

---

# serverM Intrusion Detection System Version 2.2

(Unix/Linux/BSD/Mac OS X)

© 2006 David Scholefield <http://www.port80.com>

---

:80

# Table Of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>10</b>
1.1	OVERVIEW .....	11
1.2	MAIN FEATURES .....	12
1.3	MINIMUM REQUIREMENTS.....	15
<b>2</b>	<b>INSTALLATION AND LICENSING.....</b>	<b>16</b>
2.1	INTRODUCTION .....	17
2.2	INSTALLING THE SERVERM SYSTEM .....	18
2.3	THE INTERACTIVE CONFIGURATION SCRIPT .....	19
2.4	SETTING THE PERL5LIB ENVIRONMENT VARIABLE .....	20
2.5	STARTING AND STOPPING THE SERVERM SYSTEM .....	21
2.6	USERS AND GROUPS .....	22
2.7	UNINSTALLING THE SERVERM SYSTEM .....	22
<b>3</b>	<b>BASIC CONCEPTS.....</b>	<b>24</b>
3.1	THE MASTER CONFIGURATION FILE.....	25
3.2	THE INTRUSION DETECTION ENGINE .....	26
3.3	RULES, ALARMS, AND REPORTS.....	28
3.4	DAILY REPORTS.....	30
3.5	LOGS.....	32
3.5.1	<i>The Local Log File</i> .....	32
3.5.2	<i>Managing the Log File</i> .....	33
3.5.3	<i>Syslog</i> .....	33
3.5.4	<i>Remote Logging</i> .....	33
<b>4</b>	<b>CONFIGURING THE SERVERM ENGINE .....</b>	<b>34</b>
4.1	INTRODUCTION .....	35
4.2	PARAMETERS .....	36
4.2.1	<i>status</i> .....	36
4.2.2	<i>daily-report</i> .....	36
4.2.3	<i>daily-report-time</i> .....	36
4.2.4	<i>daily-report-email[1,2,3]</i> .....	36
4.2.5	<i>email-alarms</i> .....	37
4.2.6	<i>alarm-email-throttle</i> .....	37
4.2.7	<i>alarm-sms</i> .....	37
4.2.8	<i>alarm-sms-throttle</i> .....	37
4.2.9	<i>sms-gateway</i> .....	38
4.2.10	<i>email-gateway</i> .....	38
4.2.11	<i>period</i> .....	38
4.2.12	<i>heartbeat</i> .....	39
4.2.13	<i>administrator</i> .....	39
4.2.14	<i>smtp-reply</i> .....	39
4.2.15	<i>sms-to</i> .....	40
4.2.16	<i>clear-on-pause</i> .....	40
4.2.17	<i>alarm-format</i> .....	40
4.2.18	<i>report-format</i> .....	40
4.2.19	<i>syslog-level</i> .....	41
4.2.20	<i>syslog-include</i> .....	41
4.2.21	<i>syslog-severity</i> .....	41
4.2.22	<i>log-location</i> .....	42
4.2.23	<i>rlog</i> .....	42
4.2.24	<i>rlog-severity</i> .....	42
4.2.25	<i>rlog-include</i> .....	43
4.2.26	<i>user-def</i> .....	43

4.3	USING THE 'HTMLREPORTS' TOOL.....	44
<b>5</b>	<b>DESIGNING AND IMPLEMENTING RULES .....</b>	<b>45</b>
5.1	INTRODUCTION – THE RULES FILE .....	46
5.2	DEFINING A SINGLE RULE .....	47
5.3	RULE EXECUTION ORDER .....	49
5.4	RULE PRAGMAS AND RULE CHECKING .....	50
5.4.1	Introduction.....	50
5.4.2	Pragmas for Controlling Rule Evaluation and for Alarm Triggering .....	50
5.4.3	Evaluation Pragmas .....	51
5.4.4	Alarm Pragmas and the 'Severity' Pragmas .....	52
5.4.5	Pragma Syntax.....	54
5.4.6	Pragma Examples.....	54
5.5	RULE SYNTAX: RULE NAMES .....	55
5.6	ALARMS .....	56
5.6.1	Overview .....	56
5.6.2	Meta Variables in Rules and Alarm Definitions.....	56
5.6.3	The Order of Alarm Execution.....	57
5.6.4	Using the 'def' Keyword .....	57
5.6.5	Pre-defining Arbitrary Strings.....	59
5.6.6	Accessing Values from the Master Configuration File in Rule Definitions.....	60
5.6.7	Execute Alarms Overhead and System Delays ('waitfor:') .....	60
5.7	RULE TYPES AND CONSTRAINTS.....	62
5.7.1	File Change Rule .....	63
5.7.2	Status Rule.....	65
5.7.3	New Service Rule .....	67
5.7.4	File Size Rule .....	69
5.7.5	True Rule.....	71
5.7.6	The 'ignore' Keyword .....	72
5.8	SCOPE AND SIGNATURES.....	73
5.8.1	Introduction and Global Scope.....	73
5.8.2	Signatures and Scoping.....	74
5.9	ALARMS AND WILDCARD FILE SPECIFICATIONS.....	77
5.10	EXAMPLE RULES .....	78
5.11	WRITING NEW RULES AND USING 'OFF THE SHELF' RULES .....	82
5.12	GOOD RULE WRITING 'STYLE' .....	83
5.12.1	Overview .....	83
5.12.2	Rule Efficiency .....	83
5.12.3	Alarm Flooding.....	84
5.12.4	Overlapping Rules.....	84
5.12.5	Over-use of True Rules with Alarms .....	85
5.13	CONDITIONAL PARSING.....	86
5.14	USING 'HTMLREPORTS' TO CHECK RULE DEFINITIONS .....	86
<b>6</b>	<b>REPORT FORMATS AND ALARM MESSAGES .....</b>	<b>88</b>
6.1	INTRODUCTION .....	89
6.2	EMAIL ALARM MESSAGES .....	90
6.2.1	Overview .....	90
6.2.2	Email Alarm Message Contents.....	90
6.2.3	Email Alarm Message Configuration.....	91
6.3	DAILY REPORT MESSAGES.....	92
6.3.1	Overview .....	92
6.3.2	Daily Report Message Contents .....	93
6.3.3	Daily Report Message Configuration.....	93
6.4	ADMINISTRATOR EMAIL MESSAGES.....	94
6.4.1	Overview .....	94
6.4.2	Administrator Email Message Contents.....	94
6.4.3	Administrator Email Message Configuration.....	95

6.5	SMS ALARM MESSAGES.....	96
6.5.1	Overview .....	96
6.5.2	SMS Alarm Message Contents.....	96
6.5.3	SMS Alarm Message Configuration .....	96
<b>7</b>	<b>THE ‘STANDARD’ RULES .....</b>	<b>97</b>
7.1	OVERVIEW .....	98
7.2	THE ‘MAIN’ RULES DEFINITION FILE.....	99
7.3	THE LOG FILE ARCHIVE RULES .....	105
<b>8</b>	<b>FREQUENTLY ASKED QUESTIONS .....</b>	<b>106</b>
8.1	INTRODUCTION .....	107
8.2	SERVERM CONFIGURATION .....	107
8.3	RULES.....	108
8.4	REPORTS AND ALARMS .....	109
8.5	MISCELLANEOUS .....	110
<b>APPENDIX A: SMS SERVICES.....</b>		<b>112</b>
<b>APPENDIX B: LOG FILE MESSAGES.....</b>		<b>114</b>
B.1	OVERVIEW .....	114
B.2	INFO MESSAGES.....	115
B.3.	ERROR MESSAGES .....	117
B.4.	STATUS MESSAGES.....	122
B.5.	ALARM MESSAGES .....	123
<b>APPENDIX C: THE ‘CHECKRULES’ TOOL .....</b>		<b>124</b>
<b>APPENDIX D: THE ‘HTMLREPORTS’ TOOL .....</b>		<b>126</b>
<b>APPENDIX E: THE ‘SCOUR’ TOOL .....</b>		<b>132</b>
<b>APPENDIX F: SERVERM REMOTE MONITOR (SRM) .....</b>		<b>135</b>
F1.	SERVERM MASTER CONFIGURATION FILE AND SRM.....	135
F2.	SRM CENTRALISED LOGGING DAEMON AND CONFIGURATION .....	135
F3.	STARTING AND STOPPING THE SRM SYSTEM.....	137
F4.	SYSTEM REQUIREMENTS.....	137
F5.	DATABASE LOGGING.....	138
F.6	THE SRM LOG FILE .....	139

## **TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

**0.** This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

**1.** You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

**2.** You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

**a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

**b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

**c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

**3.** You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

**a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

**b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange;

or,

**c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

**4.** You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

**5.** You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

**6.** Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

**7.** If, as a consequence of a court judgment or allegation of patent

infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

**8.** If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

**9.** The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free



Software Foundation.

**10.** If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## **NO WARRANTY**

**11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**12.** IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES

# 1 Introduction

---

port80's serverM intrusion detection system is a lightweight and powerful monitoring service for detecting symptoms of unauthorised computer access, and computer misuse. A flexible rules description language enables system administrators to monitor a large number of different kinds of events, ranging from unexpected file changes through to suspicious patterns within the event logs. serverM V2.2 is supported on all major Unix server platforms including most major Linux distributions, Free BSD, and Mac OS X. serverM provides the perfect early-warning system to enable individuals and organisations to react swiftly to potentially damaging intrusions.

---

## 1.1 Overview

Protecting your network and hosts against malicious Internet-borne attacks and internal server misuse can most effectively be achieved by a multi-layered approach to security monitoring and detection. A vital element of this approach is to deploy a **host based Intrusion Detection System** (HIDS) to constantly monitor, and react to, events that occur on any of the servers and workstations attached to a network.

serverM is a light-weight, highly modular and powerful host-based Intrusion Detection System designed to run on both Windows and Unix server platforms, including most major Linux distributions, Free BSD and Mac OS X. This manual refers exclusively to the Unix variant.

Using a powerful rules-based description language, the serverM intrusion detection engine is capable of monitoring a wide range of server events and conditions, which can then give rise to real-time SMS and email alarms or bespoke code execution. Syslog error and information message logging is also supported. Pre-defined rules can easily be included with an installation, or bespoke rules can be written to precisely mirror an organisation or individual's existing security policy.

In addition to detection, serverM provides a *reactive* component which enables the user to specify arbitrary code to be executed if detection rules are triggered. This benefit means that serverM can be used for **prevention and repair** as well as detection.

A typical example of serverM use is to detect an attempt to deface a public website, and not only alert an administrator of the potential threat, but to restore the website to its original state prior to any attack should the attack succeed.

Many Intrusion Detection Systems monitor the event logs, or file changes, or even services, but only serverM comprehensively monitors all of these areas, and more, in a single, integrated application.

With a detection engine optimised for low processor and memory overhead, serverM can be deployed across the enterprise network, or on a single server, to provide early warning against a wide range of malicious attacks. It can also be configured to take corrective action, as well as preserve vital information for later forensic investigation.

serverM is designed to require minimum maintenance, and with advanced features such as a library of 'off the shelf' common detection rules to complement the users own bespoke rules set, intelligent denial of service attack detection, and even self-monitoring, serverM is one of the most complete HIDS available for \*nix systems.

serverM for \*nix systems is distributed under the GNU Public License scheme, and is free to use or distribute. Please see the license agreement above for more details.

## 1.2 Main Features

### Lightweight architecture

port80 has designed the serverM system architecture to be as lightweight and flexible as possible. We believe that users of intrusion detection systems are primarily focused on the function of detection and monitoring, and that there is a requirement to achieve this goal without recourse to installing complex and resource-heavy configuration or reporting systems. The core serverM system requires only a few components: the detection engine service itself (a Unix daemon), the master configuration file, and the rules definitions file. Installation of the system does not require major system changes, complex inter-process communication, on-going maintenance, or constant system 'tweaking'. The formats of the working files that serverM uses during execution are *open format*, and system extension and development of support systems by users or third party developers is encouraged.

### Simple, powerful, rules system

serverM uses a unique rule system that enables the user to describe a wide range of events and conditions within individual detection rules, and then to specify alarms and other actions in the event of a rule being triggered. A library of pre-defined rules is available which can be used 'off the shelf', in addition to those rules designed by the user. This library is constantly being refined and updated, and port80 publish new, and tested, rules on the serverM website on a regular basis ([www.serverm.org](http://www.serverm.org)).

Unlike 'point and click' systems, which offer simple 'on/off' functionality for detecting intrusion symptoms, serverM provides a rich set of rule types and rule 'pragmas' that enable the user to specify execution conditions on the rules themselves. For example, it is possible to specify that a rule is disabled after certain conditions are met (to prevent alarm 'flooding'), or that certain rules are evaluated less often (to reduce detection overhead).

The types of events that can be monitored by serverM rules include:

- **file changes:** any changes to nominated file content or access modes, with the ability to recursively check entire directory sub-trees or individual files (MD5 hashes and ACLs are used);
- **additional users and user account changes:** alarms can be triggered if new users are added to the system, user groups or rights are changed, or users are removed from the system;
- **service status:** check for new processes starting or stopping, check whether a specified process is running, along with the ability to 'ignore' specified processes;
- **file sizes:** check for specified files growing over a pre-defined limit, including individual files or all files within a directory sub-tree;

- **command status:** output from user specified commands can be compared over time to detect changes and alarms can be triggered if changes have occurred. This provides huge flexibility as the commands can be any available on the system (an example is a command that lists all 'Set UID' scripts on a file system, with alarms being sent if new SUID files are created).

### **Automated Management and Bespoke Messaging**

The rules system is powerful enough to undertake many housekeeping functions on the serverM system automatically – such as regularly archiving the system log, and even changing the rules themselves in real-time. Because the rules system allows for arbitrary commands to be executed when an alarm is triggered, the user can specify a range of additional bespoke alarms over and above those provided by the serverM system by default.

### **Flexible alarm system**

In the event of a rule being triggered, the serverM system can either send an email alarm or an SMS alarm in real-time, or execute a chosen program. The alarm system allows for intelligent alarm 'throttling' (ensuring that alarm floods do not occur for repeated rule triggering).

### **Compatible with syslog**

The powerful and informative logging system provided by serverM can write information messages to a local or remote syslog server. Alarms can also be logged to a syslog server.

### **Daily Reporting System**

serverM can be configured to generate daily reports by email, containing essential information on rule triggering history during the previous 24 hours, along with important system information messages. The daily report can be specified so as to be generated at any time throughout the day, and can be sent to up to three separate email accounts. The daily report also acts as a system 'heartbeat' - re-assuring the user that the system is running normally.

### **Real-time HTML Reporting**

The serverM system ships with a stand-alone HTML reporting tool which is capable of generating comprehensive HTML web pages detailing the current status of the system, including rules file analysis, current configuration parameters, and detailed log file analysis. This reporting tool can be automated so as to generate fresh reports in the event of any relevant changes in the current configuration or alarm status.

### **Lightweight logging system**

serverM logs a wide range of performance messages and general information messages to a single text file log which can be truncated automatically, copied, or archived. All important system messages are logged for later analysis.

### **Remote logging system**

serverM can be configured to send log entries to a centralised logging server. This provides a mechanism for monitoring of multiple serverM installations from a central point. Remote logging supports encryption.

**Self monitoring and security**

ServerM can be configured so that it checks its own system files for changes or alterations (including the master configuration and rules files), and can also be configured to send warning emails to nominated accounts in the event that the detection engine is stopped or paused for any reason (in which cases the system will also take a snapshot of vital information for later forensic examination if required).

**Ease of Installation**

serverM is simple to install (and uninstall) using a Unix archive that unpacks into a single directory. Running the system involves making minor changes to the master configuration text file (defining administration email addresses and a few other parameters) and starting the detection daemon. The daemon service responds to standard Unix signals and can be stopped or paused as required.

### **1.3 Minimum Requirements**

Operating System: most major Linux distributions; Free BSD; Mac OS X (version 10.1 or above).

Language support: Perl 5.0+ with minimum additional library installation.

Hardware: Pentium class CPU and above; 10Mb Disk Space (excluding disk space for log files etc. which will depend on log archive and size management options); 128Mb RAM.

Services: if email alarms are specified, then an SMTP email relay server is required either locally, or on a remote server that allows relaying from the serverM system.

## 2 Installation and Licensing

---

The serverM system is distributed as a standard Unix 'tar' archive file. Installation consists of decompressing the package and running the configuration tool. Uninstall consists of deleting files within a single directory. Unix signals are used for simple process management

---



## 2.1 Introduction

In order to install the serverM system, download the tarball from the website and check the MD5 hash against the one published on the serverM website. The serverM system is distributed as a \*nix tar-ball archive and can be installed by simply expanding the archive into the chosen location.

The serverM engine is a Perl script and requires that Perl version 5.0+ is installed on the target system along with a small number of additional libraries. These are detailed below.

The serverM system is capable of generating a number of reports and alarms by relaying emails through an SMTP server that may be locally installed, or running on a remote network. The system requires that an SMTP server is available that allows email relaying from the local serverM system, and for TCP/IP port 25 traffic (SMTP port) to be open from the serverM system to the SMTP relay server. If this port is currently closed by your firewall, you will need to open it before any alarm emails, or report emails, can be sent. Failure to send emails will result in error messages appearing in the log file (and syslog if this feature is configured).

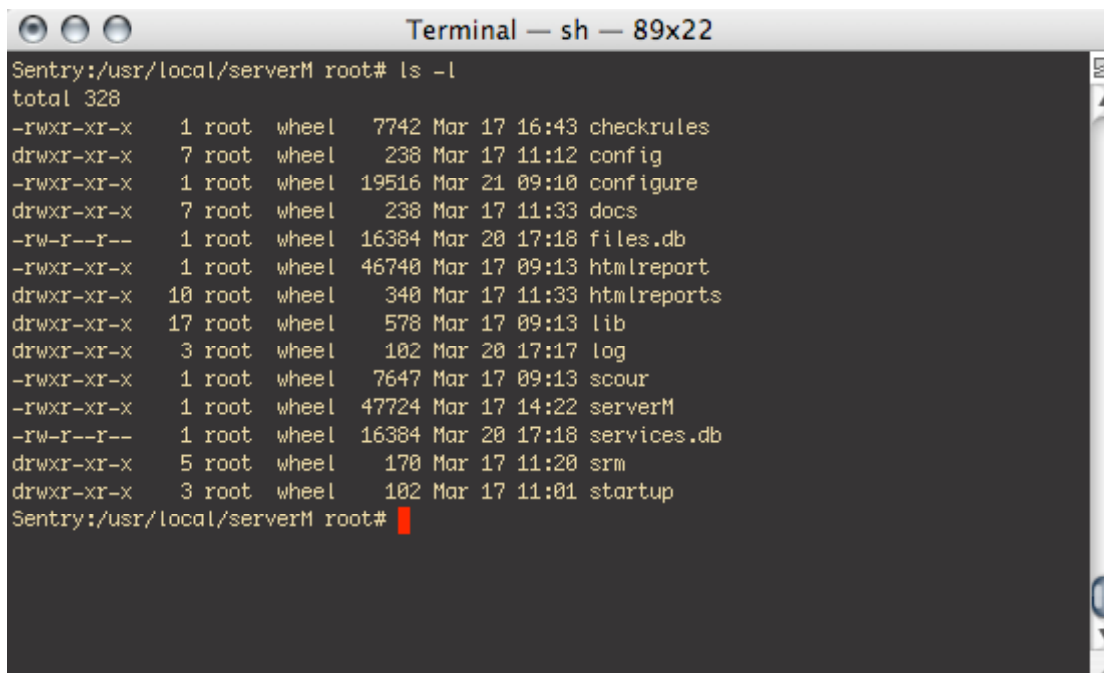
The following sections describe how to install the serverM system (and pre-requisite libraries), start and stop the detection engine, and how to uninstall the system.

## 2.2 Installing the serverM System

The serverM system is distributed as a gzip-ed \*nix tarball archive. To install the system decompress the archive in any suitable directory with the following command:

```
%>gunzip serverM.tar.gz
%>tar xvf serverM.tar
```

the archive will create a new directory ('serverM') containing further subdirectories shown in the listing below.



```
Terminal — sh — 89x22
Sentry:/usr/local/serverM root# ls -l
total 328
-rwxr-xr-x  1 root  wheel   7742 Mar 17 16:43 checkrules
drwxr-xr-x  7 root  wheel    238 Mar 17 11:12 config
-rwxr-xr-x  1 root  wheel  19516 Mar 21 09:10 configure
drwxr-xr-x  7 root  wheel    238 Mar 17 11:33 docs
-rw-r--r--  1 root  wheel  16384 Mar 20 17:18 files.db
-rwxr-xr-x  1 root  wheel  46740 Mar 17 09:13 htmlreport
drwxr-xr-x 10 root  wheel    340 Mar 17 11:33 htmlreports
drwxr-xr-x 17 root  wheel    578 Mar 17 09:13 lib
drwxr-xr-x  3 root  wheel    102 Mar 20 17:17 log
-rwxr-xr-x  1 root  wheel   7647 Mar 17 09:13 scour
-rwxr-xr-x  1 root  wheel  47724 Mar 17 14:22 serverM
-rw-r--r--  1 root  wheel  16384 Mar 20 17:18 services.db
drwxr-xr-x  5 root  wheel    170 Mar 17 11:20 srm
drwxr-xr-x  3 root  wheel    102 Mar 17 11:01 startup
Sentry:/usr/local/serverM root#
```

**Figure 2.1: Listing of the serverM Installation Archive**

---

**Note:** the file sizes and content may differ between install versions. To ensure that you have a valid installation, always check the software against the MD5 hash published on the serverM website.

---

The 'README' and 'INSTALL' text files describe latest changes and updates and installation instructions respectively. The 'checkrules.pl' script is a Perl program to enable you to check the syntactical correctness of your rules files (see Appendix E.) The 'docs' directory contains this manual along with license details and related documents. The 'lib' directory contains serverM perl libraries. The 'config' directory contains the configuration file (see next section). The 'htmlreport.pl' Perl script is a standalone Perl program that generates HTML reports on the current configuration and status of the serverM system. The 'log' directory is where serverM will store the current text log file. The 'serverM' file is the serverM engine itself.

In addition to the serverM archive, a small number of additional Perl libraries are required, these include:

Cwd  
Digest::MD5  
File::Find  
File::Basename  
POSIX  
Net::SMTP  
MIME::Lite  
Storable  
Sys::Syslog  
Crypt::CBC  
Crypt::Blowfish

The libraries can be installed using your usual Perl installation program if they are not included with your existing Perl distribution.

## 2.3 The Interactive Configuration Script

An interactive configuration script is distributed with the serverM system, and can be found in the home directory that the tarball was extracted to. The Perl script `'configure'` can be executed at any time to create a new master configuration file. The script will expect to be run in the home directory of serverM, and will create a backup of the existing configuration file called `'config.backup'` in the `'config'` directory. Executing the configuration script is straight forward:

```
%>./configure
```

Executing the script will start a dialogue that asks questions about how the user wishes serverM to behave, and will then give an opportunity to write the final configuration file to the `'config'` directory, which is where serverM looks for the master configuration file.

Although serverM is distributed with a simple, default, configuration file, it is recommended that this tool is used so that the correct recipient is provided for alarm and report emails etc. and that the SMTP server address is defined.

