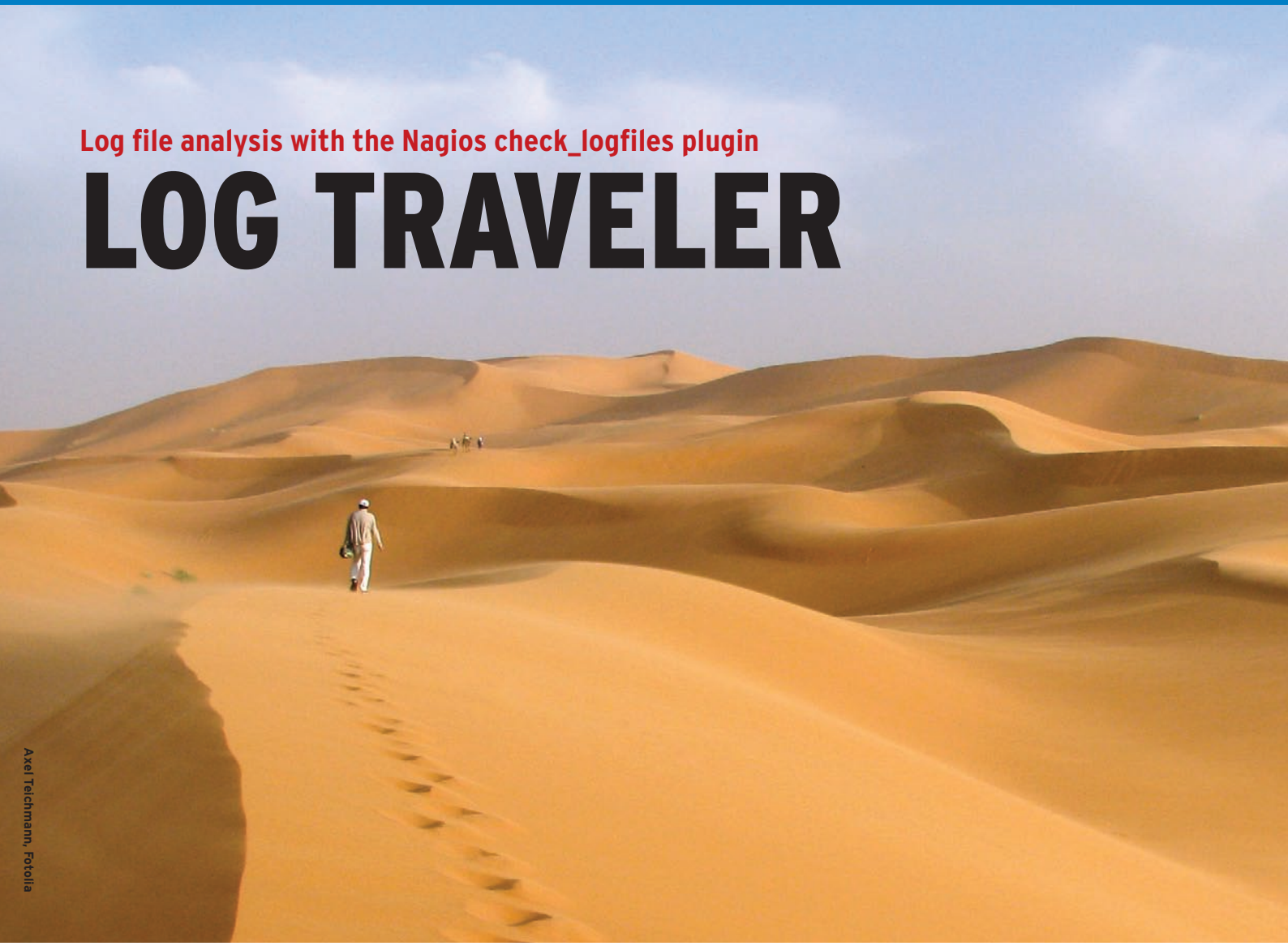

Axel Teichmann, Fotolia

Log file analysis with the Nagios check_logfiles plugin

LOG TRAVELER

The Nagios check_logfiles plugin helps you monitor your logfiles – even if the logs rotate and change names.

