the directions to enable it to store traps in MySQL before installing NagTrap.

Configuring SNMPTT for Maintainability and Configuration File Growth

SNMPTT allows an administrator to specify multiple configuration files for event definitions; take advantage of this feature to divide your configuration into files by device type or event type. For example, if we have events coming in from Oracle Enterprise Manager (OEM), Digi devices, and Bluecoat proxies, we might use the directory /usr/local/snmp/etc/snmptt/ to place configuration snippets for each device type in. We tell SNMPTT to look at multiple files by adding them to the snmptt_conf_files variable in the SNMPTT main configuration file; since SNMPTT is written in Perl, it uses an INI parser that understands Perl HERE document syntax. For example:

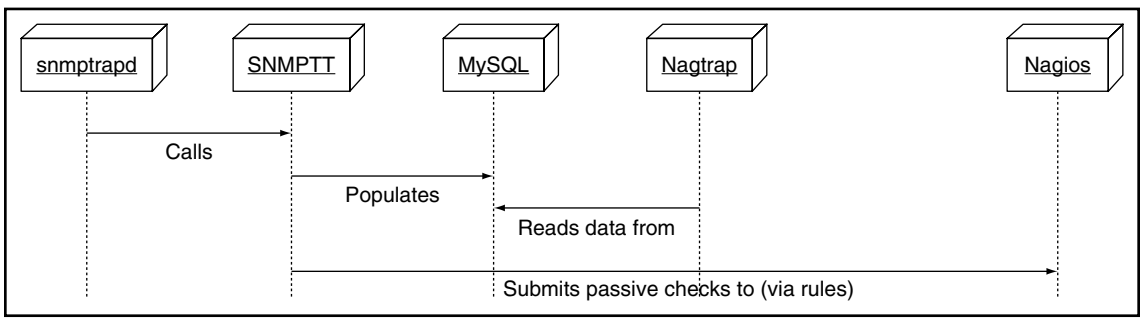
```
snmptt_conf_files = <<EOF
/usr/local/snmp/etc/snmptt/oem.conf
/usr/local/snmp/etc/snmptt/digi.conf
/usr/local/snmp/etc/snmptt/bluecoat.conf
EOF
```

NagTrap

Download it from: http://www.nagtrap.org/

NagTrap is an open-source Nagios add-on that reads SNMP traps from a database created by SNMPTT (SNMP Trap Translator). It allows you to view, query, and filter traps by host name, severity, and category from a PHP-based Web interface. It also allows you to archive traps through the GUI. Finally, it includes a script to help you get traps from SNMPTT into Nagios (Figure 5.10).

Figure 5.10 NagTrap Data Flow



Download NagTrap from http://www.nagtrap.org. Installation is straightforward and well documented. Two changes we recommend making are:

1. Change the format line field in the snmptt database from type varchar(255) to text(2000) as a number of traps will exceed the varchar(255) limit.
2. Add indexes to the SNMPTT table; we recommend adding indexes to the category and severity fields at a minimum.

While SNMPTT comes with a script that can be run as a check from Nagios that polls the SNMPTT database for new traps, we believe you will get better performance by adding script triggers to your SNMPTT configuration files that submit passive checks to Nagios as this keeps Nagios from having to poll yet another data source. We discuss how to do this and show an example script that will allow you to do this in the next section of this chapter. Ingesting Traps into Nagios.

SNMPTT allows you to specify a custom script to be called when a match for an OID is found in a configuration file you create. This hook makes it easy for us to then submit alerts to Nagios as passive checks. Since most devices we deal with will likely send dozens if not more events to your SNMPTT instance, we find it easiest to create a passive check per device type and then define the severity of the event in the SNMPTT configuration line. For example, if we have a Raritan configured to send Nagios SNMPT traps we might create a "Raritan Event" passive check that can be used to accept any of the 30 or so SNMP traps the Raritan can send.

The hardest part of ingesting traps from SNMPTT into Nagios is mapping the device names and IPs to your Nagios names and IPs. Since some devices are multihomed, you might receive a trap from a device with a source IP address in the trap that does not match the IP you have configured for the device in Nagios. There are several ways to work around this situation. First, you can change your Nagios configuration for the device in question so the IP used for traps matches the IP you use for polling. Second, you can make sure every device you receive events from has a DNS PTR (reverse IP to name) record in DNS, and turn on DNS in SNMPTT. Third, and most flexible, you can just add a local database table or flat file where you can map these additional outside IPs to your inside IPs and write your script so it looks up the IP in this local cache before it tries to match the IP address in your Nagios to IP address data store.

There are two ways to query Nagios to get the official hostname for a device for which you have an IP address. The first is to write a wrapper script that will let you query your Nagios configurations, or a script that parses your Nagios host configurations and writes the IP to host name mappings to either a flat file or a database. This method places custom development effort on you but then means you have no extra components to add to Nagios to retrieve hostname or IP address information. The second way is to install NDO (mentioned earlier in this chapter)

and then just query the Nagios event database using your favorite scripting language's database abstraction API.

For this SNMPTT to Nagios script, we chose to use NDO because we had it installed already for NagVis (mentioned earlier in this chapter). This made the SNMPTT to Nagios script very short and easy to write.