A sample configuration session startup and final write is shown below:

```

Terminal — perl — 89x22
Sentry:/Users/david/Documents/serverM root# ./configure

serverM master configuration file creator V1.0
type quit at any time to force quit with no changes

starting configuration process...

existing configuration copied to ./config/config.backup

Are you running serverM on a MAC OS X compatible system (if you answer no then a
generic Linux-ish system will be assumed which should work with most distros)
Press return to accept default value (no)
Type quit to stop at any time
enter yes or no [no]? █

```

Figure 2.2: Starting a Session with 'configure'

```

Terminal — perl — 89x22
mac: no
check cycle wait: 60
log: ./log/log.txt
administrator email: root@localhost
reply: root@localhost
mail server: localhost
daily reports: off
email alarms: on
syslog: off
rlog: off

Email alarms turned on - configuration:
alarm throttle: off
email alarm format: html

You now have four choices:
write - write the current config and exit
edit - return to the beginning, but with the values you entered as defaults
clear - return to the beginning with the original defaults
quit - exit the system and make no changes to the config file

What do you want to do?: █

```

Figure 2.3: Ending a Session with 'configure'

## 2.4 Setting the PERL5LIB Environment Variable

If you are intending to run any of the serverM tools such as 'htmlreport' or 'scour' then you will need to tell the Perl interpreter where to find the serverM libraries. This can be done by updating the 'PERL5LIB' environment variable e.g.

```
%>PERL5LIB=$PERL5LIB:/path_to_your_serverm/lib/ ; export PERL5LIB
```

## 2.5 Starting and Stopping the serverM System

After installation, control of the serverM system is achieved by the use of standard \*nix process control commands, or program control commands.

serverM may be executed as a standard Perl script with a number of options. The most important one is the '-D' option:

```
%>./serverM -D
```

which will start the serverM system in daemon mode (the standard mode for serverM execution). In this mode, no output is sent to the standard output (the terminal) once the initial banner is displayed, and the serverM detection engine will continue to run regardless of the interactive login status of the user that started the daemon (usually *root* – see below).

On startup, if the system is running in Daemon mode, no output is provided.

The serverM system can also be started interactively, with debug information being printed to the terminal. In order to start serverM in this mode omit the '-D' flag. Note that in this mode, the serverM process will terminate when the current user logs out of their shell.

When the serverM system is running in daemon mode, it is running in normal '*run*' state, and detection will proceed according to the rule definitions. The state can be changed to '*pause*' at any time by sending the process a 'HUP' signal. In pause mode the engine will continue to listen for signals, but will not evaluate detection rules and will not be able to detect any intrusion symptoms. The serverM detection engine can be forced back into the normal run state by sending a further 'HUP' signal:

```
%>kill -s HUP 1560
```

(where '1560' is the process ID of the serverM daemon)

The serverM engine can be stopped at any stage by sending the process a 'TERM' signal. The use of 'TERM' will cause the engine to terminate in an orderly fashion by completing the current detection cycle and then writing an appropriate closedown message to the log, and sending an email to the serverM administrator.

The following table shows the effect of using signals to change the state of the detection engine.

Current state	Signal	New State	Effects
Run	HUP	Pause	'Pause' notification email sent to administrator. Engine stops

			detecting.
Run	TERM	Stopped	Orderly closedown, engine stops, closedown email sent to administrator.
Pause	HUP	Run	'Continue' notification email sent to administrator. Engine now continues detecting.
Pause	TERM	Stopped	Orderly closedown, engine stops, closedown email sent to administrator.

**Figure 2.2: serverM daemon run states**

## 2.6 Users and Groups

The serverM system is designed to run with the privileges of a system administrator (user 'root' or similar). This is because it may be required to check for file changes in files that belong to a wide range of users or processes. However, if the detection rules that are used for a particular configuration do not require this level of access then the serverM process can be run as any appropriate user.

However, care must be taken to ensure that the serverM configuration and log files have the appropriate permissions so that the serverM system can read the configuration files, and write to the log file. In addition, it may be important that no arbitrary user is able to write to, or read from, the configuration files and this should be considered when choosing a user to run the system as.

It is recommended that the serverM system is run as root where practicable, and that the configuration and log files are read/write by root only.

## 2.7 Uninstalling the serverM System

Uninstalling the serverM system only requires the stopping of the serverM engine (see section 2.3 above) and the deletion of the serverM directory. serverM does not install or amend any other files (with the exception of any commands that you may define that execute as part of an additional user-defined rule).



## 3 Basic Concepts

---

An overview of how the serverM system operates, including the use of the master configuration file and rules file; how the intrusion detection engine, embedded within the daemon, checks for symptoms of intrusion; and an overview of how detection rules and associated alarms are created.

---



### 3.1 The Master Configuration File

The master configuration file specifies how the serverM system is to behave during the detection and alarm process. It can be edited using any simple text editor such as *vi* etc. It is also possible to edit the file on a remote server and transfer the configuration file using FTP.

The configuration settings determine a number of key operating parameters, including the speed at which the detection engine runs (and therefore the processor load, and response time for alarms), the email addresses that reports and errors should be sent to, IP addresses or domains for relaying reports through email and SMS servers, and other related configuration details.

The master configuration file can be changed at any time without stopping the main intrusion detection engine (serverM daemon), with any changes becoming active almost immediately.

---

**Note:** to ensure that the security of the system is not compromised, ensure that the master configuration file is only readable and writable by the root user (assuming the serverM daemon runs as 'root').

---

### 3.2 The Intrusion Detection Engine

The main intrusion detection engine, the a daemon referred to as the *watcher*, runs as a Unix process (although it can be run interactively for debugging purposes), and can be controlled easily using the usual Unix command line process control commands. The watcher operates in '*check-cycles*' which periodically checks each current active rule found in a rules specification file, and reacts to those rules whose constraints have been met. Depending on the current rules, and master configuration settings, alarms may be sent via email or SMS, or arbitrary code can be executed. These rules are defined using a flexible and powerful language.

The period between check-cycles is determined by a user parameter in the master configuration file, and along with all master configuration settings, can be adjusted in real-time without stopping the watcher process. Simply edit the configuration file at any stage, and the watcher will notice the differences, and reconfigure the detection engine if required.

At the beginning of each check-cycle, the watcher checks the master configuration file for any changes, and implements those changes before it evaluates the rules. The rules file is then also checked for changes (this process also checks for rules being removed, marked as inactive, or new rules being added). The watcher daemon then evaluates each active rule in a specific order, and performs any code execution action associated with any triggered rule. Any email or SMS alarms that are specified are noted, but these alarms are not sent until the end of each complete check-cycle. Typically, the delay between an alarm being triggered and an alarm message being sent is just a few seconds.

Email or SMS alarms are sent as a summary at the end of each check-cycle rather than sending one alarm per triggered rule. The watcher daemon then completes its check-cycle by performing a number of internal house-keeping functions, and finally goes to sleep for a short period.

After a specified sleep interval, the watcher daemon again wakes up and begins the next check-cycle. Typically the sleep interval is set to around 60 seconds, but can be configured for shorter or much longer periods depending upon the role of the server or desktop on which the serverM system is installed.

The watcher daemon can be asked to stop or pause at any stage during the check-cycle using the standard Unix process control commands, and the service will respond to this request at the end of the current check-cycle (see section 2.3 above for a description of the various run states of the system).

Various parameters set within the master configuration file determine how and when alarms are sent; for example, if the configuration asserts that the SMS alarm system is '*throttled*' i.e. can only send one SMS alarm in any 24 hours, and there has already been an SMS alarm sent to the nominated phone numbers during that day, then the watcher daemon

will record the triggered rule and send summary information in the daily report only, and no SMS alarm will be sent.

### 3.3 Rules, Alarms, and Reports

The serverM system uses a powerful rules description language to enable the user to formulate their own detection rules, as well as using a selection of 'off the shelf' rules devised by the author and the wider community of serverM users.

The rules specify to the watcher daemon which kinds of events and intrusion symptoms to monitor (including file changes, process changes, changes in arbitrary command output, and file sizes). Rule *pragmas* are also provided to control the way in which alarms are generated for specific rules, and to allow for the grouping of rules into more complex signatures if required.

The following example rule shows a check for unauthorised access attempts. The main body of the rule is highlighted in bold, and defines a rule name ('failed logon detected') a rule type ('status'), any constraints ('none') and an alarm (the email address).

The rule type is a 'status' and executes the defined command each check-cycle, and compares the output each time. If the output changes then it executes the alarm (in this example sending an email)

The additional syntax surrounding the rule is used to define scope and short-hand definitions etc. (see chapter 5 for a detailed explanation of how to write new rules.)

```
{ failed logon
  def execute check-failed -
    ("grep 'failed to authenticate user' /var/log/secure")

    failed logon detected::status check-failed::none:: -
      email(user@yourdomain.com)
}
```

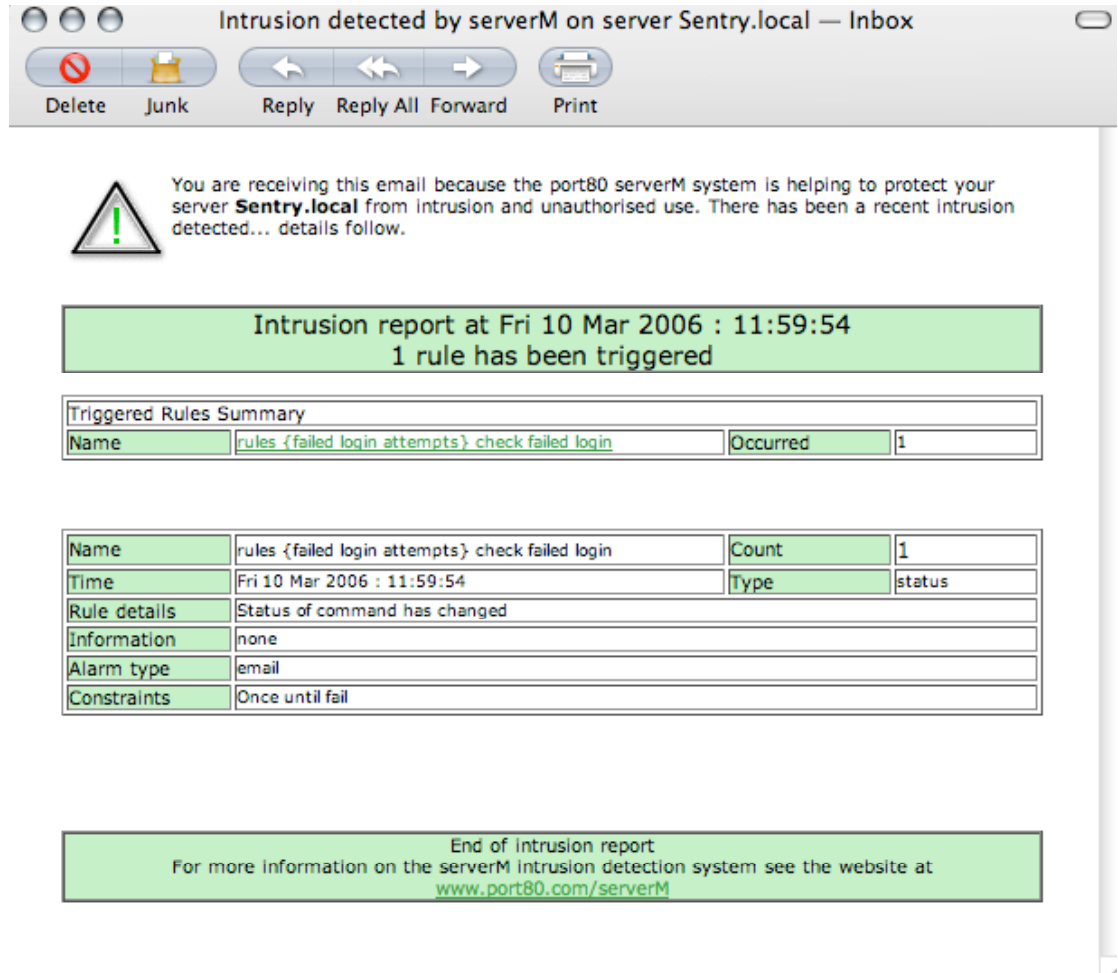
The alarm system allows for three main different types of alarm: 'email', 'SMS' (a short text message is sent to nominated mobile phones), and code execution (which enables users to define specific applications that should be executed in the event of a rule being triggered). A 'special' null alarm is also provided, which is used in more complex rule interaction (specifically when rules are grouped into complex '*signatures*').

Email alarms may be sent in either plain text or HTML (including hyper-linked summaries). Email alarms detail which rules have been triggered, and also include forensic information such as when each alarm was triggered, related system configuration information at the time of triggering, and a count of how many times each alarm was triggered since the start-up of the watcher daemon.

An example alarm email is shown below.

The alarm system is highly configurable and allows for intelligent throttling of alarms on a type-by-type basis, as well as on a rule-by-rule basis (by using rule *pragmas* – see chapter 5).

The ability to execute arbitrary code also enables the user to create his own alarm system, perhaps using a local mailing program, or network communication program.



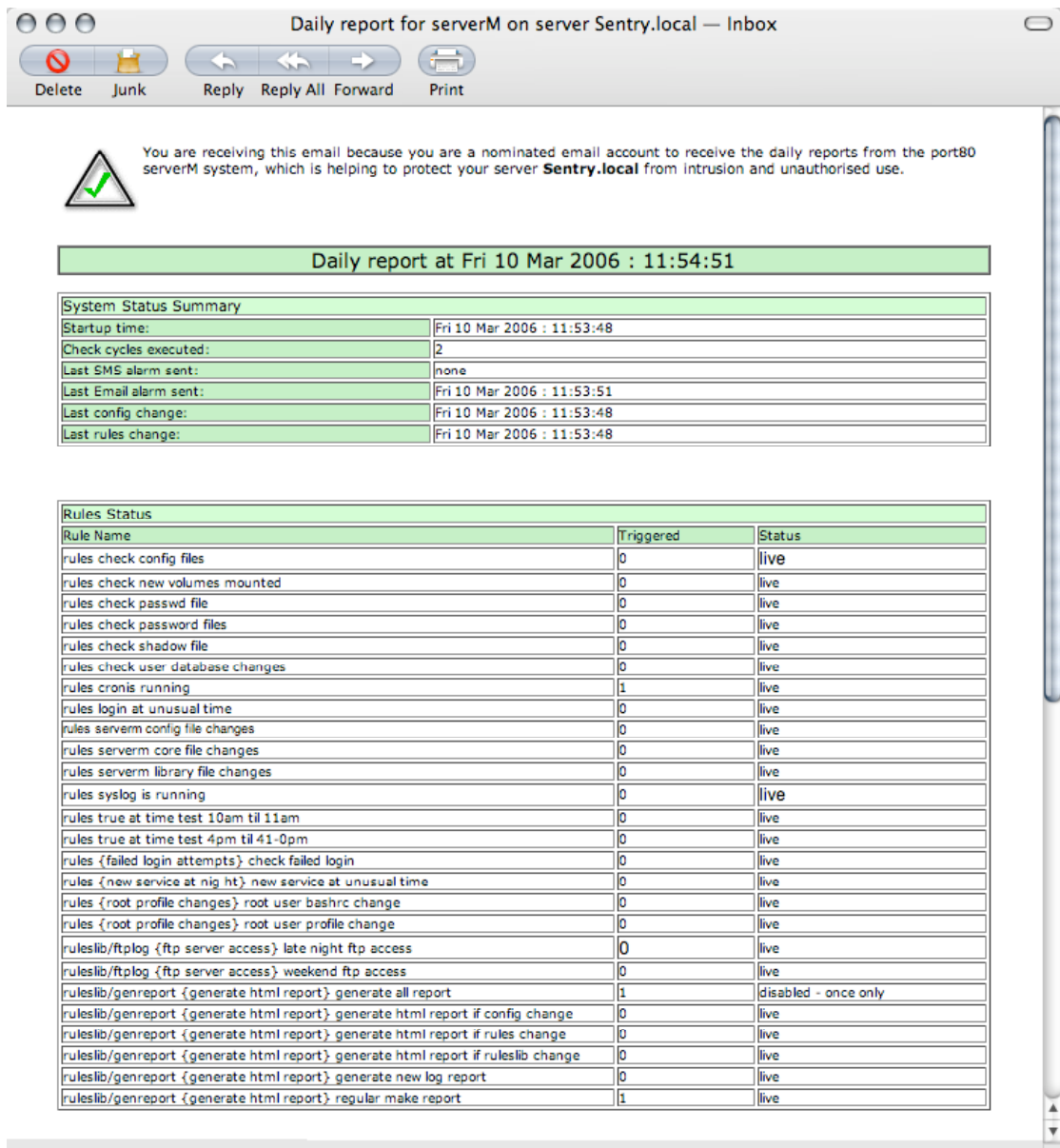
**Figure 3.3: An Example Email Alarm**

There are also a number of 'built-in' alarms which the watcher daemon can generate: in particular, if the watcher daemon is stopped, paused, or continued, then the system can be configured to send an information alarm to a nominated administrator account.

Although the rules language is extremely powerful, the process of writing correct rules can be simplified by the user of the `checkrules` command line application that ships with the serverM install. This application parses any rules file, and gives details on any syntactical errors found. See Appendix E for further details

### 3.4 Daily Reports

By default the serverM system sends daily report emails to nominated accounts at a user-specified time. These reports summarise the current configuration file and rules definitions, as well as a summary of how many times each rule was triggered in the previous 24 hour period. Even if no rules have been triggered, the daily report will be sent, demonstrating to the report recipients that the system is functioning correctly.



**Figure 3.4: An Example Daily Report Email**

Users can elect to receive daily reports at any time, and can also decide not to receive daily reports at all.

In addition to daily reporting, serverM has the capability to automatically generate HTML web page reports on the current configuration, and a detailed analysis of the current log file entries. This reporting system is implemented using the '*htmlreports*' tool which is discussed in Appendix D.

## 3.5 Logs

### 3.5.1 The Local Log File

The serverM watcher daemon writes a number of 'low level' messages to a local plain text log file. This file will contain useful peripheral information for monitoring general system behaviour and heartbeat messages etc.

The log file can be found in '%install\_path/log/log.txt' where '%install\_path' points to the directory in which the serverM system was installed<sup>1</sup>.

Each entry in the log file is of the format:

```
<date and time stamp> (<check-cycle count>) <type> message
```

for example:

```
Fri 16 July 2005 : 11:07:43 (1877) [Info] Config File Parsed OK
```

Log entries generated during system startup will have a check-cycle value of '0' (zero).

There are four types of log file entry:

- **Info:** the watcher daemon will generate a number of information messages during its normal operation – informational log file entries are usually generated by serverM internal functions such as the (correct) parsing of a configuration file.
- **Alarm:** if alarms are triggered by the watcher daemon then they will always cause an 'Alarm' log entry along with the rule name and rule type e.g.

```
Sun 8 Aug 2005 : 20:23:12 (1) [Alarm] True rule test def occurred
```

- **Error:** which indicates that the watcher service has encountered an error, and details the error message. If the 'administrator' email is set in the master configuration file, then an error message will also be sent by email to this address.
- **Status:** if the 'heartbeat' option is chosen in the master configuration file then a heartbeat message will be written to the log file at the start and end of each check-cycle with a type of 'status'. Additionally changes in the watcher daemon's run state will result in status log messages. Average check-cycle computation time is also provided with heartbeat messages.

For a complete list of messages which may be generated by the serverM system, see Appendix B – Log File Messages.

---

<sup>1</sup> The location of the log file can be changed in the master configuration file (see section 4 below)



### 3.5.2 Managing the Log File

The log file is not automatically archived or truncated, and will grow slowly over time. However, it is possible to write a simple rule in the rules file to automatically archive the log file according to a scheme that suits your own archiving policy. Section 5 shows an example rule for auto archiving the log file if required, and this can be modified to your own design.

The log file isn't held open by the wathcer daemon, and can be copied or deleted at will from the command line. If a log file does not exist when the watcher daemon is started then a new one will be created automatically.

### 3.5.3 Syslog

In addition to logging messages to the local text log, the ability to log to the local syslog daemon is included. The configuration file can be used to define which types of log messages are sent to the syslog, including the ability to assign *severity* ratings to each rule and restrict syslog entries to certain severity levels.

### 3.5.4 Remote Logging

It is also possible to log messages to a remote location by configuring serverM to send log messages across a network connection to any nominated server. To assist with remote log management, the serverM system includes a program called 'srm' (serverM remote monitor) that provides the facility to listen for serverM log messages and to interface to a centralised log file, or with a relational database.

Remote logging can optionally be encrypted to ensure that sensitive messages cannot be intercepted and read by unauthorised third parties.

Appendix F discusses the remote logging facility of serverM.

## 4 Configuring the serverM Engine

---

Within the serverM system, the *master configuration file* can be used to control system-wide configuration settings, and to control the serverM intrusion detection *watcher* daemon. The master configuration file format is described in detail, and example configuration parameters are presented.

---

## 4.1 Introduction

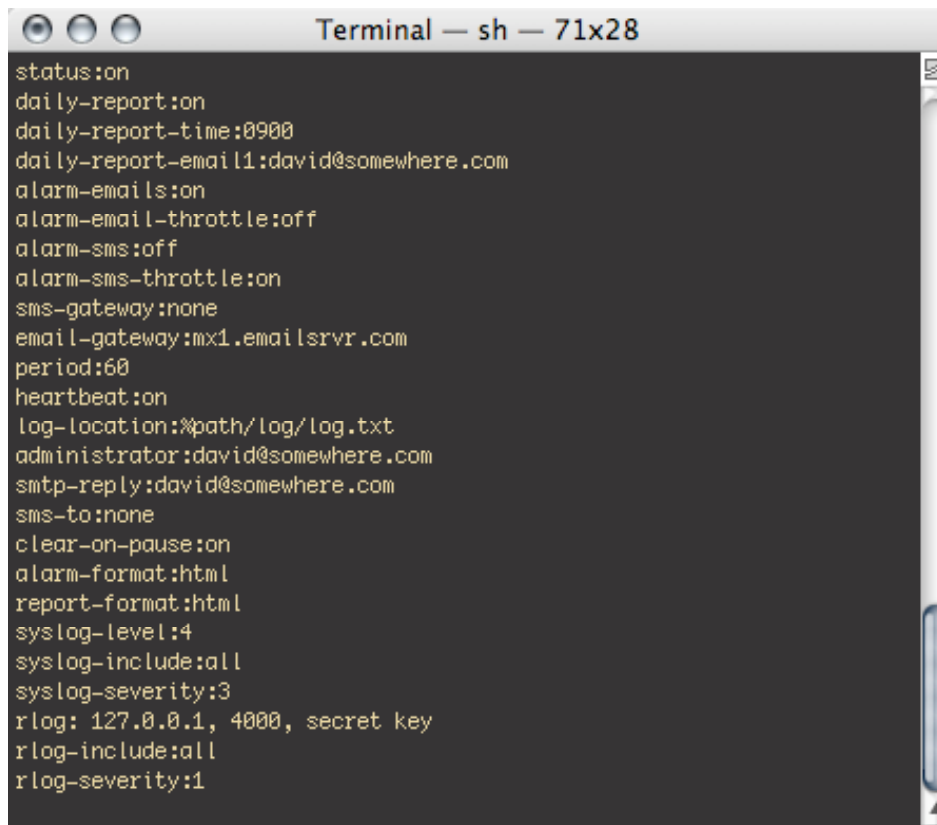
After installation of the serverM system the master config file – called *config.txt* – is located in the *config* subdirectory. This file is a simple text file, and can be edited with any text editor of your choice. In addition, a stand alone program called 'configure' ships with the serverM distribution, and can be used to provide interactive guidance on configuring the system. This section describes how configuration can be undertaken manually, and what each configuration parameter is used for.

The master configuration file contains a single line per parameter, and each line is of the form:

```
parameter-name:parameter-value
```

The master configuration file may have any number of blank lines. Any text including, and following, a 'hash' symbol ('#') will be ignored. Parameter name and value pairs may not be split over multiple lines, and there must be only one parameter name and value pair per line.

The order in which the parameter name and value pairs appear in the file is not significant.



```

Terminal — sh — 71x28
status:on
daily-report:on
daily-report-time:0900
daily-report-email1:david@somewhere.com
alarm-emails:on
alarm-email-throttle:off
alarm-sms:off
alarm-sms-throttle:on
sms-gateway:none
email-gateway:mx1.emailsrvr.com
period:60
heartbeat:on
log-location:%path/log/log.txt
administrator:david@somewhere.com
smtp-reply:david@somewhere.com
sms-to:none
clear-on-pause:on
alarm-format:html
report-format:html
syslog-level:4
syslog-include:all
syslog-severity:3
rlog: 127.0.0.1, 4000, secret key
rlog-include:all
rlog-severity:1

```

**Figure 4.1: Example Master Configuration File**

The following sections describe each parameter in detail.