BY GERHARD LAUSSER

The Nagios monitoring tool is a general framework for watching things. Nagios lets you keep an eye on computers, processes, devices, and network services. Another thing Nagios can watch is logfiles. The Nagios plugin collection comes with a number of options for monitoring logs. The *check_log* and *check_log2* plugins, for example, are popular with many admins; however, these plugins sometimes have problems in situations in which an application or script is rotating the logs. The tools tend to slip up occasionally and miss a couple of lines, which is something you can't allow if you need 100% coverage. To close the gaps, the *check_logfiles* plugin [1] was developed to check every single entry – even if a log moves, changes its name, or disappears into a compressed archive during the monitoring period.

But that's not all: A number of other sophisticated features set *check_logfiles* apart from its predecessors. For example, *check_logfiles* can work with multiple search keys, handle exceptions that identify a special subset of a search key as harmless, apply thresholds that trigger alerts after a minimum number of matches, and integrate external programs.

In this article, I show you how to start monitoring logfiles with the Nagios *check_logfiles* plugin. To begin, I'll assume you have some basic knowledge of Nagios. If you're looking for more background on the Nagios monitoring tool, see the Nagios workshop from the June 2007 issue of *Linux Magazine* [2].

Installation

The *check_logfiles* plugin is available as a tarball [1]. After unpacking, change to directory *check_logfiles-2.3.1.1* and follow the standard *configure; make; make install* steps to build and install the plugin.Various options are available for the *configure* step (see the box titled "Configure Options").

First Case

Once you have installed and configured the *check_logfiles* plugin, you're ready to put it to work monitoring logfiles.

For a first look at *check_logfiles* in action, consider the following example, which performs a simple search for the *BIGERROR* string in a file titled *rhubarbomat.log*. The call to the plugin looks like this:

```
check_logfiles ➥
-criticalpattern=➥
'BIGERROR'-logfile=➥
rhubarbomat.log
```

Configure Options

--with-perl if you prefer a separate Perl installation.

--prefix specifies the home directory of your Nagios installation. The plugin is installed in the */libexec* subdirectory.

--with-seekfiles-dir specifies the directory in which to save status information between program runs.

If the string *BIGERROR* occurs in a line that was added after the last *check_logfiles* run, the plugin returns a *CRITICAL* status; if not, it returns *OK*. The string is actually a regular expression.

Instead of *-criticalpattern*, you could use *-warningpattern*, as in *--warningpattern = 'SMALLERROR'*. The exit code for a successful search is *1* for *WARNING*. Of course, nothing stops you you from using both options at the same time.

This initial example does not take logfile rotation into account. Although the plugin would identify the search key, it would not search logfiles that had been rotated out, but would, instead, simply focus on the latest file.

To allow the search to cover logfiles rotated between two calls to *check_logfiles*, and thus to avoid gaps, the plugin needs a hint as to where to find the older files.

The parameter that handles this is *-rotation*, which either passes in the new file name or contains a regular expression that matches the rotated file names (Figure 1). First assume the *rhubarbomat.log* file is automatically renamed *rhubarbomat.log.0* on a daily basis and that an empty *rhubarbomat.log* file is created. After this, what used to be *rhubarbomat.log.0* is renamed *rhubarbomat.log.1*, what used to be *rhubarbomat.log.1* is renamed *rhubarbomat.log.2*, and so on. In this case, the *-logfile = /var/log/rhubarbomat.log* *-rotation = 'rhubarbomat\.log\.\d+'* parameter would let the plugin find and search the previous versions. As an alternative, you could explicitly specify the file name *rhubarbomat.log.0*.

The *check_logfiles* plugin only investigates lines in the logfile that have changed since the plugin was last called, and this means that re-running the plugin will return different results. After returning a *CRITICAL* result, the next call is to revert *OK*, as Listing 1 shows. A service definition for Nagios is given in Listing 2.