Here is an example of what an event configuration in SNMPTT looks like, including calls to our custom script:

```
EVENT digiLoginSuccess .1.3.6.1.4.1.332.10.14.14.0.1 "Digi" warning
EXEC /usr/local/etc/snmp/snmptt2nagios.pl $ar $s "Digi Event" "$Fz"
FORMAT Successful login: $1

EVENT digiLoginFailed .1.3.6.1.4.1.332.10.14.14.0.2 "Digi" critical
EXEC /usr/local/etc/snmp/snmptt2nagios.pl $ar $s "Digi Event" "$Fz"
FORMAT Failed login: $1

EVENT digiGeneric .1.3.6.1.4.1.332.* "Digi" ok
EXEC /usr/local/etc/snmp/snmptt2nagios.pl $ar $s "Digi Event" "$Fz"
FORMAT $+*
```

Here is what a passive check configuration in Nagios that receives events from this script looks like; note that we set up a "reset" freshness check that will be called to reset the state of the event to OK if a new event is not received within a period of time.

First, we define a base service template for the passive service:

```
define service {
  use                           generic-service
  name                          passive-base
  check_period                  24×7
  flap_detection_enabled        0
  max_check_attempts            1
  active_checks_enabled         0
  passive_checks_enabled        1
  normal_check_interval         1
  retry_check_interval          1
  check_freshness               1
  freshness_threshold           3600
  process_perf_data             1
  contact_groups                admins
  notifications_enabled         0
  register                      0

}
```

Then, we define a template for the Digi event service:

```
define service {
  use                           passive-base
  check_command                 check_digi_freshness
```

```
  hostgroup_name              digi-hosts
  service_description         Digi Event
  contact_groups              admins
}
```

Finally, the command definition for the command that will reset the service to OK after one hour (3600 seconds) without a new event arriving:

```
define command {
  command_name check_digi_freshness
  command_line $USER1$/check_dummy 0
}
```

Here is the script we can use to submit a passive check from SNMPTT to Nagios; we use the NDO database to map agent IP addresses to Nagios host names:

```
#!/usr/local/bin/perl

use strict;
use DBI;
$|++;

my $DSN = "DBI:mysql:database=nagios;host=localhost";
my $DB_USER = 'user';
my $DB_PASS = 'pass';
my $CMD_FILE = '/var/nagios/rw/nagios.cmd';

my $address = $ARGV[0] || usage("Missing address to look for!");
my $level = $ARGV[1] || die usage("Missing condition level");
my $descr = $ARGV[2] || die usage("Missing service description");
my $output = $ARGV[3] || die usage("Missing plugin output");

my %ERRORS = qw(

   OK        0
   WARNING   1
   CRITICAL  2

);

$level = uc($level);

usage("Invalid level $level") unless exists $ERRORS{$level};

my $dbh = DBI->connect($DSN, $DB_USER, $DB_PASS);

my $sql = <<EOF;
SELECT display_name from nagios_hosts where address = '$address'
EOF

my $sth = $dbh->prepare($sql);
$sth->execute;
my $row = $sth->fetchrow_hashref();
```

```
my $host = $row->{'display_name'};

usage("Host not found!") unless $host;

my $time = time();
my $return = $ERRORS{$level};
# [<timestamp>]
PROCESS_SERVICE_CHECK_RESULT;<host_name>;<svc_description>;
<return_code>;<plugin_output>

my $check = "[$time] PROCESS_SERVICE_CHECK_RESULT;$host;$descr;$return;$output";

open(my $fd, "> $CMD_FILE") ||
  die "Cannot write to Nagios external command file $CMD_FILE: $!";
  print $fd "$check\n";
  close($fd);
    exit;

sub usage {

my $msg = shift;

die <<EOF;

$msg
$0 IP OKÐWARNINGÐCRITICAL "Service Description" "Plugin output"

e.g.

$0 192.168.1.30 WARNING "Digi" "User foo logged into Digi1 from 192.168.0.33

EOF

}
```

Translating SNMP trap MIBS to SNMPTT format is an easy process, and receiving events from all of our specialized devices and monitoring programs (like Oracle's OEM) has proven very useful.

Text-to-Speech for Nagios Alerts

Often, data center or technical managers or NOC staff think that having the alerting system speak alerts will make their lives much easier. Before long, however, you will see them shrink down in their seat as alerts are spoken or just quietly turn the volume down on the speakers of the alerting station. Spoken alerts must be implemented judiciously. Only have the text-to-speech system "say" the most critical alerts.

Something else to keep in mind when implementing text-to-speech is that even the better text-to-speech libraries will not pronounce words the way we might expect them to. Therefore, it will most likely be necessary to include in any text-to-speech scripts the capability to add custom word and phrase transformations that change words that trip up voice libraries into phonetic spellings that make them sound truer to their original spellings. For an example text-to-speech daemon that polls Nagios and "speaks" alerts, see Chapter 2.

Summary

As you can see, a wide variety of add-ons and enhancements let Nagios do much more than just monitor networks and systems by running plug-ins. A major strength of Nagios (and a feature that can be intimidating to new users) is its flexibility and openness. We have covered just a small portion of the add-ons available for Nagios in this chapter; we hope that the add-ons we discussed will get you excited about the Nagios community and the power that Nagios brings to administrators and users.

This page intentionally left blank