## 4.2 Parameters

### 4.2.1 status

**Permitted values:** 'on' 'off'

**Default value (with initial configuration file from install):** 'on'

If the status is 'off' then the watcher daemon will continue to run, but will take no actions apart from the 'heartbeat' log message (if it is selected - see below). If the status is 'on' then the service will run normally. This parameter is usually set to 'on', and the 'off' setting is rarely used in normal operation. Occasionally it may be useful to set the status to 'off' in order to stop alarm flooding whilst the rules file is edited (stopping or pausing the service may have effects on future rule evaluation - see section 5, specifically the rule *pragmas*, and the 'clear-on-pause' configuration parameter below).

**Example:** status:on

### 4.2.2 daily-report

**Permitted values:** 'on' 'off'

**Default value (with initial configuration file from install):** 'off'

The serverM watcher daemon generates daily reports at a time specified by the user in the '*daily-report-time*' parameter. If the '*daily-report*' parameter is set to 'off' then no daily report is generated or sent. If this is set to 'on' then the daily report will be sent to the email addresses specified in the '*daily-report-email*[1,2,3]' parameters.

**Example:** daily-report:on

### 4.2.3 daily-report-time

**Permitted values:** 0000-2359

**Default value (with initial configuration file from install):** '0900'

This is the time, in 24-hour clock notation, when the daily report will be sent (if the '*daily-report*' parameter is set on 'on'.) The value '0000' represents midnight at the beginning of each day.

**Example:** daily-report-time:1645

### 4.2.4 daily-report-email[1,2,3]

**Permitted values:** 'email address' 'none'

**Default value (with initial configuration file from install):** 'none'

If the parameter '*daily-report*' is set to 'on' then the daily report email will be sent to the three addresses set in these parameters. The optional value of 'none' is used if there are fewer than three addresses to which daily reports should be sent.

**Example: daily-report-email1: serverm\_reports@yourdomain.com**

#### 4.2.5 email-alarms

**Permitted values: 'on' 'off'**

**Default value (with initial configuration file from install): 'on'**

If the status is 'off' then the watcher daemon will not send any email alarms, regardless of whether or not any email alarm rules are triggered. Email alarms that are triggered will still give rise to entries in the serverM log file however.

**Example: email-alarms:on**

#### 4.2.6 alarm-email-throttle

**Permitted values: 'on' 'off'**

**Default value (with initial configuration file from install): 'off'**

If the status is 'on' then there will be a maximum of one email alarm sent to *any* email alarm recipient (defined within the rules file, see following section) within any 24 hour period (measured from midnight). The first email alarm will effectively turn off the email alarm system until midnight that night. Email alarms will still result in entries in the serverM log file however. This option can be used to avoid alarm flooding, but is rarely used because vital alarms may be missed if emails are the main alarm delivery method.

**Example: alarm-email-throttle:off**

#### 4.2.7 alarm-sms

**Permitted values: 'on' 'off'**

**Default value (with initial configuration file from install): 'off'**

If the status is 'off' then SMS alarms will not result in any SMS messages being sent to any recipients. SMS alarms will still result in entries in the serverM log file however.

**Example: alarm-sms:on**

#### 4.2.8 alarm-sms-throttle

**Permitted values: 'on' 'off'**

**Default value (with initial configuration file from install): 'on'**

If the status is 'on' then there will be a maximum of one SMS alarm sent to any SMS alarm recipient (defined within the rules file, see following section) within any 24 hour period (measured from midnight). The first

SMS alarm will effectively turn off the SMS alarm system until midnight that night. SMS alarms will still result in entries in the serverM log file however. This option can be used to avoid SMS alarm flooding, and is commonly used to avoid the expense generated by multiple SMS alarms being triggered within a short time period.

**Example: alarm-sms-throttle:on**

#### 4.2.9 sms-gateway

**Permitted values:** 'ip address' 'domain' 'none'

**Default value (with initial configuration file from install):**  
'localhost'

This is the IP address, or domain, of the SMTP server that houses the 'email to SMS' server. For more details of how to use this feature (and configure an email to SMS service) see Appendix A.

**Example: sms-gateway:mail.emailtosms.com**

#### 4.2.10 email-gateway

**Permitted values:** 'ip address' 'domain' 'none'

**Default value (with initial configuration file from install):**  
'localhost'

This is the IP address, or domain, of the relay SMTP server that will be used to accept the email alarm emails and other email reports generated by the serverM watcher daemon. It must be configured to relay emails from the server or desktop computer that houses the serverM installation. Any firewalls must allow for SMTP traffic from the serverM installation to the SMTP gateway. A local address can be used if an SMTP server is available locally.

**Example: email-gateway:mail17.yourdomain.com**

**Example: email-gateway:127.0.0.1**

#### 4.2.11 period

**Permitted values:** 1 – 'any number'

**Default value (with initial configuration file from install):** '60'

This is the period, measured in whole seconds, for which the serverM watcher service will relinquish the CPU between check-cycles. Thus, by lowering the value, the service will run more frequently and react to intrusion symptoms more quickly, but consume a higher level of system resources. Note that this is *not* the period between the start of each check-cycle as the check-cycle will typically take a few seconds to finish execution before the service sleeps. Typically, a period of 60 seconds is adequate for many systems.

**Example: period:60**

#### 4.2.12 heartbeat

**Permitted values:** 'on' 'off'

**Default value (with initial configuration file from install):** 'on'

If this parameter is set to 'on' then a heartbeat message will be written to the serverM log file at the start and end of each check-cycle. This can be useful if a third-party log analyser wishes to gather statistics about check-cycle speed of execution etc. It can also be used to ensure that the serverM system is still functioning as, without this parameter being set, there may be no entries in the log file generated during a check-cycle.

**Example: heartbeat:on**

#### 4.2.13 administrator

**Permitted values:** 'email address' 'none'

**Default value (with initial configuration file from install):**  
'root@localhost'

This is the email address to which serverM will send its error reports and general system status reports (such as service status change reports). If this parameter is set to 'none' then no such emails will be generated.

**Example: administrator:serverm\_admin@yourdomain.com**

---

**Note:** for security reasons this is the only configuration parameter which cannot be changed whilst the serverM watcher process is running: the service needs to be stopped, and then re-started, for this parameter change to be effective. If the parameter is changed during execution then an alarm message will be sent to the current administrator address but the parameter will remain unchanged. The only exception to this is when there is currently no administrator email account – in this instance a new one may be defined.

---

#### 4.2.14 smtp-reply

**Permitted values:** 'email address' 'none'

**Default value (with initial configuration file from install):**  
'root@localhost'

This is the email address which will appear in the reply-to field of any email alarms, or alarm reports, which are sent by the serverM system. If the parameter is set to 'none' then the reply-to address will also appear as 'none'.

**Example: smtp-reply:serverm\_support@yourdomain.com**

#### 4.2.15 sms-to

**Permitted values:** 'email address' 'none'

**Default value (with initial configuration file from install):** 'none'

Many email to SMS services authenticate on the envelope sender of the email – and this parameter enables this address to be set. See Appendix A for more details.

**Example:** sms-to:user4774@yourdomain.com

#### 4.2.16 clear-on-pause

**Permitted values:** 'on' 'off'

**Default value (with initial configuration file from install):** 'on'

The clear-on-pause parameter controls how the serverM watcher service responds to a service pause. If this parameter is set to 'on' then, if the service is paused at any time, all rules which are affected by certain *pragmas* (specifically 'once only' and 'once until fail') will be reset as if they had never been triggered (see the following section for details of rule pragmas).

**Example:** clear-on-pause:on

#### 4.2.17 alarm-format

**Permitted values:** 'html' 'text'

**Default value (with initial configuration file from install):** 'html'

Email alarms can be sent either in HTML or text formats. This parameter sets the format for *all* email alarms sent by the serverM system. It does not however affect the administrator emails (error reports and service status change reports) which are always sent in plain text format.

**Example:** alarm-format:html

#### 4.2.18 report-format

**Permitted values:** 'html' 'text'

**Default value (with initial configuration file from install):** 'html'

The daily report can be sent either in HTML or text format. This parameter sets the format for the daily report email sent by the serverM system. It does not however affect the administrator emails (error reports and service status change reports).

**Example:** report-format:html



#### 4.2.19 syslog-level

**Permitted values: digit 0-9**

**Default value (with initial configuration file from install): 4**

If a syslog-level is set then all messages, including alarm messages, that are written to the serverM log file, are also written to the local syslog server (filtered by the 'syslog-include' parameter discussed below). The level defines the syslog level relevant to the local server configuration.

**Example: syslog-level: 4**

#### 4.2.20 syslog-include

**Permitted values: comma separated list of 'info', 'error', 'status', or 'all'**

**Default value (with initial configuration file from install): all**

If a syslog-level is set then all messages of the types described with this parameter are sent to the system's local syslog. The special value of 'all' represents all possible message types. See the following section for a description of the logging system.

All alarm messages will be written to the syslog regardless of this setting if the 'syslog-level' parameter is set (but may be filtered by the 'syslog-severity' parameter described below).

**Example: syslog-include: status**

#### 4.2.21 syslog-severity

**Permitted values: number 0-99**

**Default value (with initial configuration file from install): 0**

All detection rules can be specified with a numerical severity using a simple pragma in the rules definition file. By setting this configuration parameter, any rules that are triggered, but which have a severity greater than this value set here are not logged to syslog. If no severity is set for a specific rule then it is assumed to be '0' i.e. the highest severity.

**Example: syslog-severity: 7**

---

**Note:** it is possible on many Unix systems to log directly to a local or remote syslog server through a system command. In such systems, a rule-by-rule syslog alarm is easily defined using the arbitrary command alarm type (see next chapter).

---

#### 4.2.22 log-location

**Permitted values: filespec (including special variable %path)**

**Default value (with initial configuration file from install):**

**`%path/log`**

By default, all log messages directed to the text log will be written to the file `%path/log` where `%path` is the install path for the serverM script. If this parameter is included in the configuration file then the specified location and file will be used instead. The special variable `%path` will be replaced at run time with the current install path. By specifying the file `/dev/null` as the target for log messages, no logging will effectively take place (although this is strongly discouraged unless detailed `syslog` functionality is being used!)

**Example: log-location: /var/log/serverm.log**

#### 4.2.23 rlog

**Permitted values: <IP or domain>, port number (1-65535), key**

**Default value (with initial configuration file from install): none**

Log entries may be sent across the network to a remote logging server. This parameter defines where the server resides (either an IP address or a domain), the UDP port to use to transmit the messages on, and an optional encryption key. The encryption method used is Blowfish, and the key can be any string of characters. The key is optional and, if excluded, will result in logging messages being sent in clear text. See Appendix F for remote logging details.

**Example: 192.168.0.34, 4000, \$em356G^%hhdh**

#### 4.2.24 rlog-severity

**Permitted values: number 0-99**

**Default value (with initial configuration file from install): 0**

All detection rules can be specified with a numerical severity using a simple pragma in the rules definition file. By setting this configuration parameter, any rules that are triggered, but which have a severity greater than this value set here are not logged to the remote logging server specified by the `rlog` parameter. If no severity is set for a specific rule within the rules definition file then it is assumed to be `0` i.e. the highest severity.

**Example: rlog-severity: 3**

#### 4.2.25 rlog-include

**Permitted values: comma separated list of 'info', 'error', 'status', or 'all'**

**Default value (with initial configuration file from install): 'all'**

If an 'rlog' server is defined then all messages of the types described with this parameter are sent to the remote log server. The special value of 'all' represents all possible message types.

All alarm messages will be written to the remote log regardless of this setting (but may be filtered by the 'rlog-severity' parameter described below).

**Example: rlog-include: status**

#### 4.2.26 user-def

**Permitted values: any**

**Default value (with initial configuration file from install): none**

This parameter is used within the rules configuration files to perform conditional parsing. The user may define as many 'user-def' parameters as required (one per line of the configuration file), and then reference the values within the rules file as required.

For example, the configuration line:

```
user-def: MAC OS X
```

will enable the user to include the conditional rule parsing lines:

```
ifdef MAC OS X
... some rule definition(s)
endif
```

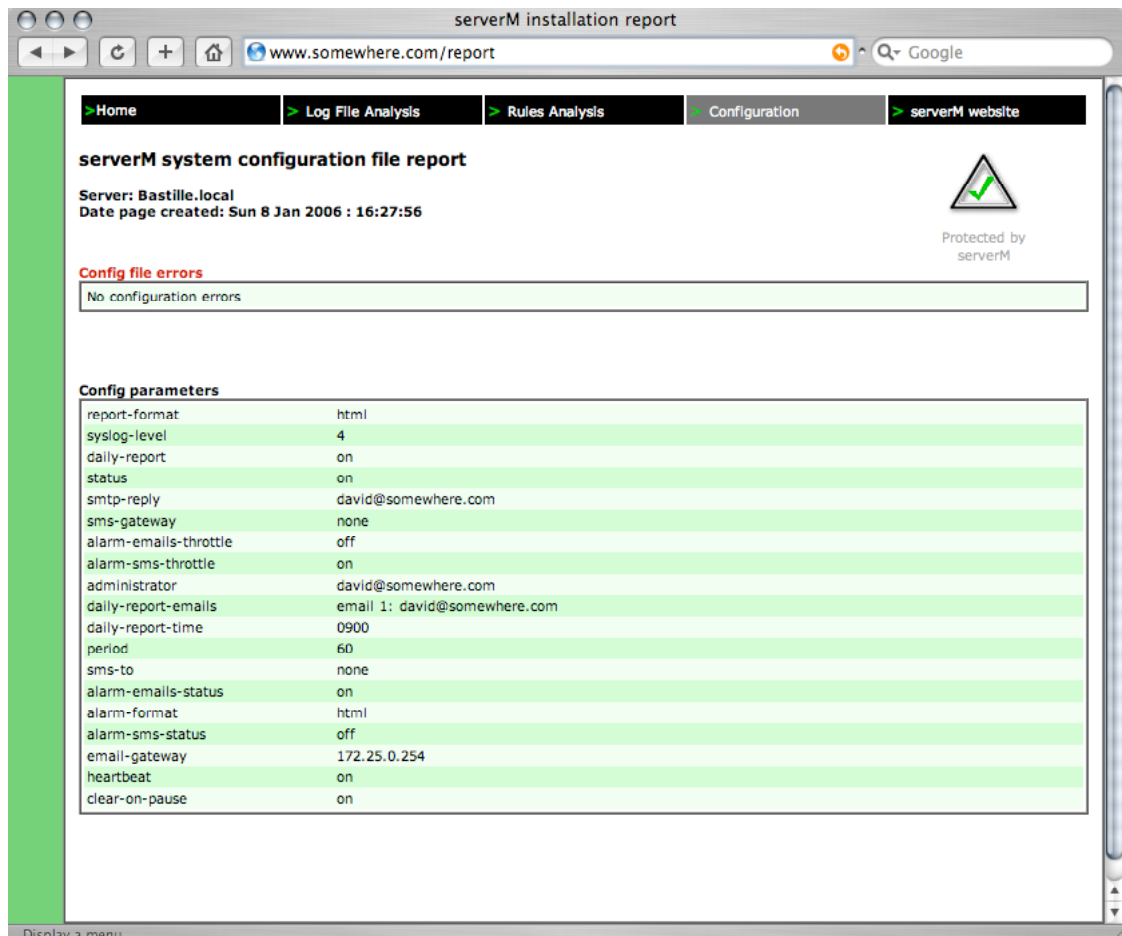
within a rule file, and these lines will be parsed as normal. If a parameter is named within an 'ifdef' scope that is not defined by a 'user-def' within the master configuration file, then all lines between the 'ifdef' and next 'endif' will be ignored by the rule parser.

**Example: user-def: RedHat Linux 7.2**

### 4.3 Using the 'htmlreports' Tool

serverM provides the capability of reporting on the current configuration settings, and highlighting any configuration file errors, by generating HTML web pages detailing the configuration file. The 'htmlreports' tool generates report pages based on a parsing of the current configuration file.

An example configuration analysis report page generated by the htmlreport tool is given below.



**Figure 4.2: The htmlreport Tool Configuration File Analysis Page**

Appendix D discusses the use of this tool in detail.

## 5 Designing and Implementing Rules

---

The serverM system provides a powerful and flexible rules language for describing which kinds of events and symptoms of attempted intrusion that the detection engine should look for, and how to react to those events. The rule syntax is described in detail, along with examples and tips on rule writing.

---

## 5.1 Introduction – The Rules File

The intrusion detection engine does not use a fixed set of signatures to detect intrusion attempts. Instead it uses a rule language that can be used to define those symptoms of attack that the user may be interested in. This provides for a greater degree of flexibility, and enable users to fine tune the detection engine for each deployment.

The rules used by the intrusion detection engine are defined in plain text files, and can be edited by any text editor or word processor. The main rules file is called `rules.txt` and can be found in the `config` directory of the main installation directory. This rules file is always parsed first. In addition, any text files with the extension `.txt` that are in the directory `config/ruleslib`, are treated as additional rules files and are also parsed by the engine. The `rules.txt` file should not be renamed, and is always read and processed first. Many serverM installations only have this one rules file.

The rules definition language is very powerful, and can be very complex. However, it is possible to write extremely effective detection rules based on a very small subset of the rules language, and rule definitions only use the more complex language constructs when fine-tuning, or devising further bespoke intelligent detection and protection mechanisms.

Example rules files are provided with the standard serverM installation, and can be used 'out of the box' with little or no changes. Section 7 describes these files in detail. In addition, pre-defined rules files can be downloaded from the serverM website at <http://www.serverm.org>. To use these files, download them and save them in the `config/ruleslib` directory.

Each rules file consists of a collection of individual rules, and allows for arbitrary comments and flexible formatting. Optional conditional parsing statements and scoping statements can provide for rules files that are both portable and easily maintained.

## 5.2 Defining a Single Rule

The basic construct of the rules file is the individual rule definition. Each rule definition is divided into four parts, with optional starting '*pragmas*' that can be used to affect the conditions under which the rule is triggered. Each rule definition is written in the form:

```
[optional pragmas] name :: rule type :: rule constraints :: alarm
```

The double colons (::) are used as separators between the rule parts and therefore cannot be used elsewhere within the rules. The exception to this is within the 'alarm' part of a rule specification, where double colons may be needed in the specification of code to execute when the rule is triggered. In this instance double colons may be used freely within the 'execute command' part of the alarm.

The **name** must be unique so that the system can provide meaningful reports on which rules have been triggered. Failure to provide a unique name for each rule will generate an error message in the serverM log file, and (if configured) an error email. The system will attempt to recover from such rule syntax errors without affecting the rest of the defined rules. serverM also provides a scoping mechanism which allows for non-unique rule names across different rules files, as long as they occur within *signatures* (described in section 5.8 below).

The **rule type** specifies which kind of intrusion symptom the rule is to check: for example, a change in the result of running a command, or the content of a specified file changing.

The **constraints** are specific to the rule type and provide additional conditions that must be met in order for the rule to trigger the alarm. For example, the time period within which an entry must have been written to the system log, or the minimum size of a file which is being checked for exceeding the size threshold.

The **alarm** determines what action is taken if and when a rule is triggered. Alarms can include sending an email, sending an SMS message, or even arbitrary code execution. Multiple alarms can be associated with a single rule.

Rule definitions may be split over multiple lines within a rules file by using a 'line split' character which is the minus symbol ('-'). When the parser recombines split lines, any white-space before and after a line split character will be ignored i.e. the rule parser will concatenate split lines by removing the line split character and *all* white space around it. Care should therefore be taken when splitting lines at a point where white space is significant to the rule definition.

Leading white-space (spaces and tabs) in front of the rule, and at the end of a rule is not significant.

An example split line rule is shown below:

```
rule name :: -
```

```
rule type::rule constraint:: -
alarm
```

This rule will be concatenated by the parser to:

```
rule name ::rule type::rule constraint::alarm
```

Alternatively, if white-space is significant at the point where the line split character is used, the double minus ('- -') may be used. In this case the lines will be concatenated in the same way, but an extra space is inserted within the line where the double minus occurred. This is useful when you may need to split a line where white-space is significant (to separate commands in an 'execute' alarm for example – see an example below.)

Comments may be used freely within the rule file by using the hash ('#') symbol. All text following a hash is ignored until the end of the line, but the hash must not be used within a split-line rule e.g.

```
# this is a comment within the rule file

rule name:: - # this is an illegal use of a comment!
rule type::rule constraint:: -
alarm # this is a perfectly legal use of a comment!
```



### 5.3 Rule Execution Order

Rules are executed strictly depending upon the order that they appear in the main rules file i.e. the top-most rule first. This ordering can be very significant when using rule *pragmas*, and other definitions within the rules files. Rule execution order can also be significant when 'execute' alarms are used that may affect the triggering of rules later in the rules file.

If additional rules files are defined in files within the '*ruleslib*' subdirectory, then these files will be read in alphabetic order by filename after the main rules file, and the rules within the files are ordered by their order of appearance in their respective files.

The master *rules.txt* file, and any files in the rules subdirectory can be edited at will, and new files added to the rules subdirectory as needed.

The parser reads and parses the rules files in a single pass. This means that all definitions that are referenced within a rule definition must have been declared at an earlier stage in that specific rules file, or in a rules file that is parsed before the current one.

It is possible to make *global* definitions that can be used in all rules files that are parsed after the point that the definitions are parsed.

## 5.4 Rule Pragmas and Rule Checking

### 5.4.1 Introduction

Rule definitions may include optional rule *pragmas*. These pragmas tell the intrusion detection engine that the rule needs to be evaluated only under certain conditions, or that alarms should be triggered only under certain conditions. The pragmas relate to issues such as telling the engine to only evaluate a rule on an occasional basis, or at specific times, or only if other rules have been evaluated and certain alarms triggered.

There are eleven optional rule *pragmas* that affect rule evaluation and alarm triggering, and which can be used in any combination within a single rule. Rule pragmas are defined at the beginning of the rule that they affect, and there are no limits to how many pragmas are defined for each rule.

An example rule pragma is the 'every' pragma. Without this pragma, a rule will be evaluated every check cycle. By using the pragma, it is possible to tell the detection engine to only evaluate the rule every 'n' check cycles. The following rule demonstrates this:

```
[every 10] rule-name::type::constraint::alarm
```

This rule will only be evaluated on every 10<sup>th</sup> check cycle. On every other check-cycle the intrusion detection engine will ignore the rule completely.

Another example pragma is the 'once only' pragma, which tells the intrusion detection engine to execute the alarm associated with a rule only once and then to 'turn off' the alarms for that rule completely. The following rule demonstrates this pragma:

```
[once only] rule-name::type::constraint::alarm
```

Any number of pragmas can be defined for a single rule, so it would be possible to define the following rule:

```
[every 10] [once only] rule-name::type::constraint::alarm
```

This rule will be evaluated only every 10<sup>th</sup> check-cycle, and if on one of those check cycles it triggers the alarm, the alarm will be executed and then turned off.

### 5.4.2 Pragmas for Controlling Rule Evaluation and for Alarm Triggering

The two example pragmas above affect the intrusion detection engine in two different ways. The first pragma ('[every n]') affects whether or not a rule is evaluated: it is called an **evaluation pragma**. The second pragma ('[once only]') has no affect on whether the rule is evaluated, only on whether or not an alarm is triggered if the evaluation results in a positive detection: it is called an **alarm pragma**. The distinction is very important, because there may be instances when a rule should be evaluated, but no alarm should be sent even if a positive detection is made as a result of rule evaluation.

Within each check-cycle the detection engine tests to see whether or not a specific rule should be **evaluated**. In the absence of any evaluation pragmas, the rule will always be evaluated in every check-cycle..

After any evaluation pragmas are tested, those rules that are to be evaluated are checked to see whether or not the rule is *triggered* i.e. whether all the constraints within the rule are met and a positive detection is made. A triggered rule does not necessarily mean that the alarm will be consequently triggered, only that during this check-cycle the rule's constraints are met.

For those rules which are evaluated, and which have been triggered, the alarm pragmas are used to check whether or not the alarm is then triggered. If none of these pragmas are present, a triggered rule will always trigger an alarm.

Evaluation pragmas and alarm pragmas are discussed in more detail below, along with the special 'severity' pragma which affects syslog output.