Configuration File

Where *check_logfiles* really shines is when you use a configuration file instead of command-line parameters. The previous example would require the following configuration file:

```
$ cat rhubarb.cfg
@searches = ({
 tag => '0815',
 logfiles => ➥
'/var/log/rhubarbomat.log',
 criticalpatterns => '.*0815.*',
 rotation => 'loglogOlog1',
 options => 'noprotocol'
});
```

The plugin is then called by a *check_logfiles -f configfile_name* command line. It is easy to see that the configuration file

Listing 1: Recurring Calls

```
01 $ logger "test1 this is 0815"
02 $ logger "this isn't because its 0916"
03 $
04 $ check_logfiles -logfile=/var/log/
  rhubarbomat.log --tag=0815 -criticalpattern=
  '.*0815.*' --rotation='loglogOlog1'
05
06 CRITICAL - (1 errors in check_logfiles.
  protocol-2007-10-10-15-10-02) - Oct 10 15:09:56
  localhost lausser: test1 das ist doch 0815 |0815_
  lines=2
07 0815_warnings=0 0815_criticals=1
  0815_unknowns=0
08 $
09 $ echo $?
10 2
11 $
12 $ logger "rhubarb"
13 $ check_logfiles -logfile=/var/log/
  rhubarbomat.log --tag=0815 -criticalpattern=
  '.*0815.*' --rotation='loglogOlog1'
14
15 OK - no errors or warnings |0815_lines=1
  0815_warnings=0 0815_criticals=0
16 0815_unknowns=0
17 $ echo $?
18 0
```

Listing 2: Service Definition

```
01 define service {
02  service_description check_0815msgs
03  host_name logserver
04  max_check_attempts 1
05  is_volatile 1
06  check_command
07  check_logfiles_critical!0815!/var/log/
   rhubarbomat.log!loglogOlog1!.*0815.*
08 }
09 define command {
10  command_name check_logfiles_critical
11  command_line $USER1$/check_logfiles $$
12  --logfile="$ARG2$"
13  --criticalpattern="$ARG4$" --tag="$ARG1$" $$
14  --rotation="$ARG3$"
15 }
```

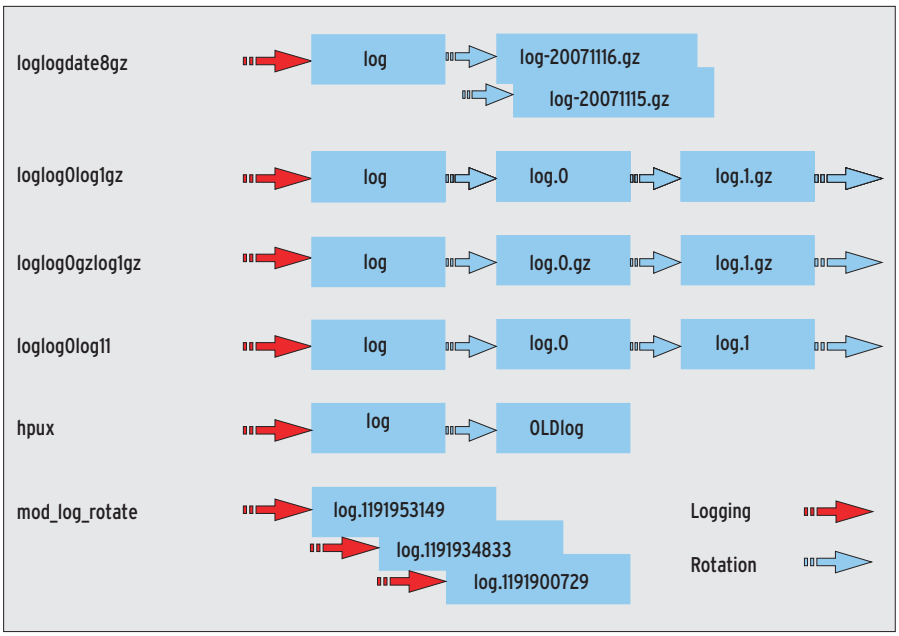

Figure 1: Logfile rotation makes life difficult for some plugins. Entries can get lost because of copying or renaming.

is made up of Perl code. The elements in the *@searches* array (which I will just refer to as "the search") are hash references that combine the logfile and search key. The tag is a unique identifier for the combination. The plugin requires this to disambiguate the files that store the status information for the next *check_logfiles* run. An array makes it possible to search multiple logfiles with a single call to *check_logfiles*.