### 5.4.3 Evaluation Pragmas

There are six *evaluation pragmas*:

`[if-and <rule name list>]` – will only allow the detection engine to evaluate the rule if all of the rules named in the `<rule name list>` have been triggered during the current check-cycle (i.e. have been evaluated, and their conditions met – not necessarily whether or not their alarms have been triggered). The shorthand `'*'` can be used to define a list which contains *every* rule known to the detection engine (across all rules files and scopes). This shorthand is only relevant if this is used on the very final rule of the check-cycle (or there will be rules which won't have had a chance to be evaluated and triggered!)

`[if-or <rule name list>]` – will only allow the detection engine to evaluate the rule if one or more of the rules named in the `<rule name list>` has been triggered during the current check-cycle (i.e. has been evaluated, and its conditions met – not necessarily whether or not its alarm has been triggered). The shorthand `'*'` can be used to define a list which contains *every* rule known to the detection engine: using `[if-or *]` will mean a rule is evaluated if *any* other rule has been triggered before this rule is reached.

`[at-time <times>]` – will only allow the detection engine to evaluate the rule if the current time of day is within one of the periods specified in the list `<times>`. This list consists of start times and end times (in 24 hour clock format 'hhmm') separated by colons. The periods are separated by commas.

**Example:** this rule should only be evaluated between 8pm and 6am

```
[at-time 2000:2400, 0000:0600] rule def..
```

`[on-day <days>]` – will only allow the detection engine to evaluate the rule if the current weekday one of the days listed in the `<days>` list. This list consists of the names of days (first three letters or more of each day), separated by commas. The special day name 'weekday' is used as shorthand for Monday through to Friday inclusive. Case is not significant.

**Example:** this rule will not be evaluated on a Sunday:

```
[on-day weekday, sat] rule def ...
```

In addition, the special keyword '*daily*' can be added to the list (or used on its own) to tell the serverM detection engine to evaluate the rule at most once every 24 hours (measuring from the time of the first evaluation).

`[every n]` – will only allow the detection engine to evaluate the rule if the check-cycle number, which increments every check-cycle, divides exactly by 'n'. The value '0' is not allowed as a count, and the value '1' is redundant (the implicit default for all rules). Any value other than '0' is permissible.

---

**Note:** regardless of the value of 'n', each `[every n]` pragma rule will be evaluated on the first check-cycle.

---

`[on-cycle n]` – will only allow the detection engine to evaluate the rule if the check-cycle number, which increments every check-cycle, is exactly equal to 'n'. The value '0' is not allowed as a count (there is no check cycle with the value zero). Any value other than '0' is permissible.

The injudicious use of the `[every n]` pragma can give rise to some unexpected behaviour, specifically with those types of rules that use historic data. Even though a rule may only be evaluated every 'n' check-cycles, it will take note of all relevant history data generated by the detection engine.

#### 5.4.4 Alarm Pragmas and the 'Severity' Pragmas

There are four pragmas that effect whether or not the specific rule alarm is *active* or not i.e. whether the detection engine will trigger the alarm of the rule during a given check-cycle (assuming the rule constraints are met). Note that on their own, these pragmas do not affect whether or not the rule is evaluated, but only whether the resulting alarm(s) are triggered.

These four *alarm pragmas* are:

`[once only]` - which asserts that the rule's alarm(s) will become inactive after the rule has been triggered for the first time, and will not cause the rule alarm(s) to be triggered in future check-cycles even if the constraints are met.

`[once until fail]` - which asserts that the rule's alarm will become inactive after the rule is triggered for the first time, but will become active again if a future check-cycle occurs in which the constraints are **not** met i.e. the alarm becomes active again on the check-cycle following the first check-cycle that the rule is not triggered (this can be extremely useful to enforce alarm throttling).

`[throttle <n>:s|m|h|d]` - which asserts that this rule should give rise to *at most* '*n*' alarms in any second, minute, hour, or day.

**Example** this rule will restrict alarms from this rule to at most 5 in any one hour:

```
[throttle 5:h] rule def...
```

`[initial]` - which asserts that this rule should not give rise to an alarm on the first check cycle, even if the rule evaluation determines that an alarm normally would be triggered. This pragma is especially useful for file change rules where the non-existence of a file on the first check cycle would otherwise give rise to an alarm.

These four alarm pragmas stay in effect from the start of the intrusion detection service, but will be reset if the service is stopped and then re-started. In addition, if the '*clear-until-pause*' option is defined in the master configuration file, then pausing and then continuing the service will also re-set the rule pragma state for all rules (see Chapter 4 above).

The '*once until fail*' pragma is particularly useful in 'file size' rules - consider monitoring the size of a log file whereby the rule is triggered once the file is larger than a certain size - this pragma will ensure that the rule will not constantly alarm each check-cycle, but will wait until the file is below the specified size and then will re-activate ready for the next time it grows beyond that size again.

Finally, the '*severity*' pragma defines whether or not a triggered alarm is written to the syslog:

`[severity n]` - defines the severity level of the alarm and can be used to filter which alarms are reported to the syslog in conjunction with the '*syslog-severity*' parameter in the master configuration file. For example, a '*syslog-severity*' setting of 5 means that a rule with a severity pragma of `[severity 6]` will not be logged to the syslog, but an alarm with a severity pragma of `[severity 5]` would be.

### 5.4.5 Pragma Syntax

The syntax for rule pragmas is straight forward: they must appear at the front of the rule, and may have optional white-space between the pragmas, and optional white-space between the pragmas themselves and the rule name. All pragmas are enclosed within square braces ('[]').

For example, the following rule will be evaluated every 100 check cycles, and after the first alarm triggering will turn off the alarm triggering until the rule fails to evaluate to a positive detection:

```
[every 100] [once until fail] name :: rule type:: -
constraints::alarm
```

### 5.4.6 Pragma Examples

The distinction between rule evaluation and rule triggering provides a powerful control mechanism for the detection rules. Consider the following three rules:

```
[every 10] rule 1::rule type::rule constraints::alarm
[every 10] rule 2::rule type::rule constraints::alarm
[every 10] rule 3::rule type::rule constraints::alarm
.
.
[if-and rule 1:rule 2] rule 5::rule type:: -
rule constraints::alarm
[if-and rule 3:rule 4] rule 6::rule type:: -
rule constraints::alarm
[if-or rule 5:rule 6] rule 7::rule type:: -
rule constraints::alarm
```

Rule 7 will only be evaluated if one of rules 5 and 6 is evaluated and triggered, which will in turn only be evaluated if both of their named 'if-and' rules are evaluated and triggered. In addition, because rule 1 and rule 3 are only evaluated every 10 check-cycles, it is only possible for rule 7 to be evaluated every 10 check-cycles i.e. the 'every' pragma propagates through 'if-and' pragmas.

This is a very complex use of pragmas and would rarely be found in most deployments of serverM.

To illustrate the difference between evaluation pragmas, and alarm pragmas, consider the following pair of rules:

```
[once only] rule 1::rule type::rule constraints::alarm
[if-and rule 1] rule 2::rule type::rule constraints::alarm
```

Even though rule 1 will only ever give rise to a single alarm i.e. the first time it is triggered, it will continue to be evaluated every check-cycle (it has no evaluation pragmas), and in any check-cycle that its evaluation results in a trigger, rule 2's alarm will trigger if its own constraints are also met.

## 5.5 Rule Syntax: Rule Names

Rule names may be any combination of characters (letters, numbers, punctuation etc.) including spaces, but must have at least one alpha-numeric character. However, leading and trailing spaces are ignored, and double colons ('::') are not allowed within a name. Single colons are also not allowed as they may interfere with *[if-and]* and *[if-or]* pragma rule name lists. Rule names should not be longer than 250 characters (including space).

It is advisable to give rules meaningful names where possible as these names appear in alarms and reports generated by the system.

Each rule must have a unique name within the same rules file: duplicating a name will give rise to an error report in the log, along with an error email to daily report recipients (if configured), and will also cause the last rule in the rules file with the duplicated name to be taken as the live rule (earlier duplicates are ignored). The exception is rule names within named *signatures* (a scoping mechanism with the rules definitions) where rule names are automatically prefixed with their scope name to ensure uniqueness (see section 5.8 below).

## 5.6 Alarms

### 5.6.1 Overview

The alarm system enables the user to determine which kind of action should be taken when a rule is triggered. There are four possible alarm types:

**email:** specifying that an email alarm should be sent to the nominated email alarm account(s). Details of which rule was triggered, along with many other relevant details will be included in the email alarm;

**sms:** specifying that an SMS alarm should be sent to the nominated SMS alarm phone numbers. Truncated details of which rule was triggered, along with many other relevant details will be included within the SMS message;

**execute(command):** specifying that a shell command should be executed when the alarm is triggered.

**none:** a special alarm type which performs no action. This alarm type is useful for grouped rules (signatures) - where component rules have no need of independent alarms - and also in certain rules that perform configuration or housekeeping activities.

Email and SMS alarms are specified by defining the alarm type, and any required additional information within parenthesis. For example, in order to generate an email alarm to be sent to the user 'someone@yourdomain.com' the alarm is written:

```
email(someone@yourdomain.com)
```

Multiple alarms may be associated with any rule, separated by commas (and white-space if required for readability) for example:

```
name :: rule type :: constraints :: -
email(someone@yourdomain.com), -
email(someone_else@yourdomain.com), -
sms(07900123456, 07900123462)
```

### 5.6.2 Meta Variables in Rules and Alarm Definitions

There are also a number of meta variables which can be used in rules and alarm definitions that will be substituted with values provided by the intrusion detection engine at run-time.

Substitutions of these variables with their run-time values occurs at the time of rule evaluation. They are specifically relevant to 'execute' alarms, and any file path specifications. The 'Rule Type' column refers to the rule types defined in the following section.



Variable	Rule Type	Description
%sv	All 'execute' alarms	The name of the computer that serverM is executing on
%cc	All 'execute' alarms	Current check-cycle count (always increments by 1 for each check cycle)
%rn	All 'execute' alarms	Current rule name
%ss	All 'execute' alarms	Seconds since midnight
%mn	All 'execute' alarms	Minutes since midnight
%hr	All 'execute' alarms	Hours since midnight
%dt	All 'execute' alarms	Date and time
%dnm	All 'execute' alarms	Day name
%dn	All 'execute' alarms	Day number
%mo	All 'execute' alarms	Month Name
%yr	All 'execute' alarms	Year (in 'yyyy' format)
%df	All 'execute' alarms	Date and time without spaces or formatting (example: 'Wed14Jul2004115450')
%path	All 'execute' alarms and File change/File size, and status rule bodies	Full home install path of serverM
%fn	File change and File size 'execute' alarms	File name of changed file
%sn	New service 'execute' alarms	Service name

**Figure 5.1: serverM Rule 'Meta Variables'**

### 5.6.3 The Order of Alarm Execution

There is no pre-defined order in which alarms are executed within a particular rule: care should be exercised when order of alarms is important (with 'execute' alarms which may affect each other for example). Alarm order can be enforced by repeating the rule definition (with unique names) and giving each copy of the rule different alarms because rules are evaluated in order of definition.

### 5.6.4 Using the 'def' Keyword

You may find yourself repeatedly using the same email addresses, SMS numbers, strings, or commands within a number of alarms definitions and other parts of the rule definitions. To simplify this, serverM provides a 'pre-definition' syntax so that you can define your own special variables within the rules file and use them repeatedly.

The keyword '*def:*' enables the user to define groups of email addresses, SMS numbers, and execution commands.

The ``def`` keyword can be placed anywhere in the rules file, but must occur before the pre-defined value is used within an alarm or rule definition. The syntax for a ``def`` value is:

```
def: type name (list)
```

Where ``type`` is one of ``email``, ``sms``, ``string`` or ``execute`` (case is not important); the ``name`` is any combination of alphanumeric characters with underlines (``_``), minus (``-``), periods (``.``), colons (``:``), and semi-colons (``;`); and the ``list`` is a list of any number of values separated by commas (e.g. email addresses, SMS numbers, or a command string).

Definition ``names`` must be unique to a type (but may be duplicated between types).

A command (within an ``execute`` type) must be enclosed in quotes e.g.:

```
def: execute somename ("find / -name *.pl | wc")
```

Only one execute command may be defined for any single ``def`` variable, but any number of email addresses or SMS numbers may be listed.

Examples of using ``def`` are:

```
# set up an execution command to write a simple
# heartbeat message to a file in the serverM home
# install directory

def: execute write-heartbeat ("echo heartbeat at %dt >> --
    '%path\heartbeat.txt'")

# set up a list of email recipients for email alarms

def: email administrators --
    (user1@yourdomain.com, user2@yourdomain.com)

# set up a list of SMS cell phone numbers to
# receive urgent alarms

def: sms administrators --
    (07901123456, 07901654321, 07901666555)
```

These definitions may now be used within a rule:

```
name::rule type::constraints:: -
    email(administrators), sms(administrators),
    execute(write-heartbeat)
```

All ``execute`` alarms *must* use a pre-defined value using the ``def`` keyword. Email alarms and SMS alarms may use pre-defined values, but may also have a **single** email address, or SMS number, embedded in the rule alarm definition itself. If serverM sees an ``@`` symbol in a rule alarm definition then it will assume that it is a specific email address and not a pre-defined value, and if it sees only numbers (and optionally the ``+`` symbol) in an SMS alarm definition then it will assume that it is an SMS number, and not a pre-defined value.

---

**Note:** pre-defined values always respect meta variables. For example:

```
def: execute copy-file ("cp %fn %fn_old")
```

will perform the file copy with the variable '%fn' replaced by the file-name triggering the rule at the time of rule evaluation.

---



---

**Note:** caution must be exercised when using the minus ('-') line separator within 'execute' alarms. The line separator and *all white-space* (spaces and tabs) either side of the separator are removed during parsing - this may cause problems with command line arguments. If this is the case then use the double minus ('—') line separator which will insert a single space replacing the double minus.

---



---

**Note:** definitions will propagate through rules files so that definitions made in one rules file will be available to rules files that are parsed later (see section 5.7 on scoping below).

---

The `def` keyword can be *scoped* to apply to all rules within the system, or rules that are defined within the current rules file only, or even to groups of rules within the current file. This is discussed in more detail in section 5.8 below.

### 5.6.5 Pre-defining Arbitrary Strings

The special `def` type of *'string'* can be used to define any string of characters or numbers which can then appear in any part of the rules file. A string definition enables any repeating parts of definitions, or any parts of rules that may change regularly, to be grouped into a convenient location.

A string definition is used by inserting the syntax ``${string-name}`` anywhere in the rules file, and the rules pre-parser will replace this specially formatted text with the defined value, e.g.:

```
def: string system-log ("/var/log/system.log")
def: string max-log-size ("10000k")

check system log size::file size->`${system-log}`:-
    >`${max-log-size}`::email(administrator)
```

Will rewrite to:

---

```
check system log size::file size->/var/log/system.log::-
>10000k::email(administrator)
```

Although for the sake of clarity it is not recommended, strings can even define themselves:

```
def: string mail-domain ("somewhere.com")
def: string my-email (me$$$somewhere.com)
def: email administrator ($$my-email$$)
```

### 5.6.6 Accessing Values from the Master Configuration File in Rule Definitions

Pre-defined email 'groups' are available to rule definitions in the form of square parenthesis. An alarm definition of the form:

```
email(some-name])
```

tells the rule parser to replace the square brackets, and the text within the brackets, with the value of the parameter 'some-name' from the serverM master config file. Thus if the master configuration file contains the line:

```
administrator:someone@somewhere.com
```

Then the alarm definition:

```
email([administrator])
```

will be replaced by the definition:

```
email(someone@somewhere.com)
```

within the appropriate rule.

### 5.6.7 Execute Alarms Overhead and System Delays ('waitfor:')

Execute alarms are extremely powerful alarm types within the serverM system - they enable the rules to proactively protect against intrusion by performing essential system commands in the event that suspicious behaviour is detected. However, in normal use, the execution of any commands are undertaken 'in-line' i.e. the serverM daemon will wait for the completion of any command before continuing with evaluating any following rules.

If an execute alarm results in executing a command that either fails to terminate, or which takes a considerable amount of time to complete, then the serverM daemon may be rendered ineffective until the command completes. To help with this issue, an additional keyword '*nowait:*' can be pre-pended to the command within the '*def:*' definition. Using this keyword tells the serverM daemon to fork an additional process in order to execute the command, and to continue without waiting for the command to terminate.

Caution should be exercised with this command. If a command is forked which does not terminate before the same rule triggers another execution of the command, a large number of processes may build up in the process table, and the serverM system may cause a significant resource utilization problem with the hosting server.

An example use of the *'nowait:'* keyword:

```
def: execute copy-log ("nowait: --  
    cp '%path/log/log.txt' '/tmp/oldlog'")
```

## 5.7 Rule Types and Constraints

There are five different rule types. The syntax for the rule types are given below along with examples.

### 5.7.1 File Change Rule

The 'file change' rule type checks each specified file during each check cycle to see if the content or access control list for that file has been changed. The constraints part of the rule specifies whether to check for ACL or content change or both.

Rule-type Syntax	<b>file change -&gt; &lt;file specification&gt;</b>
Details	<p>Where &lt;file-specification&gt; is either in individual filename, or a file pattern (i.e. using *nix globbing file pattern meta-characters). Recursion through globbing is controlled by the 'recurse' constraint described below.</p> <p>White space is not significant either side of the arrow but <i>is</i> significant within the filename.</p>
Example(s)	<pre>file change-&gt; /var/log/ftpd.log file change-&gt; /etc/*.conf</pre>
Constraints Syntax	<b>any combination of 'new' 'deleted' 'acl' 'content' 'not' and 'recurse:n'</b>
acl/acl-a	<p>'acl' specifies that the entire access control list for the file(s) should be checked for changes since the last check-cycle. The modification time, changed time, access time, and file 'mode' are checked.</p> <p>'acl-a' specifies that the entire access control list, <b>with the exception of access time</b>, for the file(s) should be checked for changes since the last check-cycle. The modification time, changed time, and file 'mode' are checked.</p>
recurse:n	If the <file specification> is a file pattern (e.g. contains '*' or '?' patterns etc.) then the 'recurse' constraint will recurse down the directory tree, applying the file pattern to each subdirectory found. The recurse depth is specified by the number <n>.
content	Specifies that the content of the file(s) should be checked for changes.

new	If a file which matches the <file specification> is created since the last check cycle then the alarm(s) will be triggered.
deleted	If a file which had previously been checked for content or acl changes, or recognised as a 'new' file, has been deleted then the alarm(s) will be triggered. This rule will only be triggered once for each deleted file.
(optional) not	Invert the 'acl' and 'content' constraints e.g. for 'not acl' then the alarm is triggered if the acl for the specified file(s) have not changed since the last check-cycle, etc. The 'not' keyword applies to all acl or content keywords.
Details	White-space is required between constraint keywords. Any amount of white-space may be used. If a file pattern is used, the rule will not recurse through sub directories.
Example(s)	content (check for content changes) not acl content (check to see if neither the content or acl have changed)

### Example file change rule.

```
# check config files for ACL changes

example file change:: -
    file change->/etc/*.conf:: -
    content acl::email(admin@yourdomain.com)

# check for deletions or new instances of
# scheduled daily tasks
scheduled task removal:: -
    file change->/etc/crontab/daily/*::deleted new::-
    email(admin@yourdomain.com)

# check serverM config file
serverM config file::file change->%path/config/config.txt::-
    acl content::-
    email(admin@yourdomain.com)
```

---

**Note:** the meta variable '%path' can be used within the file change rule type as part of the filename specification. It will be replaced at run-time with the install path for the serverM system.

---



### 5.7.2 Status Rule

The 'status' rule executes a user defined shell command and examines the output that would normally appear on STDOUT. If the output changes from check-cycle to check-cycle then an alarm is triggered.

This rule is the most powerful rule in the serverM system and enables users to alarm on arbitrary commands. For example, it is possible to 'grep' for a pattern in a group of files, and then generate an alarm if the output changes.

There is only one optional constraint – the 'baseline' option. This enables the serverM system to compare the current output with the output generated from the first time the command is run (rather than updating the comparison baseline with every new check).

serverM does not create any temporary files during comparison, but instead holds the results of the command output in memory. This is so that any file change rules are not affected by any status rules. It also means that each time the serverM system is stopped and started, the output from the command is refreshed. However, to ensure that the serverM engine does not become overwhelmed with too much output, if the status rule generates more than 1000 lines of output an error is generated and the rule is ignored.

The command that is executed must be predefined by a 'def: execute' definition.

Rule-type Syntax	<b>status &lt;command&gt;</b>
Details	<p>Where &lt;command&gt; is predefined with a 'def: execute' statement</p> <p>White space is not significant but there must be at least one space between the keyword 'status' and the command.</p>
Example(s)	<pre>def: execute failed-login ("grep 'failed to authenticate user' /var/log/secure.log")  status failed-login</pre>

Constraints Syntax	<b>'baseline' or 'none'</b>
(optional) baseline	Record the command output on the first time the rule is evaluated and compare all future output with this baseline.
Details	none
Example(s)	baseline

### Example Status Rule.

```
# test for whether user 'root' has logged in
# or logged out since serverM started

def: execute logged-in ("who -q | grep root")

root logged in::status logged-in::baseline:: -
    email(admin@yourdomain.com)
```

Because *'status'* rules execute arbitrary commands in order to compare output from one execution to another, it may be possible to cause significant delay to the serverM system if the command chosen takes a long time to complete execution. Command execution always takes place in-line i.e. the serverM daemon will wait for the status rule to complete before continuing with the evaluation of the next rule.

Unlike the execution of commands in the *'execute'* alarm type, there is no mechanism for forking the *'status'* rule command (see section 5.6.7 above). This is because the triggering of an alarm for a status rule needs to capture the output of the command for comparison.

Consider the following rule that checks for new SUID files anywhere within the mounted root directory or below:

```
def: execute setuid ("find / -type f -perm 4000 -print")

[severity 3] --
[initial] --
[on-day daily] --
[once until fail] --
    check setuid::status setuid::baseline::email([administrator])
```

This command causes the *'find'* tool to recurse down through every directory on the file system each time it executes. This can be an extremely time consuming activity - hence the use of the *'on-day'* pragma to limit the number of times the rule is evaluated.

### 5.7.3 New Service Rule

The 'new service' rule type checks for new processes (services) starting, and specific processes being present.

Rule-type Syntax	<b>new service</b>
Details	White space is not significant either side of the two keywords. Case is not significant.
Example(s)	new service

Constraints Syntax	<b>'none', 'lookfor:&lt;service pattern&gt;'</b>
none	Checks for <i>any</i> new service starting (i.e. in the 'running' state) since the previous check-cycle and triggers the rule if any new service is present
lookfor	<p>Tells the engine to look for this service and trigger the rule if the process is not currently 'running'. The service pattern may be part of a name, and will match all services whose names contain the pattern given.</p> <p>The pattern is tested against the output of the 'ps' command with the options 'awx', and <i>any</i> part of the process name (including the path to the executable) will result in the pattern being found.</p> <p>The pattern can be any string of text, and can make use of the Perl regular expression syntax.</p>
Details	White space is significant only within a service name. Case is significant only within a service name.
Example(s)	lookfor:/usr/sbin/cron (check for the running state of the cron scheduling service)

**Example New Service Rule.**

```
# check that the cron service is running

example new service rule::new service:: -
  lookfor:/usr/bin/cron:: -
  email(user1@yourdomain.com)
```

### 5.7.4 File Size Rule

The 'file size' rule type checks a file (or files) to see if they have exceeded a specified size. The alarm is triggered if the file is larger than the limit specified.

Rule-type Syntax	<b>file size -&gt; &lt;file specification&gt;</b>
Details	<p>Where &lt;file-specification&gt; is either an individual filename (specified from the drive root), or a file pattern (i.e. using bash style file globbing)</p> <p>White space is not significant either side of the arrow but <i>is</i> significant within the filename. Case is not significant.</p>
Example(s)	<p>file size-&gt; /var/log/secure.log</p> <p>file size-&gt; /var/log/*.log</p>
Constraints Syntax	<b>&gt;&lt;number&gt;k</b>
Details	<p>White space is significant. The 'greater than' symbol is required. The 'k' symbol is optional. The file size check uses the limit 'number' (of Kb in size) to determine whether or not to trigger the rule. The size must be <i>greater than</i> the size given for the rule to trigger.</p>
Example(s)	>1000k (greater than 1000 Kb)

#### Example File Size Rule.

```
# check that the httpd logs are not larger than 100Mbs each
# and copy them to a larger partition if they are,
# then delete the original

example file size rule:: -
    file size->/usr/local/apache2/logs/*.log:: -
    >100000k:: -
    execute(rotate-httpd_logs)
```

---

**Note:** the meta variable %path can be used within the file size rule type as part of the filename specification. It will be replaced at run-time with the install path for the serverM system.

---

### 5.7.5 True Rule

The 'true' rule is an artificial rule type that triggers every check-cycle (unless modified by a rule evaluation pragma). It is often used to create 'heartbeats' or perform repetitive tasks within the system (examples are given below).

Rule-type Syntax	<b>true</b>
Details	Case is not significant.
Example(s)	true

There are no constraints given for 'true' rules, the keyword 'none' is used as a placeholder in the constraints part.

#### Example True Rule.

```
# write a simple heartbeat to a nominated file
# every 100th check-cycle

[every 100] --
    example true rule::true::none::execute(write-heartbeat)
```

### 5.7.6 The 'ignore' Keyword

If rules are written that can capture a large number of similar symptoms, but some of which are *acceptable* within your security policy, then the 'ignore' keyword can be used to remove any commonly occurring false positive alarms. For example, the 'new service' rule is indiscriminate in that it will alarm on *any* new process, including processes that you would expect to stop and start without concern.

The 'ignore' keyword tells the detection engine to ignore certain values when evaluating a rule, such as certain processes, or files (for file changes rules and file size check rules).

The syntax of the ignore statement is:

```
ignore: <type> "<value>"
```

where 'type' is one of:

- **filesize:** indicates that the file specified by <value> is to be ignored in any 'file size' rule;
- **filechange:** indicates that the file specified by <value> is to be ignored in any 'file change' rule;
- **service:** indicates that any service matching the name specified by <value> should be ignored in any 'new service' rule;

#### Examples

```
# ignore the 'bash' and 'pgp' services
ignore: service "-bash"
ignore: service "pgp-agent"
```

```
# ignore the temp log file for file size and changes
ignore: filechange "/tmp/log.txt"
ignore: filesize "/tmp/log.txt"
```

Like that 'def' keyword, the *scope* of the 'ignore' keyword can be modified to relate to all rules throughout all rules files, or to a specific set of rules within a file. This is explained in more detail in section 5.8.



## 5.8 Scope and Signatures

### 5.8.1 Introduction and Global Scope

By default, a `'def'` or `'ignore'` statement will affect any rules that are parsed after the position in the rules file where the statements occur. Unless you tell the detection engine otherwise, `'def'` statements and `'ignore'` statements are **local** to the rules file within which they are defined, from their point of definition, and will not affect rules defined in other rules files. They are said to be in **local scope**.

This is also true of rule names, where duplicate rule names within the same rules file will give rise to a parsing error, and the final duplicate named rule to be assumed to be the 'live' one (and an error message will be generated). Rules in different rules files can be identical however (the parser automatically pre-pends the rules filename to the rule name to avoid clashes).

However, there are ways of expanding or contracting the scope of rule names, `'def'` statements, and `'ignore'` statements.

There are three kinds of scope to consider:

- **global scope:** by using the keyword `'global'` at the start of a `'def'` or `'ignore'` statement, the statement will affect *all* rules that the detection engine uses, regardless of which file the rules are defined in. Globally scoped `'ignore'` statements even affect rules which are defined in rules files which *precede* the file in which the global statements occur (although global `'def'` statements only affect rule definitions from the point of definition onwards);
- **local scope:** by using no additional keyword, or by not defining within a signature scope, definitions are by default *local* to the current rules file only;
- **signature scope:** by using *signature scoping* (see below), `'def'` and `'ignore'` statements can be limited to only affect those rules which are defined within a signature block, and rule names become local to the signature scope.

#### Examples of Global Scope

```
# define an email group 'admins' to use
# throughout all of the rules files

global def: email admins (admin1@domain.com, --
                        admin2@domain.com)

# always ignore the WWW service
global ignore: service "httpd"
```

### 5.8.2 Signatures and Scoping

serverM rule files allow for the grouping of rules into *signatures* i.e. sets of rules with optional *'def'* and *'ignore'* statements that affect only those rules defined within the signature.

To group rules together into signatures, use opening and closing braces ('{' and '}') together with a scope name. The syntax for signature scoping is:

```
{ <scope name>
    definitions, ignores, and rules...
    ...
}
```

The <scope name> can be any sequence of characters except the closing scope character ('}'), and preceding and trailing white space is ignored.

The opening brace (along with the scope name) must be on a line by itself (any white space is ignored). The closing brace may be at the end of any line, or on a line by itself (with any white space allowed).

Using a scope affects the parsing and definition of rules in two beneficial ways: they are significant in the way in which the uniqueness of *'def'* group names and rule names are checked, and they also provide a clear indication of the signature to which a rule belongs when rule names are reported both in the log files and also in any alarm messages that are sent.

For *'def'* and *'ignore'* statements, using a scope makes the statement definitions unique to the signature by associating the scope name with the defined groups or ignore values. This means that you can use *'def'* group names within a scope without fear of the name clashing with existing *'def'* statements in any other files. If you do not use a scope, then non-global *'defs'* and *'ignores'* will remain in *local* scope.

For rule names, the scope name is pre-pended to the rule names at the time that they are defined, guaranteeing the uniqueness of the rule name. Together with the automatic pre-pending of the filename containing the rule to the rule name, rule names in signatures are guaranteed to be unique.

An example of using signature scoping is given in the signature definition below. In this example, the scope name is "suspicious configuration changes", and because a scope is used, the email group *'admin'* is not checked for uniqueness, and will override any locally scoped, or even globally scoped, email group with the same name.

Any rule definitions before, or following, this signature will not have access to the *'admin'* group defined within this signature, and the *'filechange'* ignore statement will have no affect on any other rule.

```
# check to see if any conf files have changed in
# the /etc directory - we will ignore pserv.conf
# because we expect, and accept, regular changes

{ suspicious configuration changes

    ignore: filechange "/etc/pserv.conf"

    def: email admin (admin2@yourdomain.com)

    configuration change::-
        filechange->/etc/*.conf::-
            email(admin)

}
```

If the *'ignore'* or *'def'* statements had been preceded with a *'global'* keyword then they would be treated as any global statement, and would affect all rules within all rules files (global definitions do not respect scope boundaries). A summary of the affect of scoping on *'def'* and *'ignore'* statements is given in the table below.

Statement	Scope	Affects
global def:	global	All rules defined within all rules files will have access to this definition. Must be unique between other local defs and global defs.
global ignore:	global	All rules defined within all rules files will have access to this definition. Must be unique between other local ignores and global ignores (note: even affects rules defined <i>before</i> global ignore is defined!)
def:	local	All rules defined within the current rules file will have access to this definition. Must be unique between other local defs and global defs.
ignore:	local	All rules defined within the current rules file will have access to this definition. Must be unique between other local ignores and global ignores.
{... def:	signature	All rules defined within the current signature scope will have access to this definition. Must be unique within the signature scope. No need not be unique between other defs - local or global. Overrides other <i>local</i> or <i>global</i> defs.

{... ignore	signature	All rules defined within the current signature scope will have access to this definition. Must be unique within the signature scope. No need not be unique between other ignores - local or global.
-------------	-----------	---

**Figure 5.3: 'def' and 'ignore' Scoping Rules**

## 5.9 Alarms and Wildcard File Specifications

When a file size rule, or file change rule, is specified with wildcards in the file specification, the engine treats the rule as a rule 'template' and, *logically*, expands the rule for each file within the specification. The engine does not create a new name for each expanded rule, but *will* create a new entry in any alarm reports for each file triggering the rule, and will execute the command in any 'execute' alarm each time for each file for which the constraints are met (performing any substitutions of meta variables on a file by file basis).

Thus, a simple rule which copies a changed file to a new location e.g.

```
def: execute make-copy ("cp %fn /tmp/%fn_%dt")

keep modified files::file change->/var/log/*.log:: -
    acl::-
        execute(make-copy)
```

will make a copy of each log file whose access control list has changed since the last check-cycle (overwriting old copies if necessary).

Similarly, when a rule's constraints are met by a number of instances of data e.g. services in a 'new service' rule, an alarm entry will be generated for each instance. Thus a rule which checks for an process occurring will generate an entry in any alarm type for each occurrence of that process appearing in the current process list.

## 5.10 Example Rules

The following rules show many of the facilities listed above, and demonstrate some types of intrusion that may be detected.

### Example 1: Writing a simple heartbeat to a text file

The following rule writes a single line of text to a text file located in the installation directory of the serverM system on each check cycle. It uses the 'true' rule type to ensure that it is triggered every cycle, and the substitution variables to provide some information within each line.

```
def: execute write-heartbeat --
    ("echo 'heartbeat at %dt' >> '%path/heartbeat.txt'")

write heartbeat::true::none::-
    execute(write-heartbeat)
```

### Example 2: Warning of repeated failed attempts to login

The following two rules use the 'scour' tool (provided with the serverM distribution – see Appendix F) to check the system log file for 'unauthorised access' messages. The 'scour' tool checks for a pattern within the lines of a log file and can return a range of output types depending on the flags set. In the example below, a total of at least 6 failed login attempts need to have occurred within the previous hour for an alarm to be triggered.

```
# note that the syslog location, and access message format
# is specific to Mac OS X. Linux definitions require editing
# of these definitions

def: execute find-failed-logins -
    ("%path/scour -yesno num:6 file:/var/log/system.log -
    previous:01:00:00 pattern:'Unauthorised access'")

failed logon email::status find-failed-logins:: -
    none:: -
    email(user1@yourdomain.com)
```

This rule turns out to be rather simplistic. In most installations it is possible to run a timed process that compresses log files when they reach a certain size. In a default Linux install for example, it is not uncommon for the system log to be compressed with a 'zip' command, and have an index number appended to the file name of the compressed log. The numbers are rotated so that the current log file becomes compressed file '0', the existing compressed file '0' becomes compressed file '1', and so on.

In order to cater for the fact that log rotation might have occurred since the last check-cycle, or that failed authorisation attempts have occurred that span two log files, the rule has to be slightly more complex.

The 'scour' tool enables the user to replace the file that is being searched with the output of a command. Thus we can define a rule that looks at both the current log file, and the unzipped previous log file 'on the fly'. The rule is now defined as:

```
# note that the syslog location, and access message format
# is specific to Mac OS X. Linux definitions require editing
# of these definitions

def: execute find-failed-logins -
    ("%path/scour -yesno num:6 file'!gunzip -cf --
    /var/log/system.log /var/log/system.log.0.gz' --
    previous:01:00:00 pattern:'Unauthorised access'")

failed logon email::status find-failed-logins:: -
    none:: -
    email(user1@yourdomain.com)
```

In this example rule the 'scour' option '!' is used before the 'gunzip' command which forces the output of the command following the '!' to be executed and the output to be used for searching rather than the files themselves. The '-cf' options will force the 'gunzip' command to list the uncompressed file unchanged, and forces all output to the standard output.

Linux distros often use different log file names. By default, Debian will not compress the first backup of the syslog file, but only the second onwards. The name of the syslog file is also different.

We can use the conditional parsing syntax to create a rule which will work either on Mac OS X, or on Debian Linux as follows:

```
ifdef MAC OS X
def: execute find-failed-logins -
    ("%path/scour -yesno num:6 file'!gunzip -cf --
    /var/log/system.log /var/log/system.log.0.gz' --
    previous:01:00:00 pattern:'Unauthorised access'")
enddef

ifdef Debian
def: execute find-failed-logins -
    ("%path/scour -yesno num:6 file'!cat --
    /var/log/syslog /var/log/syslog.0' --
    previous:01:00:00 pattern:'Unauthorised access'")
enddef

failed logon email::status find-failed-logins:: -
    none:: -
    email(user1@yourdomain.com)
```

Only the definition statement that defines the log location and the command to list the system log files is wrapped in the conditional parsing statements in this instance as the rule body is valid for both platforms (see section 5.13 for an explanation of conditional parsing).

**Example 3: Checking Processes**

The following signature checks for any processes starting up at unusual times (between 8pm and 6am on any day), excluding the 'backup' and 'htmreport.pl' processes.

```
# any service that starts during unusual hours we are
# interested in

{ new service at night

    ignore: service "/usr/sbin/backup"
    ignore: service "%path/htmlreport.pl"

    [severity 10]--
    [at-time 2000:2400, 0000:0600] --
        new service at unusual time::new service::none::-
            email(administrator)
}
```

**Example 4: Auto-Archiving the Log File**

The serverM log file can grow very large under some circumstances (many rule triggers for example, or a file change checking rule that checks for large numbers of files etc.), but it is possible to automatically archive the current log file when it gets to a certain size. This rule archives the file to a specific directory, using the current date and time as the filename (without any additional formatting to ensure no spaces in the filename).

```
def: execute archive-log --
    ("cp '%fn' '/tmp/serverMlog%df.txt' ; rm '%fn'")

[every 100] auto archive serverM log:: --
    file size->%path/log/log.txt::>1000k:: -
    execute(archive-log)
```

Note that this rule will work correctly regardless of where the serverM system was installed due to the use of the '%path' variable. The semicolon in the execution string enables the rule to execute more than one command in the appropriate order (bash shell compliant). Also, the use of the '%df' variable will give rise to a unique filename every time the rule is triggered (it will be replaced by a compressed form of the current date and time stamp which includes seconds).

The *[every 100]* pragma also provides a degree of efficiency: in most cases, the log file will grow very slowly, and checking for the file size on every check-cycle would be unnecessary.



**Example 5: Looking for 'Rootkit' exploits**

Some common \*nix root kits attempt to gain access to a server by altering standard bash startup scripts and user profiles, especially those belonging to 'root' user. Failure to detect such changes can leave servers with backdoor access for some time. The following rule checks for changes to these vital files.

```
# check for root user bashrc and profile changes
{ root profile changes

    [severity 3] --
    [every 20] --
    root user profile change::file change -
        ->/root/.profile::content::-
        email(administrator)

    [severity 3] --
    [every 20] --
    root user bashrc change::file change ->
        /root/.bashrc::content::-
        email(administrator)
}
```

**Example 6: Bespoke Alarm Messages**

If the serverM system is running on a networked server, and the administrator requires alarms to be sent using the 'tell' functionality of \*nix (a simple remote console messaging system), then an execute alarm can be set with an appropriate message.

```
def: execute net-tell-truncate --
    ("/sbin/tell alanf@remoteip.net --
        serverM log is being truncated on server %sv")

auto archive serverM log net say:: -
    file size->%path/log/log.txt::>1000k:: -
        execute(net-tell-truncate)
```

Similarly, assuming an SMTP server is running on the local server, then an email message could be sent.

For more examples of rule definitions, see section 7 which discusses the standard rule definition files included with the serverM distribution.

## 5.11 Writing New Rules and Using 'Off The Shelf' Rules

During the installation process the serverM system installs a very basic set of rules into the configuration directory. The file `'rules.txt'` in the directory `%path/config/rules.txt` is where the server engine expects to see the rules file, and any new rules can be added to this file as required. The serverM detection engine will check for changes to this file at the start of every check cycle, and will accommodate new rules, or rule changes accordingly.

On reading a rules file, the serverM engine will log the fact that the rules file has changed, and also log the names of those rules which have been found during the parsing of the file. Any syntax errors found in the rules file will also be logged and a warning email sent to the nominated administrator recipient (if configured). In the event of there being syntactic errors in the rules file, the serverM engine will attempt to read and install as many well-formed rules as possible so as to provide continued protection.

You can also place any additional rules files into the directory found in `'config/ruleslib/'`. This may be useful if you have rules files specific to a particular server (i.e. *local* rules files) but want to share a single main rules file between multiple serverM installs.

---

**Note:** if you install additional rules files into the `'config/ruleslib/'` directory the serverM system will read these files *after* the main rules file, and will evaluate these rules in the order in which they appear within each rules file. Rules files are read from this directory in alphabetical order of file name. For example `'_plib001.txt'` will be read (and the rules evaluated) before `'newrules.txt'`.

---

## 5.12 Good Rule Writing 'Style'

### 5.12.1 Overview

The rules language for serverM is very powerful, and extremely flexible; as a result it is possible to write the same rules in a number of ways. Various 'good practice' tips have emerged from the user base that are summarised here.

### 5.12.2 Rule Efficiency

File patterns in file change and file size rule types can give rise to very large processing overheads if large directory structures are included. For example, it is possible to check for any file changes in the root mount point on a \*nix server system, to a depth of 20 directories, with a simple rule:

```
check system-wide file changes:: -
  file change->/*::-
  content recurse:20::-
  email(administrators)
```

However, this rule will typically cause the detection engine to check the contents of up to 100,000 files or more. Each file is checked by reading the contents and comparing a hash with the previously recorded hash. Although the serverM system will undertake this level of file checking quite happily, the time taken to perform each check-cycle, and the overhead required, may be very large.

It is better to consider which files you are specifically interested in, and concentrate on those. For example, if you are concerned about applications (such as system commands etc.) then a more fine-grained rule may suffice:

```
check bin changes::-
  file change->/bin/*::-
  content acl recurse:2::-
  email(administrators)

check usr bin changes::-
  file change->/usr/bin/*::-
  content acl recurse:2::-
  email(administrators)

etc...
```

This will considerably lower resource utilisation by the daemon.

Similarly, if you are using execute alarms, care should be taken to ensure that the commands that are executed are relatively lightweight. The detection engine will wait until command completion before continuing with the check-cycle, and significant pauses may result (although it is possible to fork execution - see section 5.6.7 above).

An alternative to reducing the number of files checked is to include an `[every n]` pragma, and check on a less frequent basis, e.g.

```
[every 1200] check bin changes::-
    file change->/bin/*::-
    content acl recurse:5:-
    email(administrators)
```

### 5.12.3 Alarm Flooding

The serverM system provides facilities to avoid alarm flooding by using the throttle settings in the master configuration file. However, this is a very blunt tool, and can cause vital alarms to be missed if not used carefully. The rule pragmas are a better way of dealing with potential alarm flooding, especially with the use of the *'clear-on-pause'* setting in the master configuration.

Consider the following rule, which detects failed logon attempts (see section 5.9 above for a definition of the *'failed-logins'* command):

```
failed logons::-
    status failed-logins::-
    email(administrator)
```

If an attacker were to use a brute-force hacking script to attempt random username and password combinations, then this would give rise to continued alarms every check-cycle. By using the *'once until fail'* pragma, the first check-cycle in which this rule was triggered would give rise to an alarm, but then no more alarms would be sent until the a check-cycle occurred in which there were no new events to trigger the rule i.e. until the attack which gave rise to the alarm had stopped for a short while. This would give rise to the rule:

```
[once until fail] failed logons::-
    status failed-logins::-
    email(administrator)
```

Alternatively, throttling can be achieved by using the `[every n]` pragma, but specific care must be taken in the way in which the engine tests, and then discards, historic data.

### 5.12.4 Overlapping Rules

It is common to write rules which *'overlap'* i.e. whereby one rule is a subset of the trigger conditions of another. The following two rules give an example:

```
failed logons::
    file change->/etc/httpd.conf::-
    execute(delete-mutex)

failed logons brute-force::
    file change->/etc/*.conf::-
    execute(set-mutex)
```

In this pair of rules, the second would always be triggered whenever the first rule is triggered. This *may* be desired behaviour, especially if the alarms differ, but it is advisable to consider overlaps carefully, especially if the alarms execute commands that contradict, or depend upon files which one command deletes as the rule is triggered (as in this example).

#### 5.12.5 Over-use of True Rules with Alarms

'true' rules are extremely useful, especially in combination with rule pragmas, and enable the system to perform extremely flexible housekeeping functions. However, because they trigger each time a check-cycle occurs, injudicious use of these rules can give rise to repeated alarms. It is recommended that 'true' rules are given either alarm type 'none', or 'execute' with simple commands to execute. Alternatively, true rules are made safer by using the [every *n*] pragma.

Even with 'execute' alarm settings on these rules, unforeseen behaviour can occur:

Consider the following rule:

```
heartbeat message::
  true::none::-
    execute(udp-send-heartbeat-message)
```

This rule will generate an alarm message to be sent by the 'udp-send' network service every check-cycle, perhaps as often as once every few seconds. The only way to stop these alarms occurring would then be to edit the rules file or stop the watcher daemon.

### 5.13 Conditional Parsing

It may be useful to include conditional parsing to the rules file. This may assist in writing portable rules files that could be used on multiple platforms. The keywords `'ifdef'` and `'enddef'` provide scope blocks for conditional rules parsing.

If the rules parser encounters a line containing (only):

```
ifdef <value>
```

and the master configuration file contains the `'user-def'` parameter definition:

```
user-def: <value>
```

then the rules parser will continue parsing the lines between the `'ifdef'` line and the next `'enddef'` line. If, however, the master configuration file does not contain the appropriate `'user-def'` definition, then all lines between the `'ifdef'` and `'enddef'` will be ignored.

The most common reason to use conditional parsing is for platform-specific rule definitions.

Note that conditional parsing will not enable *parts* of rules to be included in the parsing, and they must surround whole rule definitions. They *can* be used for surrounding `'def'` statements, which means that fine grained control of parsing is possible.

### 5.14 Using 'htmlreports' To Check Rule Definitions

serverM provides the capability of reporting on the current rules files and the rules definitions, and highlighting any rules file errors, by generating HTML web pages detailing the results of parsing the rules files. The `'htmlreports'` tool generates static HTML report pages on demand (see Appendix F for details).

An example rules file analysis report page generated by the `htmlreport` tool is given below. It shows the rules that were read from the current rules files (in order that they will be evaluated), and rules that contain errors (along with the detailed error report), current `'ignore'` definitions (local and global) and scope details for each rule.

The `htmlreports` tool can be executed automatically as the result of a rule evaluation by serverM itself, and, if a local web server is running, provide remote access to a report on the current system status.

serverM installation report

Home

Log File Analysis

Rules Analysis

Configuration

serverM website

serverM system rule parsing report

Server: Bastille.local

Date page created: Tue 28 Feb 2006 : 12:56:50

Protected by serverM

Rules parsed successfully (in order of evaluation)

check passwd file	File: rules.txt
check shadow file	File: rules.txt
{failed login attempts} check failed login	File: rules.txt
check config files	File: rules.txt
login at unusual time	File: rules.txt
cron is running	File: rules.txt
syslog is running	File: rules.txt
{new service at night} new service at unusual time	File: rules.txt
{root profile changes} root user profile change	File: rules.txt
{root profile changes} root user bashrc change	File: rules.txt
serverm config file changes	File: rules.txt
serverm library file changes	File: rules.txt
serverm core file changes	File: rules.txt
{ftp server access} late night ftp access	File: ruleslib/ftptlog
{ftp server access} weekend ftp access	File: ruleslib/ftptlog
{generate html report} generate html report if rules change	File: ruleslib/genreport
{generate html report} generate html report if ruleslib change	File: ruleslib/genreport
{generate html report} generate html report if config change	File: ruleslib/genreport
{generate html report} generate all report	File: ruleslib/genreport
{generate html report} generate new log report	File: ruleslib/genreport
{generate html report} regular make report	File: ruleslib/genreport

Rules parsed unsuccessfully

No rules parsed unsuccessfully

All Errors in Rules Files

No Errors in Rules Files

Global ignores

No global ignores

Name:	check passwd file	File:	\Users\david\Documents\opensource\serverM\rules.txt	Scope:	N/A
Definition	filechange -> \etc\passwd:;-acl content				

Figure 5.4: The htmlreports Rules Analysis Page

## 6 Report Formats and Alarm Messages

---

During normal operation, the serverM system may be configured to send a variety of alarms and reports via email and SMS. The format of these messages are detailed and examples given.

---



## 6.1 Introduction

A standard install of the serverM system is capable of sending three types of email messages, and one type of SMS message:

### Email messages:

- email alarm messages - generated by the 'email' alarm type, and sent to the user specified email accounts defined in the rules file;
- daily report messages - generated by the serverM watcher service, and sent to the 'daily-report-email[1,2,3]' email accounts at a time specified in the 'daily-report-time' configuration parameter (see section 4);
- administrator messages - including service run state change notifications and error messages, sent to the 'administrator' email account configuration parameter (see section 4).