The Perl code for this configuration is shown in Listing 3.

On the other hand, if you want the plugin to raise the alarm if a search key is missing from the logfile, the search pattern must start with an exclamation mark (*!*). The following syntax tells you whether the late-night backup has completed without errors:

```
criticalpatterns => ➥
['!backup successful']
```

Also, you can define exceptions that look like the message you are searching for but represent a special case:

```
criticalpatterns => ➥
['SCSI Error'],
criticalexceptions => ➥
['SCSI Error. *disk0 .*'],
```

These entries would raise a Nagios alarm for the *SCSI Error /dev/disk5 I/O Timeout* line, but they tell the plugin to ignore *SCSI Error /dev/disk0 I/O Timeout*.

Rotation

At the end of each run, the plugin stores the last position it has read in the logfile, along with the change date and the file's inode number. This information is stored in what is known as a seek file. *check_logfiles* generates the name of the seek file from the logfile name and the day.

The next time the plugin is called, it compares this data with the properties of he current logfile and checks that the logfile has been expanded, deleted, rotated, or created as a new file.

In case of rotation, the plugin searches for the rotated archive, which it has to read to ensure seamless monitoring coverage.

Depending on how long ago the last plugin launch was, several rotations might have occurred. The plugin uses the timestamp and the *rotation* parameter to find matching files.

In most cases, the logfile has grown by just a couple of lines. *check_logfiles* continues at the position that it tagged at the end of the last run and reads the following lines until it reaches the end of the file (Figure 2). This design gives the

Listing 3: Sample Configuration

```
01 @searches = (
02 {
03  tag => 'lamp-apache'
04  logfile => '/var/log/apache/error.log',
05  criticalpatterns => ['.*error.*,  '.*fatal.'],
06  rotation => 'solaris'
07 },
08 {
09  tag => 'lamp-mysql',
10  logfile => '/var/log/mysql.log',
11  criticalpatterns => ['corruption',
    'you hit a bug']
12 }
13 );
```

Listing 4: Searching virtual Logfile Types

```
01 @searches = (
02  {
03   tag => 'host0',
04   logfile => '/sys/class/scsi_host/host0/state',
05   type => "virtual",
06   criticalpatterns => [
07    'Link [^Up]+' # Alarm, wenn nicht "Link Up"
     drinsteht
08   ],
09   options => 'noprotocol',
10  },
11 );
```

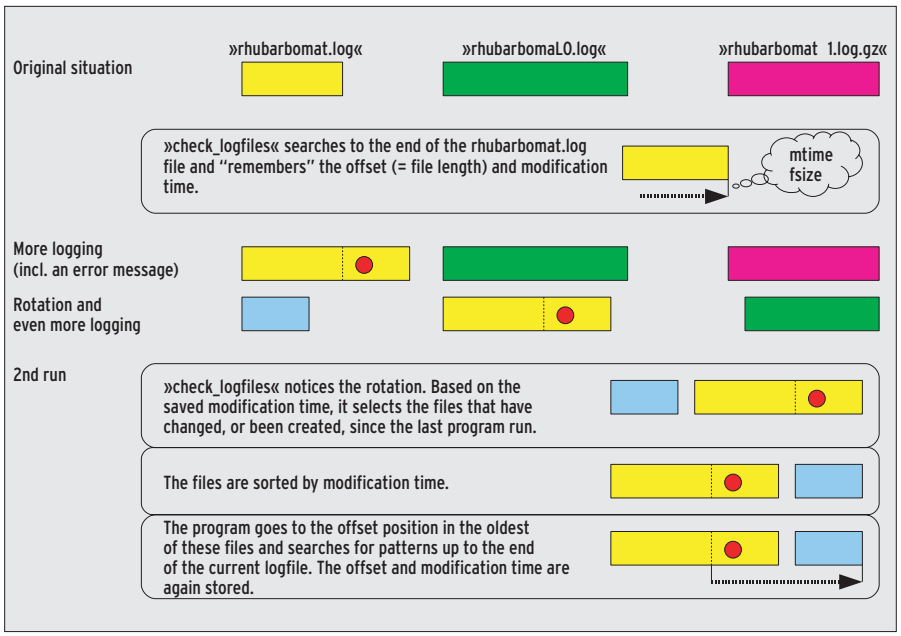

Figure 2: The Log Checker remembers the file and the position it read to when last called and continues at exactly that position.

plugin an enormous speed advantage compared with other approaches that rely on *diff* to ascertain the differences between the current logfile and a stored copy, especially in the case of rapid logfile growth.

By entering *./configure with-seekfile-dir*, you can specify a directory for the seek files, or you can change the path on the fly with the *$seekfilesdir* variable in the configuration file.

The plugin uses */tmp* by default; however, it makes sense to change this to */var/tmp* because some operating systems do not keep the content of */tmp* on reboot.

Types

In addition to the *rotation* parameter, you can also search on the *type* parameter, which specifies the type of logfile. If the rotation parameter exists, *type* assumes a value of *rotation*. This means that the archive files are relevant to the search. If the rotation parameter does not exist, *type* assumes a value of *simple*. The setting makes sense if an application continually generates new logfiles and deletes the existing files or if the administrator is prepared to accept the fact that the last few lines in a rotating logfile will not be taken into consideration.

The *virtual* logfile is another type that *check_logfiles* will search. This type is used, say, for the */proc* filesystem on Linux machines (and this gives you the option of setting up hardware monitoring simply).

The files in this filesystem do not grow; instead, you need to treat them as if they had been created immediately before reading. The plugin always investigates these logfiles from the first line down (see Listing 4).

Also, the *errpt* type searches the AIX Error Report. This tells the plugin to search for patterns in the output from the *errpt* command, just as if it were a normal logfile. The *psloglist* type is still experimental; it lets the plugin search the event log on a Windows machine.

Search Parameters

Several parameters are available for patterns to search the logfiles. The most important parameters are listed in the

Listing 5: Parameters in the Configuration File

```
01 $ cat rhubarb.cfg
02 @searches = ({
03  tag => '0815',
04  logfile => '/var/log/rhubarbomat.log',
05  archivedir => '/var/log/archives',
06  rotation => 'loglog0gzlog1gz',
07  criticalpatterns => '.*0815.*',
08  criticalexceptions => '.*0815 macht aber nix.*',
09  warningpatterns => ['.*failure.*',
    '!successful'],
10  warningthreshold => 10,
11  okpatterns => '.*cleared.*',
12  options => 'case,noprotocol,script'
13  script => 'restart_rhubarbomat'
14 });
```

Listing 6: Searching Multiple Patterns

```
01 @searches = (
02  {
03   tag => 'minor_errors',
04   type => 'errpt',
05   criticalpatterns => ['ADAPTER ERROR',
06     'The largest dump device is too small.',
07     'The copy directory is too small.',
08     'Kernel heap use exceeds allocation count',
09     'Kernel heap use exceeds percentage thres',
10     'LINK ERROR',
11     'SCSI BUS OR DEVICE ERROR',
12     'SCSI DEVICE OR MEDIA ERROR',
13     'Possible malfunction on local adapter',
14     'ETHERNET DOWN',
15     'UNABLE TO ALLOCATE SPACE IN KERNEL HEAP'
16   ],
17  }
18 );
```