### SMS messages:

- SMS alarm messages; generated by the 'SMS' alarm type, and sent to the user specified SMS cell phone numbers defined in the rules file;

The content and layout of these messages are determined by the serverM watcher service and are not configurable (with the exception that email alarm messages and the daily report may be sent in plain text or HTML format according to the relevant parameters set in the master configuration file).

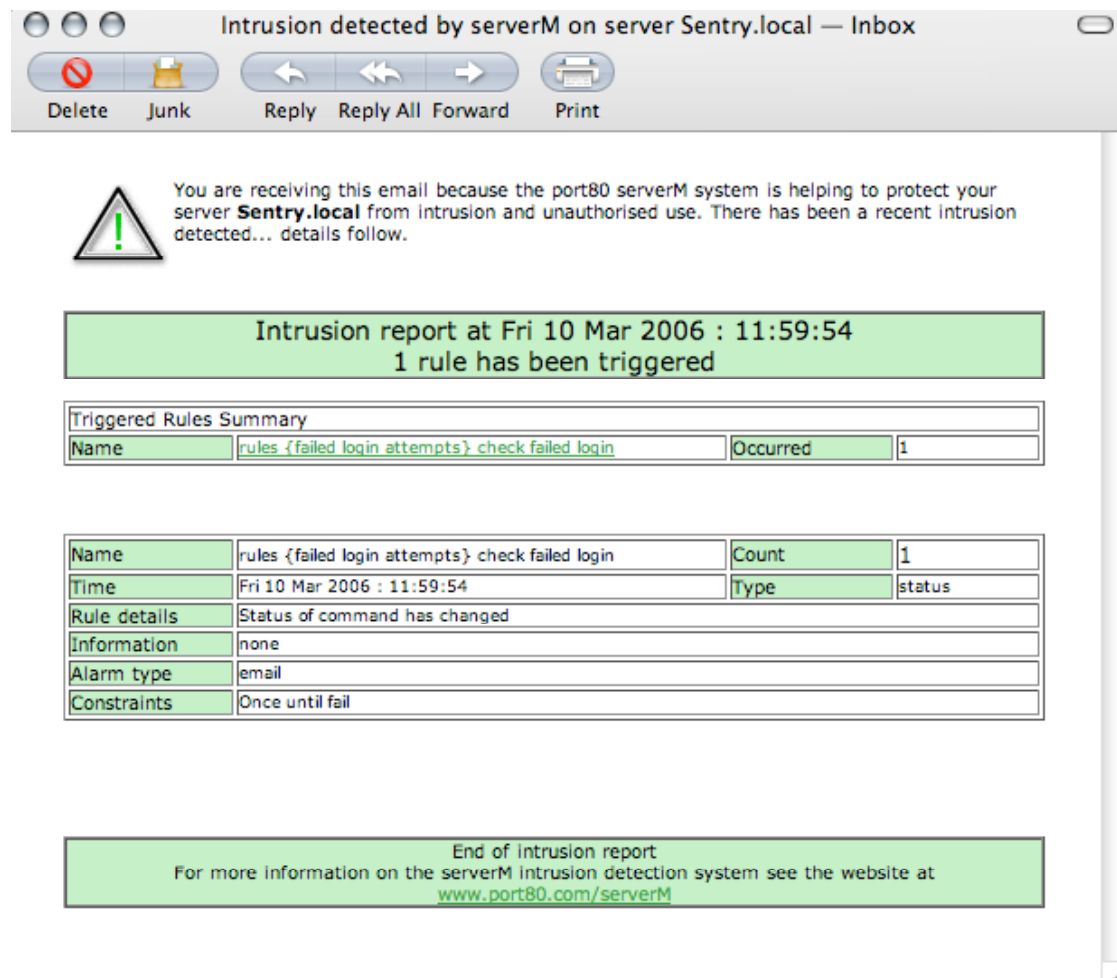
Below, we detail the standard contents of these messages and the parameters which affect each one.

## 6.2 Email Alarm Messages

### 6.2.1 Overview

If a detection rule has an associated 'email' alarm type, then at the end of each check-cycle, if the rule alarm has been triggered then an email alarm message will be sent to the email addresses given in the alarm definition.

For each email recipient who is configured to receive an email alarm message at the end of the check-cycle, a summary of all rules which have been triggered during that cycle is given.



**Figure 6.1: An Example Email Alarm Message (HTML Format)**

### 6.2.2 Email Alarm Message Contents

Whether in HTML or plain text format, the email alarm message will contain the following information:

- a short introductory paragraph explaining why the recipient has received the message and the server name that the message has originated from;
- a report banner with the timestamp of the sending of the message and the number of rule alarms that have been triggered;
- a summary of the names of each rule name that has triggered an alarm, and the count of how many times that rule alarm has been triggered during the check-cycle;
- for each unique rule that has had an alarm triggered, a summary containing information on the rule name, the *count* showing how many times the rule alarm has been triggered since the start-up of the serverM watcher service (this count is regardless of the alarm triggering, it refers to the *rule* triggering – see section 5.3.1. on the distinction between rule triggering and alarm triggering), the rule type, the parameters of the rule (rule constraints and any other relevant details that may be useful for forensics), the alarm type, and any alarm triggering pragmas (see section 5.3);
- an 'end of email' banner to confirm that the whole email has been sent correctly and not truncated in any way.

For HTML formatted email alarm messages, the summary table at the start of the message will be hyper-linked to the first occurrence of each rule detailed within the email.

---

**Note:** for each unique rule detailed within the email alarm message, there will be a maximum of the first 5 alarm reports – this is to ensure that the email alarm message remains within sensible size limits. However, the count will reveal any further alarm triggers.

---

### 6.2.3 Email Alarm Message Configuration

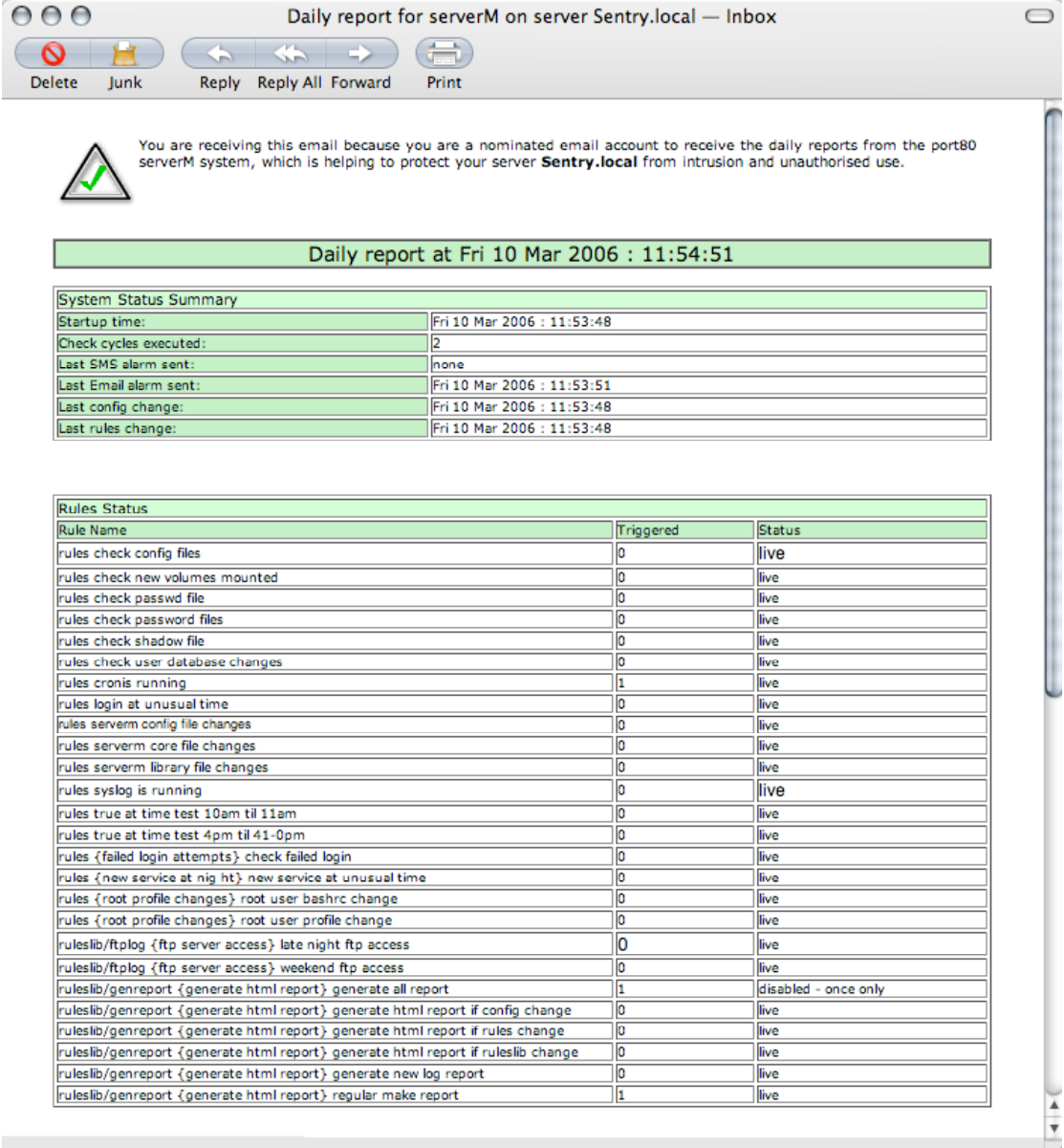
The parameters that affect the email alarm messages are:

- the master configuration file parameter '*alarm-emails*' which determines whether or not email alarms are sent;
- the master configuration file parameter '*alarm-email-throttle*' which determines whether more than one email alarm is sent in any 24-hour period (measured from midnight local time);
- the master configuration file parameter '*smtp-reply*' which determines the reply-to address and the envelope-sender address of the email;
- the master configuration file parameter '*email-gateway*' which determines the IP or domain of the SMTP relay server through which all serverM emails are sent.
- the master configuration file parameter '*alarm-format*' which determines whether the email alarm message is sent in plain text or HTML.

## 6.3 Daily Report Messages


### 6.3.1 Overview

The serverM system can be configured to send a daily report which summarises the current status of the system and includes details on which rules have been triggered during that past 24 hour period. The contents of the report are independent of any email alarm message settings, or other alarm settings, and provides a mechanism for detecting alarms for those rules which have had their alarms 'de-activated' by pragmas or message throttling. Each email address defined by the 'daily-report-email[1,2,3]' parameters in the master configuration file will receive an email in plain text or HTML format which is sent at approximately the time specified by the 'daily-report-time' parameter.



Daily report for serverM on server Sentry.local — Inbox

Delete Junk Reply Reply All Forward Print

 You are receiving this email because you are a nominated email account to receive the daily reports from the port80 serverM system, which is helping to protect your server **Sentry.local** from intrusion and unauthorised use.

**Daily report at Fri 10 Mar 2006 : 11:54:51**

System Status Summary	
Startup time:	Fri 10 Mar 2006 : 11:53:48
Check cycles executed:	2
Last SMS alarm sent:	none
Last Email alarm sent:	Fri 10 Mar 2006 : 11:53:51
Last config change:	Fri 10 Mar 2006 : 11:53:48
Last rules change:	Fri 10 Mar 2006 : 11:53:48

Rules Status		
Rule Name	Triggered	Status
rules check config files	0	live
rules check new volumes mounted	0	live
rules check passwd file	0	live
rules check password files	0	live
rules check shadow file	0	live
rules check user database changes	0	live
rules cronis running	1	live
rules login at unusual time	0	live
rules serverm config file changes	0	live
rules serverm core file changes	0	live
rules serverm library file changes	0	live
rules syslog is running	0	live
rules true at time test 10am til 11am	0	live
rules true at time test 4pm til 41-0pm	0	live
rules {failed login attempts} check failed login	0	live
rules {new service at nig ht} new service at unusual time	0	live
rules {root profile changes} root user bashrc change	0	live
rules {root profile changes} root user profile change	0	live
ruleslib/ftplog {ftp server access} late night ftp access	0	live
ruleslib/ftplog {ftp server access} weekend ftp access	0	live
ruleslib/genreport {generate html report} generate all report	1	disabled - once only
ruleslib/genreport {generate html report} generate html report if config change	0	live
ruleslib/genreport {generate html report} generate html report if rules change	0	live
ruleslib/genreport {generate html report} generate html report if ruleslib change	0	live
ruleslib/genreport {generate html report} generate new log report	0	live
ruleslib/genreport {generate html report} regular make report	1	live

**Figure 7.2: An Example Daily Report Message (HTML Format)**

### 6.3.2 Daily Report Message Contents

Whether in HTML or plain text format, the daily report message will contain the following information:

- a short introductory paragraph explaining why the recipient has received the message, and the computer name that the message has originated from;
- a report banner with the timestamp of the sending of the message;
- a system status summary showing information on when the service was started, the current check-cycle count, the times of the last SMS and email alarm messages to be sent, and the times of the last changes to the master configuration file and rules file;
- a summary of the current rules status which details which rules are currently defined by the system, how many times they have been triggered since system startup, and the current rule status (whether they are live or not depending on the evaluation pragmas);
- a summary of the current configuration parameters from the master configuration file.

### 6.3.3 Daily Report Message Configuration

The parameters that affect the daily report messages are (see section 4 for details of the master configuration file parameters):

- the master configuration file parameter `'daily-report'` which determines whether or not daily reports are sent;
- the master configuration file parameter `'daily-report-time'` which determines when the daily report email should be sent (measured from midnight local time);
- the master configuration file parameters `'daily-report-email[1,2,3]'` which determine the recipients of the daily report email;
- the master configuration file parameter `'smtp-reply'` which determines the reply-to address and the envelope-sender address of the email;
- the master configuration file parameter `'report-format'` which determines whether the daily report email is sent in plain text or HTML.

## 6.4 Administrator Email Messages

### 6.4.1 Overview

If the 'administrator' parameter is set in the master configuration file (see section 4 for details of the master configuration file parameters) then a number of plain text email messages will be sent to the nominated email account under a number of conditions:

- the run state of the serverM watcher daemon has changed;
- an error has occurred during the running of the serverM watcher daemon.

The 'run state' administrator email messages provide the system administrator with details of a warning in the case that the watcher service stops or pauses for any reason (due to a system error or malicious intent). These emails are generated when the service stops, pauses, or continues.

The 'error' administrator email provides the system administrator notification of any errors in configuration or system operation without having to manually check the serverM log file for error messages.

### 6.4.2 Administrator Email Message Contents

Administrator email messages for service run state changes will contain the following information:

- the server on which the serverM system is running;
- the timestamp of the run state change;
- which run state the service is entering.

Administrator email message contents for errors will contain the following information:

- the server on which the serverM system is running;
- the timestamp of the run state change;
- a detailed error message (or messages).

---

**Note:** administrator error emails will give details for all errors encountered during rule evaluation for a complete check-cycle rather than on an error-by-error basis. All other error message emails will be sent at the time of the error occurring.

---



---

**Note:** all error messages will be written to the serverM log regardless of whether or not administrator email messages are sent.

---

### 6.4.3 Administrator Email Message Configuration

The parameters that affect the administrator email messages are (see section 4 for details of the master configuration file parameters):

- the master configuration file parameter *'administrator'* which determines the recipient of the emails;
- the master configuration file parameter *'smtp-reply'* which determines the reply-to address and the envelope-sender address of the email.

## 6.5 SMS Alarm Messages

### 6.5.1 Overview

If a detection rule has an associated 'SMS' alarm type, then at the end of each check-cycle, if the rule alarm has been triggered an SMS alarm message will be sent to the cell phone numbers given in the alarm definition.

For each SMS recipient who is destined to receive an SMS alarm message at the end of the check-cycle, a summary of all rules which have been triggered is given.

Because of the size limitations of SMS messages, they may be truncated by the SMS service provider, and to this end, the serverM system will include only a summary of which alarms have been triggered.

See Appendix A for more details of how to set up an SMS alarm service.

### 6.5.2 SMS Alarm Message Contents

SMS messages will contain the following contents:

- the server on which the serverM system is running;
- the timestamp of the sending of the SMS message;
- a list of each of the rules which have alarms triggered during that check cycle (possibly truncated due to SMS message size limitations)

### 6.5.3 SMS Alarm Message Configuration

The parameters that affect the SMS alarm messages are (see section 4 for details of the master configuration file parameters):

- the master configuration file parameter '*sms-alarm*' which determines whether or not SMS alarms are sent;
- the master configuration file parameter '*sms-alarm-throttle*' which determines how many SMS alarms are sent in any 24-hour period (measured from midnight local time);
- the master configuration file parameter '*sms-to*' which determines the sender of the SMS email to the SMS service provider (usually used for authentication purposes);
- the master configuration file parameter '*sms-gateway*' which determines the IP or domain of the email-to-SMS gateway service;
- the master configuration file parameter '*smtp-gateway*' which determines the IP or domain of the SMTP relay server through which all serverM emails are sent.



## 7 The 'Standard' Rules

---

The serverM system is distributed with a number of pre-defined rules files. These are discussed in detail.

---

## 7.1 Overview

This section describes the standard rules files that are distributed along with the serverM system. These files will provide the first-time user with an effective detection system 'out of the box'. However, it is recommended that the rules definitions within the files are amended to more closely reflect the local server's requirements, and that additional rules are defined for bespoke elements of the local server's operations.

The serverM system is distributed with three pre-defined rules files:

- *config/rules.txt* which contains the main intrusion detection system rules;
- *config/ruleslib/\_p80\_arclog.txt* which contains the rule to archive large serverM log files
- *config/ruleslib/\_p80\_genreport.txt* which contains the rule to automatically generate the html report on the current serverM system status

The following sections discuss these rules files in detail and suggest amendments that may improve detection and prevention for your local installation.

## 7.2 The 'Main' Rules Definition File

The rules definition file '*config/rules.txt*' is serverM's main rules file. This file is always parsed first, and normally contains global definitions as well as the most important detection rules.

The serverM system ships with a pre-defined rules file that will work with both Mac OS X and Linux based operating systems. The file uses *conditional parsing* through the use of '*ifdef*' statements to ensure that only those rules specific to a given platform are parsed.

In this rule file the email alarm recipient is set to the email address defined in the '*administrator*' parameter in the master configuration file. It may be desirable to change this to an email address belonging to either an email distribution group for your security team. In order to do this, firstly place a '*def*' statement somewhere near to the top of the rules file:

```
def: email security-team (secteam@yourdomain.com)
```

and change the email alarms from:

```
email([administrator])
```

to:

```
email(security-team)
```

within each rule definition.

If you wish to use the same email address name throughout all of the rules files then use the '*global*' keyword:

```
global def: email security-team (secteam@yourdomain.com)
```

The rules contained within this file are detailed below based on the Linux platform (the Mac OS X rules contained within the file are similar).

### SETUID/SETGID

The first rules in the file are common to Linux and Mac OS X. They detect new files that have the '*setuid*' and '*setgid*' bits set in their ACLs.

These two rules depend on executing a system '*find*' command to find the files with the appropriate ACLs set. The pragmas are defined so as to only evaluate the rules once per day, and between the hours of midnight and 2am. This is because the processor load will be relatively high, as all files mounted on the root mount point will be checked (potentially hundreds of thousands of files).

Both '*setuid*' and '*setgid*' files are potential security threats, and the creation of new files with these flags set, or the changing of existing files to include these flags, could be seen as suspicious.

```

def: execute setuid ("find / -type f -perm 4000 -print")
def: execute getuid ("find / -type f -perm 2000 -print")

[severity 3] --
[initial] --
[on-day daily] --
[at-time 0000:0200] --
[once until fail] --
  check setuid::status setuid::baseline::email([administrator])

[severity 3] --
[initial] --
[on-day daily] --
[at-time 0000:0200] --
[once until fail] --
  check getuid::status getuid::baseline::email([administrator])

```

The *'initial'* pragma ensures that the rule isn't evaluated during the very first check cycle of the serverM system - which would delay the system startup process.

The *'baseline'* constraint of each rule ensures that the test will compare the changes to the files found against the record made when the rule was first evaluated.

## PASSWORD/ACCOUNT CHANGES

The next two rules are relatively straight-forward. They check for changes to the password files (both the main password file, and the shadow password file). If your system is not using shadow passwords (and it should be!) then the second rule is not required and can be removed from the rules file.

```

[severity 3] --
  check passwd file::file change->/etc/passwd::content:: --
    email([administrator])

[severity 3] --
  check shadow file::file change->/etc/shadow::content:: --
    email([administrator])

```

Both of these rules are determined to be medium severity (you may wish to change this if you feel that password or account changes are very significant on your specific server). The file change rule looks for any change in the content of the files and emails the administrator if any changes occur. The rules are not resource intensive and so are left to evaluate every check cycle.

## FAILED LOGINS

This rule is complex. It uses the stand-alone command *'scour'* which is supplied with serverM in order to check the system log file for unauthorised access attempts.

Depending on how your syslog configuration is undertaken, it may be necessary to check within a log file other than the main *syslog* file. This can easily be changed by editing the *'def'* line and changing the *'file'* parameter of the *scour* command.

```

{ failed login attempts

    def: execute three_failed_in_hour ("%path/scour" -yesno num:6
previous:01:00:00 pattern:'Authentication failure'
file:/var/log/syslog")

    def: execute keep-logstatus ("cp /var/log/syslog
'%path/log/%df.log'")

    [severity 1] --
    [once until fail] --
    [initial] --
        check failed login::status
        three_failed_in_hour::baseline::--
            email([administrator]), -
            execute(keep-logstatus)
}

```

This rule is scoped because there are multiple complex definitions, and it is visually clearer to group them together into one signature (this is a common reason for scoping which is an advantage over and above the localising of definitions).

The `'scour'` command, when used with these parameters, will return a '0' or '1' on the standard output depending on whether six or more lines within the system log contain the pattern 'Authentication failure'. If the alarm is triggered then the administrator is sent an email, and a copy of the log file is made so that it can be analysed later (an intruder may attempt to clear the log, or flush the log by causing log rotation).

If, on your server or desktop computer, you would expect to see a certain number of failed logins, then the threshold (determined by the `'num:'` parameter) could be changed to a higher number. Similarly, if you are more concerned about failed logins, then this could be reduced to only '1' or '2'.

### CONFIG CHANGES

On most \*nix systems, configuration of services and programs is undertaken using 'flat files' in the `'/etc/'` directory. A simple file change rule that recurses down the directory looking for changes to configuration files is defined within the rules file.

```

[severity 3] --
[initial] --
    check config files::file change->/etc/*.conf::content --
        recurse:5:--
        email([administrator])

```

The rule checks for any changes to content of any file with a `'.conf'` extension and emails the administrator if there have been any changes.

Note that it would be possible to add a 'repair' option to this rule that copied a standard configuration file back to the appropriate location if any changes were detected (and possibly making a backup of the changes made for later investigation).

## NEW DEVICE MOUNTS

In most \*nix variants, mounting a new device on the file system results in changes to the current mount map, and this can be detected and alarms produced if this behaviour is not expected.

```
def: execute get_volume_list ("mount | sort")
[severity 9] --
[once until fail] --
  check new volumes mounted::status get_volume_list::none::-
    email([administrator])
```

The use of the `'sort'` command within this rule is to ensure that there is no simple differences in the output of the `'mount'` command each time it is checked.

If someone now attempts to mount a new device such as a USB key or inserts a CD-ROM into the drive, an alarm will be generated.

## LOGIN AT UNUSUAL TIMES

On most servers, interactive logins are not expected during certain hours. This rule sends an alarm if a successful login occurs between 8pm and 8am.

```
def: execute who ("who")

[severity 1] --
[at-time 2000:2400, 0000:0800] --
[once until fail] --
  login at unusual time::status who::none:: --
    email([administrator])
```

The use of the `'once until fail'` pragma ensures that only one email will be sent for the first unusual login. This may not be required and can be removed if an alarm is required for each login between these times.