Search Parameters

- *tag* A short unique descriptor for this search.
- *logfile* The name of the logfile you want to scan.
- *archivedir* The directory with the rotated logfiles.
- *rotation* A regular expression that is used to locate rotated archive files. Predefined values exist for the most common patterns.
- *criticalpatterns* A single pattern that the plugin searches for in the logfile. If you want the plugin to search for multiple patterns belonging to a category, you need to specify them as the elements of an array (e.g., see Listing 6).
- *criticalexceptions* Support more granular specification of patterns: The parameter ignores exceptions that are not counted as errors.
- *warningthreshold* Thresholds are used whenever you want to count a certain number of matches before alerting: *warningthreshold => n* means that every *n*th match counts.
- *okpatterns* Resets the counter and deletes all previously found critical and warning matches.
- *nologfilenocry* Ignores missing logfiles; otherwise, if the logfile is missing, the plugin returns a status of *UNKNOWN*.
- *syslogserver* If the logfile contains messages from multiple servers, the plugin uses this option to search only the messages from the local host.
- *syslogclient=host_name* Just like the server option; however, in this case, only messages from a specific client are investigated. This option is interesting for central Syslog servers.
- *nocase* Ignores case in regular expressions.
- *options* Multiple, comma-separated options give more granular control over the plugin's actions. A prefix of *no* reverses the meaning.
- *- nocase* Means that the patterns are case insensitive.
- *- noprotocol* Prevents the plugin from creating a protocol file. Normally, any lines in the logfile that contain the search pattern are written to the protocol file. This saves time-consuming processing in the case of an alert.

"Search Parameters" box. A complete list of all possible parameters can be found online [1].

The parameters are used in the configuration file as shown in the excerpt in Listing 5.

Output and Performance Data

The output from *check_logfiles* contains references to the findings, for example:

```
CRITICAL - (3 errors in ➥
check_logfiles.protocol-2007➥
-10-10-16-21-09) InnoDB: ➥
Database page corruption on ➥
disk or a failed ...|➥
mysql_lines=12 ➥
mysql_warnings=0➥
mysql_criticals=3 ➥
mysql_unknowns=0
```

Besides the typical Nagios exit code, you can see the three dots (...), which indicate that more lines with matches exist.

For each search or day, the plugin also returns a set of four performance statistics:

- *_lines* – The number of lines searched in the logfile.
- *_warnings* – The number of lines that contain warning patterns.
- *_criticals* – The number of lines that contain critical patterns.
- *_unknowns* – The number of lines that contain unknown patterns.

These parameters give you a quick indication of the problem density.

Actions

The *script* option can run a program in case of a specific match:

```
script => 'name_of_program'
```

or in the latest version:

```
script => sub { perl-code }
```

Listing 7: Calling Scripts

```
01 $scriptpath = '/usr/bin/nagios/libexec:
   /usr/local/nagios/contrib';
02 $MACROS = {
03   CL_NSCA_HOST_ADDRESS => "lpmon1.muc",
04   CL_NSCA_PORT => 5778
05 };
06
07 @searches =(
08 {
09  tag => 'rhubarb',
10  logfile => '/var/log/rhubarbomat.log',
11  criticalpatterns => ['ERROR', 'crashed'],
12  script => 'restart_rhubarbomat',
13  scriptparams => '--rhubarbprefix=bla',
14  options => 'script'
15 },
16 {
17  tag => 'san',
18  logfile => '/var/adm/messages',
19  criticalpatterns => [
20    'Link Down Event received',
21    'Loop OFFLINE',
22    'fctl:.*disappeared from fabric',
23    '.*Lun.*disappeared.*'
24  ],
25  options => 'script',
26  script => 'send_nsca',
27  scriptparams => '-H $CL_NSCA_HOST_ADDRESS$
    -p $CL_NSCA_PORT$ -to $CL_NSCA_TO_SEC$
    -c $CL_NSCA_CONFIG_FILE$',
28  scriptstdin => '$CL_HOSTNAME$\t$CL_SERVICEDESC$\
   t$CL_SERVICESTATEID$\t$CL_SERVICEOUTPUT$\n',
29 });
```

Global Parameters

Besides parameters that relate to a single search entry, additional global variables are read by all searches, defining the behavior of the plugin independent of an individual search.