## STOPPING CORE SERVICES

The `'cron'` service, and `'syslog'` services are often targeted by attackers who gain access to a server, and it is important that these services are not stopped, and are re-started if an attacker manages to stop them. This rule checks that they are running and if they stop, sends an alarm and also attempts to restart them.

```
{ key process check

def: execute start-cron ("/etc/init.d/cron start")

[severity 2] --
[every 10] --
  cronis running::new service::lookfor:/usr/bin/cron::-
    email([administrator]), -
    execute(start-cron)

def: execute start-syslog ("/etc/init.d/sysklogd start")

[severity 2] --
[every 10] --
  syslog is running::new service::lookfor:/sbin/syslogd::-
    email([administrator]), -
```

```

        execute(start-syslog)
    }

```

Note that these rules are only evaluated every ten check cycles, and if an attacker manages to stop and restart the services within this period, it may be the case that these service interruptions are not caught by this rule. However, there are a number of solutions to this issue if this presents a potential problem.

The first is to increase the evaluation regularity by removing the 'every' pragma. The second is to use the 'scour' program to check the log files for the services stopping and starting.

### SERVICES STARTING AT UNUSUAL TIMES

In most instances, with very few exceptions, it would not be normal for services to stop and start at unusual times. This rule checks for new services starting up between 8pm and 6am, and sends an alarm if a new service is found.

```

{ new service at night

    ignore: service "/usr/sbin/backup"
    ignore: service "%path/htmlreport"

    [severity 10] --
    [at-time 2000:2400, 0000:0600] --
        new service at unusual time::new service::none::-
            email([administrator])
}

```

Again, by default this rule only checks every ten check cycles, and this may be increased if required.

In this example, an exception is expected: that of the 'backup' process, which is timed to run during these times. Also note that the rule is placed within a scope so that the service 'ignores' do not affect any other rules defined after this one.

### PROFILE CHANGES

If an attacker manages to gain access to a server then they may attempt to modify the root user's bash profile so that new processes are executed when the root user next logs in. This rule detects such changes and sends an alarm if any are detected.

```

{ root profile changes

    [severity 3] --
    [every 20] --
        root user profile change::file change ->
~root/.profile::content:-
            email([administrator])

    [severity 3] --
    [every 20] --
        root user bashrc change::file change ->
~root/.bashrc::content:-
            email([administrator])
}

```

The location of the root user's login scripts may be different if other shells are in use and the rules can be edited accordingly. Note that this rule is only evaluated every 20 check cycles, which should be sufficient to detect any compromise.

### STARTUP PROCESS CHANGES

An attacker may wish to change the startup process list so that any Trojan programs and scripts that they have deposited on the server survive a server reboot. This rule checks the Linux standard startup directory for changes and sends an alarm if any changes occur.

```
{ check startup processes

    [severity 1] --
    [every 10] --
        check startup processes::file change -> /etc/init.d/*:--
-
        content recurse:3::--
        email([administrator])
}
```

This rule will recurse into the */rcn.d* directories to ensure that the links to the main service control scripts are also not changed.

### SERVERM FILE CHANGES

It is advisable that any changes to the serverM system configuration, or daemon itself, is known about and alarms sent if any changes are detected. This rule detects changes to content.

```
[severity 2] --
[every 10] --
    serverm config file changes::file change ->
%path/config/*.txt::--
    content recurse:2::--
    email([administrator])

[severity 2] --
[every 10] --
    serverm library file changes::file change ->
%path/lib/*.pm::--
    content::--
    email([administrator])

[severity 2] --
[every 10] --
    serverm core file changes::file change ->
%path/serverM::--
    content::--
    email([administrator])
```



### 7.3 The Log File Archive Rules

The serverM distribution includes a log archiving rule that can be found in `'config/ruleslib/_p80_arclog.txt'`. The rule is defined by:

```
def: execute archive-log ("cp '%path/log/log.txt' --
    /tmp/sM%df.txt --
    & rm '%path/log/log.txt'")

[every 10] _
library_ arclog archive serverM log:: --
    file size->%path/log/log.txt:: -
    >50k::execute(archive-log)
```

This rule is evaluated every ten check cycles, and if the serverM log is greater than 50K in size it is copied to the `'tmp'` directory, and the old log is then deleted.

If larger log files are acceptable then the threshold may be increased. A compression utility may also be used to further reduce the size of the archived log file.

Note that if the `'log-location'` parameter is used in the mast configuration file, then this rule will need to be updated so that the correct path is used.

## 8 Frequently Asked Questions

---

Questions arising from the day-to-day running of the serverM system are regularly submitted to port80 and, along with general customer feedback, port80 maintains a short serverM FAQ.

---

## 8.1 Introduction

The following questions are the most frequently arising issues from the port80 serverM support email system.

If you have any questions about the serverM system which aren't answered within the FAQ then please send an email to [serverm@port80.com](mailto:serverm@port80.com)

## 8.2 serverM Configuration

Q. Can I manually edit my master configuration file using any text editor?

A. Yes. Any editor capable of editing and saving plain text files can be used to edit the master configuration file.

Q. Can I copy my master configuration file from one server to another?

A. Yes. There are no installation-specific settings in the master configuration file and an identical file may be distributed to any server running a version of serverM.

Q. Can I copy my rules file from one server to another?

A. Yes. There are no installation-specific settings in the rules file and an identical file may be distributed to any server running a version of serverM. Care should be taken that any commands used in 'execute' alarms are supported by the target server.

Q. Do I need to stop the serverM daemon in order for configuration changes to take effect?

A. No. Any changes made to the master configuration file will take effect automatically on the start of the next check-cycle. Check the log files for any errors in the configuration file, or wait for the administrator error report email if this has been configured.

Q. I don't want to create a text log file - do I have to have one?

A. No. You can set the log-location parameter in the master configuration file to /dev/null. This is not advised unless you have full logging to either a local syslog, or a remote logging server because critical alarm or log error messages may be missed.

### 8.3 Rules

Q. What happens if I define two rules which 'overlap' in their constraints? Are both rules triggered?

A. All rules are evaluated in isolation – overlapping rules will be triggered without interaction and it is common to design rules which overlap on a number of constraints.

Q. Should I include the entire root mount point in the file change rules?

A. Not usually – this will cause a great deal of processing resource overhead and will slow down the serverM reaction times. If it is necessary to check a large number of files then either increase the sleep period between each check-cycle, or use an `[every n]` pragma to lessen the number of times the files are checked.

Q. The serverM system complains about a rule that I've written and it appears to be syntactically correct – what should I do?

A. Email your rule definition (including any relevant 'def' statements), and the associated error information, to port80 at [serverm@port80.com](mailto:serverm@port80.com) and we will endeavour to provide information or feedback on why your system appears to have a problem with it.

## 8.4 Reports and Alarms

Q. If I want to send reports or alarm emails to a lot of recipients, how do I best achieve this?

A. Either set up a *distribution list* on the receiving SMTP server to distribute the email content to any number of individual recipients (which will ensure that the serverM system does not spend too much time sending multiple emails), or define a group using the 'def' keyword in the rules file (see section 5.5.2).

Q. How do I test if the email alarm system is working before any 'real' rule alarms are triggered?

A. Include a `[once only]` true rule in the rules file that sends an email alarm. This will ensure that at least one email alarm is sent. If an alarm does not appear to be sent then check the log file (see section 3.4) for error messages pertaining to the sending of emails etc.

Q. How do I test if the SMS alarm system is working?

A. Include a `[once only]` true rule in the rules file which sends an SMS alarm. This will ensure that at least one SMS alarm is sent. If an alarm does not appear to be sent then check the log file (see section 3.4) for error messages pertaining to the sending of emails, and the sending of SMS alarms etc.

Q. How do I know if an 'execute' alarm has worked successfully?

A. It depends on the command which is to be executed. The serverM system records which commands are executed during rule evaluation in the serverM log file, along with the result (if any) that the command line application returns after the execution has been completed.

## 8.5 Miscellaneous

Q. How does serverM defend against Denial of Service attacks against itself?

A. serverM is designed to react to Denial of Service attacks intelligently using a number of unique heuristics which ensure that processing overhead is minimised, but that no intrusion symptoms are missed.

Q. Can I stop and re-start the serverM system without affecting the detection of intrusion symptoms?

A. In most cases, the detection engine is unaffected by the stopping and re-starting of the watcher daemon. However, there are a number of exceptions:

- any [once only] and [once until fail] rules will become active again;
- event 'history' is lost for events up until the last event read by the detection engine during its final check-cycle before stopping.

Pausing and continuing the service will not have these effects unless the 'clear-on-pause' option is selected in the master configuration file (in which case the [once only] and [once until fail] rules will be reset.) If you need the system to simply stop sending alarms or evaluating rules, then use the 'status' parameter in the master config file (see section 4).

Q. If my server is compromised, how do I defend against a malicious intruder from stopping the serverM system itself?

A. If a malicious intruder gains administrator access to the server on which serverM is installed then there is only one line of defence – the sending of the 'shutdown' email if the watcher daemon is stopped. However, a sophisticated attack will attempt to overcome this by changing the administrator target for the shutdown emails. This is countered by the fact that any changes to the 'administrator' parameter in the master config file is ignored until the serverM watcher service is fully stopped and then re-started.

In most cases the initial probing of a system required to gain access will have caused various alarms to have been triggered and so damage may be limited or even negated by swift action by the alarm recipients.

Q. How do I write my own entries into the log file?

A. Define an execution rule, and associate it with a 'true' rule. For example:

```
def: execute write_log_file --
    ("echo 'my message' >> %path/log/log.txt")
write log entry::true::none::execute(write_log_file)
```

Substituting your own content for 'my message'. Take care not to overwrite the log file with an 'open new and write' redirector ('>'), but use an 'append' redirector instead ('>>').

## Appendix A: SMS Services

The serverM system sends SMS alarms to up to three cell phone numbers by using an 'email-to-SMS' service provider. The configuration file enables the administrator to setup three cell phone numbers, and to configure the system to send an email with the appropriate format to your SMS service provider.

Because there is not yet a standard format for all email-to-SMS services, serverM is programmed to use the most commonly found system currently in use. If your service provider requires (for example) additional authentication, then it is recommended to use an 'execute' alarm with a suitable third-party SMTP application (see below).

The serverM system has a number of requirements of the SMS service provider in order for the system to function correctly:

- the SMS service provider must be able to authenticate on the 'from' email field rather than using a separate authentication method;
- the SMS service provider must be able to accept the cell phone number as the email 'to' address (of the form <number>@gateway).

The configurator uses the 'sms-gateway' and 'sms-to' settings to indicate the domain (or IP) of the email-to-SMS server, and the email 'to' address respectively.

The serverM system generates the appropriate email with the alarm message truncated where necessary to attempt to bring the overall email size down to that of the service provider's SMS size limit. However, it is possible that the alarm message may be further truncated if there are a large number of alarms triggered, and it is recommended that all SMS alarms are duplicated with email alarms where possible.

---

### Note

port80 recommends the use of the SMS service provider 'sms2email', found at <http://www.sms2email.com>. This is because this supplier has been tested and found to work correctly. If you know of any other working (tested) supplier, please send details to [serverM@port80.com](mailto:serverM@port80.com) and we can include the details in this manual.

---

### Using Non-Standard email-to-SMS Services

Because the serverM system allows for the execution of arbitrary commands within alarms, if you are already a subscriber to an email-to-SMS service provider that does not use the required form of authentication or addressing, it is possible to send bespoke SMS alarms using a third party application.

For example, assuming an SMTP mailer application 'mail' with options for authentication, a bespoke alarm might be:



```
def: bespoke-email-alarm execute(" echo --  
    Rule unauthorised logon attempt %rn for %sv at %dt --  
    | mail -user:username -pass:password -  
        to:07291999999@somewhere.com --  
        from:server_%sv@yourdomain.com --  
        gateway:82.0.0.1")
```

## Appendix B: Log File Messages

### B.1 Overview

During normal operation, the serverM watcher daemon will write a number of log messages to the system's internal log file located at ``%path/log/log.txt`` unless configured differently within the master configuration file – see section 2.4 for an overview.

The messages which can be generated are listed below in type order.

## B.2 Info messages

- **sending type email to *administrator-email***

The serverM system is about to send an email to the administrator email address (defined in the config file) to inform them that the daemon is either shutting down, pausing, or continuing.

- **sent type email to *administrator-email***

The serverM system has successfully sent an email to the administrator email address (defined in the config file) to inform them that the daemon is either shutting down, pausing, or continuing.

- **configuration file changed - re-read file**

The serverM daemon has detected that the master configuration file has changed during running and has re-read the file and is about to parse it.

- **config file parsed OK**

The serverM daemon has detected that the master configuration file has changed during running and has re-read the file and has parsed it successfully.

- **rules file changed - re-read file**

The serverM daemon has detected that one of the rules files has changed during running and has re-read the file and is about to parse it.

- **rules file reading results:**

The serverM daemon has detected that one of the rules files has changed during running and has re-read the file and has parsed it, the results will appear in the following log file entries.

- **parsed rule *name* OK**

The serverM daemon has parsed the rule named and it was parsed without errors.

- **rules file parsed OK**

The rules file (or files) have been parsed.

- **sending alarm SMS to *sms\_number* with *message***

A rule has been triggered that has an associated SMS alarm, and the alarm is about to be sent with the message given.

- **SMS message sent to *sms\_number***

An SMS alarm has been successfully sent to the sms number given. Note: this indicates that the email has been sent successfully, not that the email to SMS gateway has managed to send the SMS message.

- **sending html daily report to *email-addresses***

The serverM daemon is about to send the daily report email to the email addresses listed.

- **not sending error report by email, no administrator email address is configured**

No email address is specified for the administrator in the config file and so the email message that would have normally been sent has not be dispatched.

- **sending error report email to *administrator-email* at gateway gateway**

The serverM daemon has encountered an error and is sending an email to the 'administrator email' defined in the config file at the SMTP gateway address given.

- **rule *name* is 'once only' and has been triggered, now inactive**

The rule named with the pragma [once only] has been triggered and will no longer be active.

- **rule *name* is 'once until fail' and has been triggered, now inactive until fail**

The rule named with the pragma [once until fail] has been triggered and will no longer be active until the next time it fails to trigger.

- **executed command *execute* for rule *name***

The rule named has been triggered and the named execute alarm has been executed.

- **adding checksum for new file *file***

A file change rule has been evaluated with the file named in scope and a checksum for that file has been added to the files checksum database.

- **status command *command* in *name* created oversized output - ignoring rule (>1000 lines)**

The status rule named has executed the command named, but the output is extremely long, and in order to ensure that the check-cycle is not extended the output is being ignored (note that this rule will not evaluate again!)

### B3. Error Messages

- **can't connect to SMTP server for email alarm!**

An attempt was made to connect to an SMTP server specified in the configuration file, but the server was unreachable.

- **error in configuration file config.txt: <ERROR MESSAGE>**

There has been one or more errors whilst parsing the master configuration file. The remainder of the message provides details.

- **failed to send shutdown email to \$administrator!**

- **failed to send pausing email to \$administrator!**

- **failed to send continuing email to \$administrator!**

It was not possible to send an email to the administrator nominated in the master configuration file due to the SMTP server not responding or rejecting the email.

- **there was an error reading the configuration file config.txt**

There was at least one error in the master configuration file, or there was a problem reading the file itself. This error message is usually followed by additional, more detailed, error messages.

- **configuration file error: (line n) Don't understand input field <FIELD>**

There was a parameter defined in the master configuration file that was unexpected. The 'field' value describes which parameter caused the error and 'n' provides the line number.

- **configuration file error: Can't find config file in directory <DIRECTORY>**

The master configuration file could not be found in the usual location (%path/config/config.txt).

- **configuration file error: (line n) <VALUE> is not a permissible value for field <FIELD> (default used)**

An unexpected type of value was provided for a parameter within the master configuration file and it has been rejected. The system has set the value of the parameter to the default value.

- **error in rules file(s) - can't read ruleslib <DIRECTORY>**

It was not possible for the system to read the files in the rules definition subdirectory (%path/config/ruleslib). Even if additional rules files are not placed in this directory, it should be present and readable by the system.

- **Can't find rules file <FILE>!**

The system could not find the main rules file %path/config/rules.txt. This file should always be present.

- **error in rules file(s)**

There is an error in one of the rules files parsed by the system. Following error messages provide more detail.

- **Signature scope not ended before end of file**

A signature block has been started within a rules file, but the end of file has been reached before the end of the block was defined.

- **Already in scope! Ignoring scope enter**

A signature block has been started within an existing signature within a rules file – this is not permitted.

- **No scope name found!**

All signature blocks have to be named by following the opening brace with the name.

- **Can't exit scope - not in one! Ignoring scope exit"**

An end of signature block closing brace has been found, but there appears to be no corresponding opening block. It will be ignored.

- **Don't understand definition <DEFINITION> <DETAILS>)**

A rule definition has been made in a rules file that the system cannot parse directly. Details are provided, but they may not be too helpful!

- **Don't understand ignore definition <DEFINITION> (<DETAILS>)**

An 'ignore' definition has been made in a rules file, but either the type is incorrect, or the definition is incomplete in some manner. Details are included.

- **Don't understand line <LINE> (fewer than 4 parts to rule!)**

A rule definition has been made in a rules file, but there are not four parts to the rule. All rule definitions require a name, type, constraint, and alarm definition.

- **string <STRING> not defined in <LINE>**

The string that is referenced in the rules file line provided has not been defined earlier in the rules file.

- **'every' pragma is set to zero!**

A rule has been defined in a rules file with an 'every' pragma with a zero value. This would define that the rule is never evaluated.

- **'on-cycle' pragma is set to zero!**

A rule has been defined in a rules file with an 'on-cycle' pragma with a zero value. This would define that the rule is never evaluated.

- **'throttle' pragma value is not understood!**

A 'throttle' pragma has been defined with a value that is of an incorrect type, or unusual value.

- **Don't understand pragma <PRAGMA>**

An unsupported pragma has been defined.

- **Name for rule must have at least one alpha-numeric character**

All rule names must have at least one character from the set A-Z, a-z, or 0-9. The rule being defined in the rules file does not have such a character.

- **name for rule <NAME> is duplicated**

Rule names must be unique within a rules file, or within a signature block. A duplicate name has been found within a rules file.

- **alarm type is not understood (<ALARM>)**

An alarm definition within a rule in the rules file is not syntactically correct.

- **cannot have once-only and once-until-fail in same rule <NAME>**

The named rule appears to have both types of pragmas, but the combination is non sensical. The named rule will be ignored until this is changed.

- **Don't understand rule NAME of type '<VALUE>'**

A rule is being defined which the system cannot determine the type of.

- **don't understand command in <COMMAND> (no surrounding quotes?)**

A command is being defined in a 'def' statement, within a rules file, that is not syntactically correct. The common error is to omit the required quotes, and none appear to have been found.

- **don't understand command <COMMAND> in <VALUE>**

A command is being defined in a 'def' statement that is not syntactically correct.

- **don't understand status command in <COMMAND>**

There has been an error parsing a 'status' rule in a rules file.

- **don't understand filename in <FILENAME>**

There has been an error parsing a 'filechange' or 'filesize' rule within a rules file.

- **can only execute one command for status rule, <COMMAND> has multiple commands in <VALUE>**

More than one commands have been found within a definition of a 'status' rule within a rules file.

- **don't understand file change constraint in <NAME> (<CONSTRAINT>)**

A file change rule named <NAME> has been defined within a rules file that has a constraint value that the system does not understand. Details of the constraint are included.

- **don't understand constraint in <NAME> (<CONSTRAINT>)**

A general syntax error has been found in the constraint portion of a rule definition. Details of the error are included.

- **don't understand filename in <VALUE> (filesize)**

A file size rule has been defined within a rules file that has a file name value that the system does not understand. Details of the constraint are included.

- **don't understand file size in <CONSTRAINT>**

A file size rule has been defined within a rules file that has a constraint value that the system does not understand. Details of the constraint are included.

- **don't understand new service constraint (<CONSTRAINT>)**

A new service rule has been defined that contains a constraint that the system does not understand. Details of the constraint are included.

- **can't understand the TRUE rule constraint in <CONSTRAINT>**

A TRUE rule has been defined that contains a constraint that the system does not understand. Details of the constraint are included.

- **can't find email group '<GROUP>'**

An alarm has been defined within a rule that contains a group name that has not been predefined. The name of the group is included.

- **can't find execute command '<COMMAND>'**

An 'execute' definition is being used in a status rule, or an alarm execute, but the execution definition has not been defined.

- **can't find sms group '<GROUP>'**

An SMS alarm group is being used in an SMS alarm definition, but the group has not been defined.

- **don't understand <ALARM>**

An alarm definition within a rule is not understood by the system. The <ALARM> is provided.

- **email group name <GROUP> already exists**

An attempt has been made to define an email group using a name that is already defined in the current scope.

- **email <EMAIL> in <LINE> is not an email address**

An email alarm has been defined in the line provided, but the email address given is not a valid email address.

- **sms group name <GROUP> already exists**

An attempt has been made to define an SMS group using a name that is already defined in the current scope.

- **sms <SMS> in <LINE> is not an sms cell phone number**

An SMS alarm has been defined in the line provided, but the SMS number given is not a valid telephone number.

- **execute command <NAME> already exists**

An attempt has been made to define an execute command using a name that is already defined in the current scope.

- **string <GROUP> already exists**

A generic error message which asserts that a group name (email or SMS) has already been used in the current scope – this message will usually be followed by a more descriptive one.

- **don't understand ignore type <IGNORE>**

An 'ignore' definition has been attempted with a type that is not understood by the system.

- **don't understand ignore definition <LINE>**

An 'ignore' definition appears to be syntactically incorrect.

- **(file change) Can't open db for reading checksums**

The database file for storing file change checksums and ACLs could not be opened. Either it was not possible to create a new file (for the first instance of a 'filechange' rule), or an existing database file has permissions set so that it cannot be read by the system.



- **(new service) Can't open db for reading checksums**

The database file for storing service checksums could not be opened. Either it was not possible to create a new file (for the first instance of a 'new service' rule), or an existing database file has permissions set so that it cannot be read by the system.

- **can't open <FILE> found for file change check**

A specific file is being checked as a result of a 'filechange' rule, but the file ACLs could not be read. This may be due to restrictive file permissions, or the file is corrupt in some manner. In file 'globbing' rules (that use wild cards to specify which files are checked) only this file will be ignored and the remainder of any files will be checked as requested.

- **can't open file <FILE> for MD5 check!**

A specific file is being checked as a result of a 'filechange' rule, but the file contents could not be read so that an MD5 checksum could be created. This may be due to restrictive file permissions, or the file is corrupt in some manner. In file 'globbing' rules (that use wild cards to specify which files are checked) only this file will be ignored and the remainder of any files will be checked as requested.

- **error occurred during checksum insert <ERROR>**

An attempt to write an MD5 hash checksum into the filechange database has failed for some reason. The error provided may include details.

- **error occurred during checksum delete <ERROR>**

An attempt to delete an MD5 hash checksum from the filechange database has failed for some reason. The error provided may include details.

- **status command <COMMAND> in <RULE> created oversized output - ignoring rule**

The 'status' rule will only check changed output if the current output size is lower than a pre-set threshold (1000 lines of output). If the output results in a higher level of output then it will be ignored. This does not protect against very large output, for example the command `cat /dev/random` will effectively stop, and even crash, the serverM system. This issues will be addressed in future versions of the system.

- **not in ifdef scope! Ignoring ifdef scope exit**

The rules parser has found an 'enddef' statement, but the parser has not entered a conditional 'ifdef' block.

- **Already in ifdef scope! Ignoring ifdef scope enter**

Conditional 'ifdef' blocks cannot be nested.

- **conditional scope not ended before end of file**

Conditional 'ifdef' blocks cannot span multiple rules files. The parser was expecting to find an 'enddef' statement before the end of the file, but failed.

## B4. Status Messages

- **serverM system has stopped**

The serverM daemon has been terminated graceful and has now stopped detection.

- **serverM system has paused**

The serverM daemon has received a 'HUP' signal and has paused evaluating rules at the end of the check-cycle. Evaluation will continue once a further 'HUP' signal is received.

- **serverM system continuing**

The serverM daemon has been in 'pause' mode and has received a 'HUP' signal. Evaluation of the current rules will now continue.

- **Heartbeat: starting check-cycle**

The serverM daemon is about to start evaluating rules for the current check-cycle.