- *$seekfilesdir* Specifies the directory where files with status information are saved.
- *$scriptpath* A list of paths the plugin searches for external scripts, which it triggers with the *script* parameter.
- *$prescript* Specifies an external program to be executed during startup.
- *$postscript* Specifies an external program to be executed before termination.
- *$protocolsdir* A directory in which *check_logfiles* writes protocol files with the matched lines.

Actions include restarting an application or sending SNMP traps and NSCA messages. This means that *check_logfiles* can run as a standalone application without relying on the Nagios event handler.

The *scriptparams* and *scriptstdin* parameters allow users to run external scripts with command-line parameters – and even to pass in input from STDIN. Listing 7 gives an example.

In the example in Listing 7, whenever an error message appears in a line of the *messages* file, the line is sent to the Nagios server with the *send_nsca* command as a passive service result.

In the simplest case, the exit code returned by the external script will be irrelevant and will not influence the *check_logfiles* exit code. For example, even if the Rhubarbomat application in the example in Listing 5 restarts successfully, *check_logfiles* will still return a *Critical* status to Nagios.

The *smartscript* option passes the external script's exit code in to the *check_logfiles* result. The plugin acts as if it had discovered another line after the triggering line, which had the text that was the first line in the script output and the analysis that was the script's exit code. This lets you throw an error, but not to revert the original message in the logfile or to reevaluate the message.

The third option is *supersmartscript*. Scripts of this type overwrite the triggering match in the logfile with their exit codes and output, instead of adding an entry. Several environmental variables are available for these scripts:

- *CHECK_LOGFILES_SERVICEOUTPUT* – the content of the triggering line
- *CHECK_LOGFILES_SERVICESTATE* – WARNING, CRITICAL, or UNKNOWN
- *CHECK_LOGFILES_SERVICESTATEID* – 1, 2, or 3

With the use of this information and other data – such as the time of day or the results of the application relaunch – the error message can then be reevaluated. This lets the logfile checker demote a *CRITICAL* status to *WARNING* or return an exit code of *0* and thus cancel the alert. Listing 8 gives an example.

Prescripts and Postscripts

Actions can also be triggered before starting to search a logfile or after completing all searches.

The parameter *$prescript*, which points to an external script or Perl subroutine, helps with triggering actions. Supersmart prescripts cancel the *check_logfiles* run if the exit code is greater than zero. This makes it possible to check to see whether a specific process is running.

If the process is not running, why bother checking the corresponding application logfile for errors? Prescripts can also force applications to write (flush) their logfiles, thus making sure that the data is up to date.

Supersmart postscripts can replace the *check_logfiles* results completely, no matter how many error messages they originally contained.

Or, if the standard *check_logfiles* output format is not to your liking, you could run a supersmart postscript to modify it for better emphasis. ■

INFO

[1] *check_logfiles*: *http://www.consol.com/opensource/nagios/check-logfiles/*

[2] "Network Monitoring: Watching your Systems with Nagios" by Julian Hein, *Linux Magazine*, June 2007: *http://www.linux-magazine.com/issues/2007/79/nagios*

THE AUTHOR

Gerhard Laußer works for ConSol in Munich, Germany. He installed his first version of Linux back in 1992 from a pile of floppy disks. In 2003, Gerhard managed the Linux roll-out for a major automobile manufacturing company, where he has since set up a substantial Nagios installation.

Listing 8: Supersmart Script

```
01 @searches =(
02 {
03  tag => 'rhubarb',
04  logfile => '/var/log/rhubarbomat.log',
05  criticalpatterns => ['ERROR', 'crashed'],
06  script => sub {
07   if (`restart_rhubarbomat` =~ /successful/) {
08    if ($ENV{CHECK_LOGFILES_SERVICEOUTPUT} =~ /
  ERROR/) {
09     printf "OK - restarted rhubarbomat\n";
10     return 0;
11    } else {
12     printf "WARNING - restarted crashed
  rhubarbomat\n";
13     return 1;
14    }
15   } else {
16    printf "CRITICAL - could not restart
  rhubarbomat\n";
17    return 2;
18   }
19  },
20  options => 'supersmartscript'
21 },
```