- **Heartbeat: check-cycle ended – alarms have been sent (av: x secs)**

The serverM daemon has completed a full check-cycle and alarm messages have been sent to either email recipients or SMS recipients. The average number of seconds that check-cycles are currently taking is included.

- **Heartbeat: check-cycle ended - no alarm messages sent (av: x secs)**

The serverM daemon has completed a full check-cycle and no alarm messages have been sent to either email recipients or SMS recipients. The average number of seconds that check-cycles are currently taking is included.

## B5. Alarm Messages

- **file change rule *name* occurred (file *file* created)**

The named file has been created, triggering the file change rule named that contained the filename and the constraint 'new'.

- **file change rule *\$name* occurred (file *\$file* deleted)**

The named file has been created, triggering the file change rule named that contained the filename and the constraint 'deleted'.

- **file deleted rule *<NAME>* occurred**

A 'filechange' rule that was specifically looking for the presence of a file has triggered.

- **file change (created) rule *<NAME>* occurred**

A 'filechange' rule that was specifically looking for the presence of new files has triggered.

- **file change rule *<NAME>* occurred**

A 'filechange' rule that was specifically looking for the presence of changed files has triggered.

- **file change rule *<NAME>* occurred (file *<FILE>* *<TRIGGER>*)**

A 'filechange' rule has occurred. Details include the rule name, the file that has changed, and the trigger (constraint) that caused the alarm.

- **new Service rule *<NAME>* occurred**

A 'new service' rule (named) has been triggered.

- **status rule *<NAME>* occurred**

A 'status' rule (named) has been triggered.

- **true rule *\$name* occurred**

A 'true' rule (named) has been triggered.

- **users rule *\$name* occurred**

A 'users' rule (named) has been triggered

## Appendix C: The 'checkrules' Tool

The serverM installer places a command line application in the serverM home install directory. This application is capable of parsing any number of rules files and indicating whether or not there are any syntactic errors. It can be useful to check any new rules that are to be added to a serverM install. The application also calculates MD5 hashes for all rules that are parsed so that their integrity can be checked against a published hash.

The syntax of the checkrules application is as follows:

```
checkrules [-r] [-h] [-md5] c:<config dir> [PATH|FILE|PATH\ruleslib]
```

Where:

- [-r] is an optional parameter that causes a report of all successful rules parsing as well as errors;
- [-h] is an optional parameter that cause the 'help text' to be printed on the terminal;
- [-md5] is an optional parameter that will calculate and display the MD5 hash of all rules that are parsed;
- c:<config dir> defines the directory in which the current master configuration can be found;
- PATH is any path (absolute or relative to the current directory), and will cause any file called 'rules.txt' and any text files within PATH\ruleslib to be checked;
- PATH\ruleslib is the relative or absolute path to a directory called 'ruleslib' which will cause any text files within to be checked;
- FILE is a filename (including an optional absolute or relative path), and will cause that file to be checked only.

To execute the command, open up a command line window and navigate to the serverM home install directory.

### Example uses of the command.

Check the file rules.txt in the config directory and all .txt files in the ruleslib subdirectory of the config directory of the current serverM installation (this will include all rule files currently parsed by the serverM system), and give a full report:

```
%>./checkrules -r c:config config
```

The following screen-capture shows the `checkrules` command being used to check the rules files for the current installation of serverM.

```

Terminal — sh — 111x40

Start rules checking...
processing arg -r
processing arg c:config
processing arg config
Checking file /Users/david/Documents/serverM/config/rules.txt: Parsed ok
Checking file /Users/david/Documents/serverM/config/ruleslib/ftplib.txt: Parsed ok
Checking file /Users/david/Documents/serverM/config/ruleslib/genreport.txt: Parsed ok

REPORT (in order of evaluation)
  OK Rule -> /rules.txt: rules check setuid
  OK Rule -> /rules.txt: rules check getuid
  OK Rule -> /rules.txt: rules {passwd file change from nidump} check nidump passwd file
  OK Rule -> /rules.txt: rules {failed login attempts} check failed login
  OK Rule -> /rules.txt: rules check config files
  OK Rule -> /rules.txt: rules check new volumes mounted
  OK Rule -> /rules.txt: rules login at unusual time
  OK Rule -> /rules.txt: rules {key process check} cronis running
  OK Rule -> /rules.txt: rules {key process check} syslog is running
  OK Rule -> /rules.txt: rules {new service at night} new service at unusual time
  OK Rule -> /rules.txt: rules {root profile changes} root user profile change
  OK Rule -> /rules.txt: rules {root profile changes} root user bashrc change
  OK Rule -> /rules.txt: rules {check startup processes} check system startup processes
  OK Rule -> /rules.txt: rules {check startup processes} check local startup processes
  OK Rule -> /rules.txt: rules serverm config file changes
  OK Rule -> /rules.txt: rules serverm library file changes
  OK Rule -> /rules.txt: rules serverm core file changes
  OK Rule -> /ruleslib/ftplib.txt: ruleslib/ftplib {ftp server access} late night ftp access
  OK Rule -> /ruleslib/ftplib.txt: ruleslib/ftplib {ftp server access} weekend ftp access
  OK Rule -> /ruleslib/genreport.txt: ruleslib/genreport {generate html report} generate html report if r
ules change
  OK Rule -> /ruleslib/genreport.txt: ruleslib/genreport {generate html report} generate html report if r
uleslib change
  OK Rule -> /ruleslib/genreport.txt: ruleslib/genreport {generate html report} generate html report if c
onfig change
  OK Rule -> /ruleslib/genreport.txt: ruleslib/genreport {generate html report} generate all report
  OK Rule -> /ruleslib/genreport.txt: ruleslib/genreport {generate html report} generate new log report
  OK Rule -> /ruleslib/genreport.txt: ruleslib/genreport {generate html report} regular make report
Sentry:/Users/david/Documents/serverM root#

```

**Figure E.1: Using the 'checkrules' Command**

## Appendix D: The 'htmlreports' Tool

The 'htmlreports' tool is a command line application which installs into the serverM home installation directory. The tool is capable of generating a set of web pages that report on the current status of the serverM installation at the time of the tool being executed, including:

- information on the current watcher service run status;
- the last modification times for the configuration, rules, and log files;
- whether or not there are any configuration errors, or rules errors;
- comprehensive details on the current log file (most recent alarms, error reports, and the most recent log file entries);
- comprehensive details on the current rules files (definition of all of the current rules, details on global and local ignores, any errors within the rules file);
- a view of the current configuration file, with current values, and any configuration errors.

In order to generate the HTML report, the tool uses the same parsing and analysis tools as the serverM system itself, so that the report is an accurate reflection of how the serverM detection engine itself parses rules and configurations.

The htmlreport tool re-parses the current configuration and results files, and analyses the current log file, and then outputs the result as a set of static HTML pages which can be viewed by any web browser.

It is possible to run the htmlreports application as the result of a serverM rule execution alarm, and this automates the reporting process so that if there is a web server running on the same server as serverM, a real-time snapshot of the system status is always available (example rules to generate reports are given below.)

The syntax for the htmlreports tool is:

```
%>./htmlreports path:<path to serverM> root:<path to html target dir>
[rules | config | log] limit:<number>
```

The path to serverM is the current home installation directory for the serverM system ('C:\Program Files\serverM' by default.) The path to the target directory is the directory into which the generated HTML pages are to be written. The 'rules', 'config' and 'log' parameters determine which pages are written (any combination of these three parameters can be used):

- 'rules' tells the htmlreport tool to generate an HTML page detailing the current rules definitions;
- 'config' tells the htmlreport tool to generate an HTML page detailing the current configuration;
- 'log' tells the htmlreport tool to generate an HTML page detailing the current log entries.

The 'limit' parameter limits the number of lines that the htmlreport tool reads from the end of the current log file (defaults to 20 lines if the parameter is not used).

The htmlreport tool will automatically generate a home page (called main.html) which will contain links to the remaining pages (rules.html, config.html, and log.html respectively).

### Example.

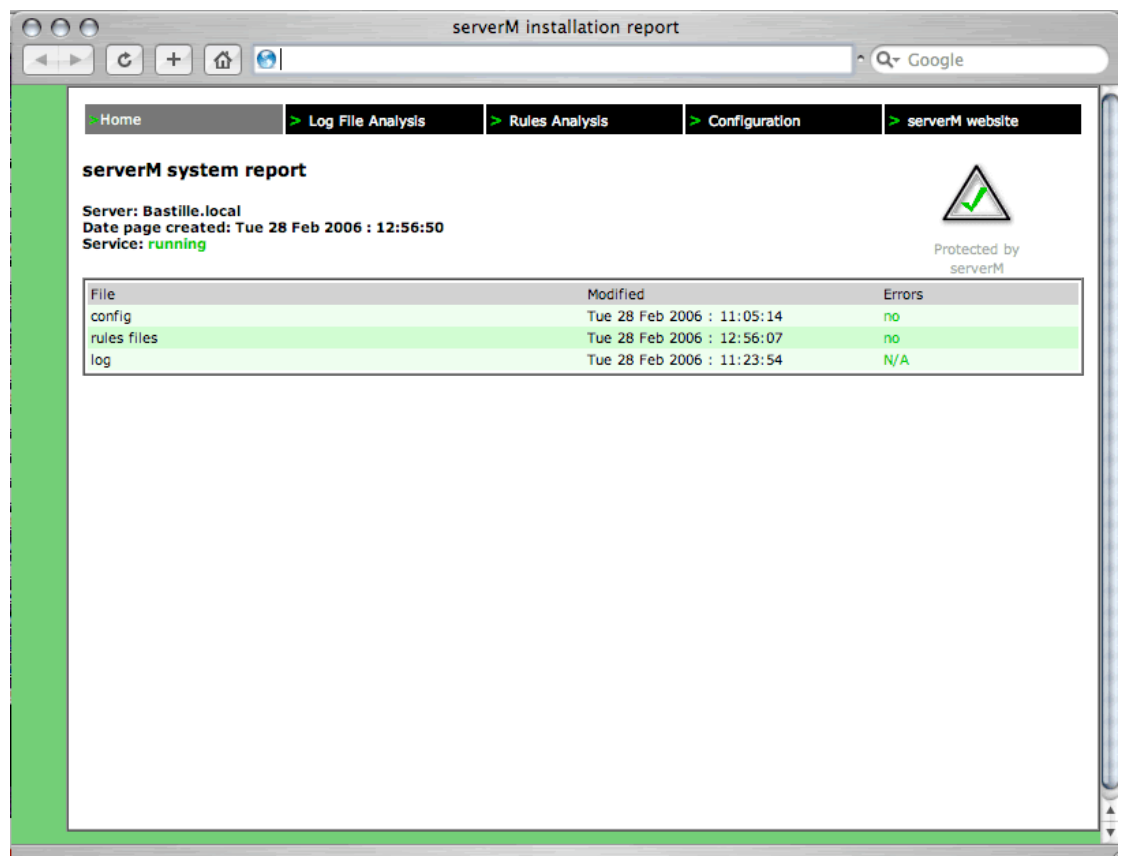
Consider the following command:

```
%>htmlreports path:. root./htmlreports rules config log
```

this command will generate a number of HTML files in the 'htmreports' directory, of which "main.html" is the home page of the report site.

### Screenshots.

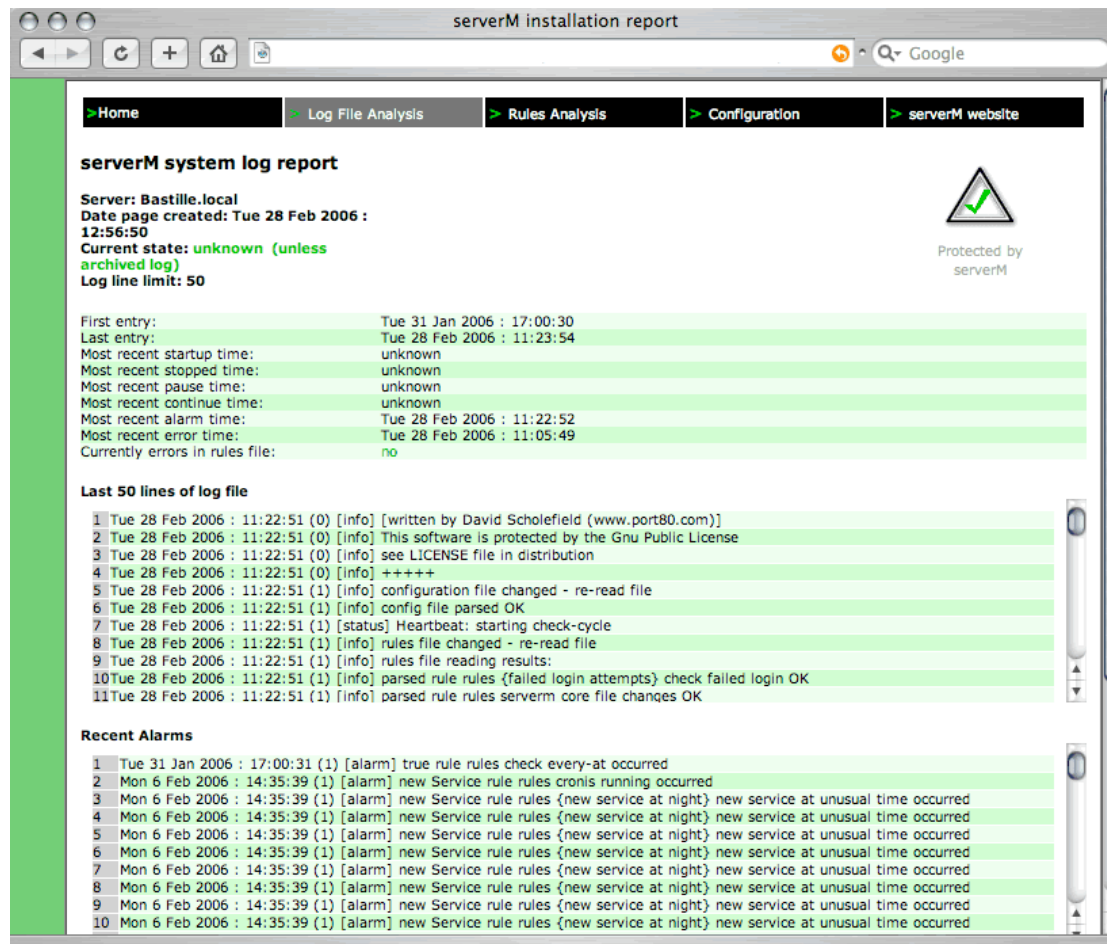
The home page contains information on the current status of the watcher daemon, and configuration files:



**Figure D.1: The htmlreports Home Page**

The menu bar at the top of the home page gives access to the remaining pages.

The log analysis page provides detailed information on the current log file:



**Figure D.2: The htmlreports Log Analysis Page**

The rules analysis page shows all current rules, along with any errors in any of the rules files:



The screenshot shows a web browser window titled "serverM installation report". The navigation bar includes links for Home, Log File Analysis, Rules Analysis (selected), Configuration, and serverM website. The main content area is titled "serverM system rule parsing report" and includes a green checkmark icon with the text "Protected by serverM".

**Server: Bastille.local**  
**Date page created: Tue 28 Feb 2006 : 12:56:50**

**Rules parsed successfully (in order of evaluation)**

check passwd file	File: rules.txt
check shadow file	File: rules.txt
{failed login attempts} check failed login	File: rules.txt
check config files	File: rules.txt
login at unusual time	File: rules.txt
cron is running	File: rules.txt
syslog is running	File: rules.txt
{new service at night} new service at unusual time	File: rules.txt
{root profile changes} root user profile change	File: rules.txt
{root profile changes} root user bashrc change	File: rules.txt
serverm config file changes	File: rules.txt
serverm library file changes	File: rules.txt
serverm core file changes	File: rules.txt
{ftp server access} late night ftp access	File: ruleslib/ftptlog
{ftp server access} weekend ftp access	File: ruleslib/ftptlog
{generate html report} generate html report if rules change	File: ruleslib/genreport
{generate html report} generate html report if ruleslib change	File: ruleslib/genreport
{generate html report} generate html report if config change	File: ruleslib/genreport
{generate html report} generate all report	File: ruleslib/genreport
{generate html report} generate new log report	File: ruleslib/genreport
{generate html report} regular make report	File: ruleslib/genreport

**Rules parsed unsuccessfully**  
 No rules parsed unsuccessfully

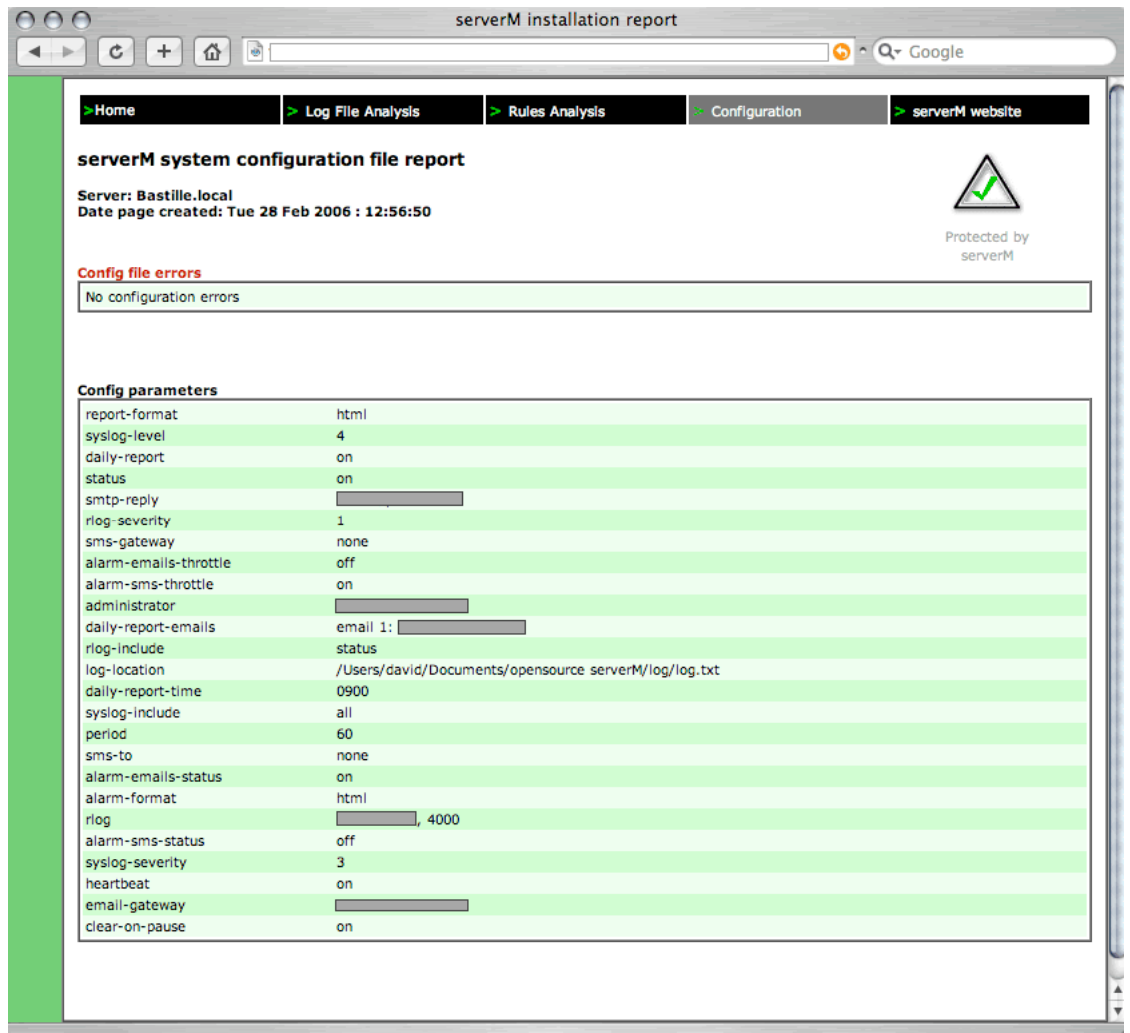
**All Errors in Rules Files**  
 No Errors in Rules Files

**Global ignores**  
 No global ignores

<b>Name:</b>	check passwd file	<b>File:</b>	\Users\david\Documents\opensourceserverM\rules.txt	<b>Scope:</b>	N/A
<b>Definition</b>	filechange -> \etc\passwd:;-acl content				

**Figure D.3: The htmlrports Rules Analysis Page**

The configuration analysis page shows the current configuration file contents, along with any errors:



**Figure D.4: The htmlreports Configuration Analysis Page**

**Automatically generating report pages.**

Rather than using the tool at the command line, it is possible to generate HTML reports using a rule in the serverM rules file. It is normally advisable not to generate the reports on every check-cycle (for efficiency), but rather to generate reports at any time that relevant information changes.

The example rule definition will generate a report under the following conditions:

- the configuration file or rules files have changed (in which case generate only the relevant pages);
- there have been any alarms (using the [if-or \*] pragma);
- every 60 check-cycles (about once an hour for the default 'period' setting of 60 seconds) only on weekdays;

Note that in this example the root directory for publishing the HTML report is one inside the serverM directory, normally this would be placed under a web server root directory.

```
# we want to create a new report if there have
# been any relevant changes, or every 60 cycles otherwise

{ generate html report
  def: execute generate_log_report ("perl -I --
    '%path/lib' '%path/htmlreport' --
    root:'%path/htmlreports' path:'%path' limit:50 log")

  def: execute generate_config_report ("perl -I --
    '%path/lib' '%path/htmlreport' --
    root:'%path/htmlreports' path:'%path' limit:50 config
log")

  def: execute generate_rules_report ("perl -I --
    '%path/lib' '%path/htmlreport' --
    root:'%path/htmlreports' path:'%path' limit:50 rules
log")

  def: execute generate_all_report ("perl -I '%path/lib/'
    '%path/htmlreport' --
    root:'%path/htmlreports' path:'%path'" limit:50 rules
log config")

  generate html report if rules change::-
    file change->%path/config/rules.txt::content:: --
    execute(generate_rules_report)

  generate html report if ruleslib change::-
    file change->%path/config/ruleslib/*.txt::content:: --
    execute(generate_rules_report)

  generate html report if config change::-
    file change->%path/config/config.txt::content:: --
    execute(generate_config_report)

  [if-or *] generate new log report::-
    true::none::execute(generate_log_report)

  [on-day weekday] [every 60] --
    regular make report::true::none:: -
    execute(generate_all_report)}
```

## Appendix E: The 'scour' Tool

The 'scour' tool is shipped with the default installation of serverM, and was designed to be used within 'status' rules in order to check the contents of various types of standard log files for patterns of entries. The tool is aware of the timestamps found in *syslog*, *apache error logs*, and *apache access logs*, and can use the timestamps to focus the search for log entries to specific time periods.

For example, it is possible to ask the scour tool to check the syslog entries for a given number of failed login attempts within the last hour, or between the hours of 6pm and 6am.

The syntax for the scour command is given below:

```
usage: scour [options] file:[!]<logfile> pattern:<pattern>

[-list]           list lines from logfile that match constraints
[-count]          show count of number of lines in logfile that
                  match constraints
[-yesno]          show either '0' or '1' depending on whether any
                  lines were found
[previous:<time>] only consider lines in logfile added in
                  previous <time> (see below)
[num:<num>]        look for at least 'num' number of lines (used with
                  -yesno option)
[from:<date>]      bound search to those lines added on or after date
                  (see below)
[to:<date>]        bound search to those lines added on or before date
                  (see below)
[--verbose]       be verbose with reporting

<time> is defined in format 'hh:mm:ss' with or without quotes
<date> is in Unix date string format 'YYYY-MM-DD HH:MM::SS'
[!]<logfile> is any text file currently on any mounted volume, the
'pling' (!) will treat the file name as a command, and will use
the output to check for the pattern
<pattern> is any Perl regular expression

list, count, and yesno options are mutually exclusive
'previous' is mutually exclusive to 'from' and 'to' options
```

The two most important values are those for the logfile which to search in, and the pattern to search for. The logfile can be any file on the local server or mounted volume, and the pattern is any Perl regular expression (which may be a simple string of characters).

If the filename is preceded with a 'pling' (!) then the filename is treated as a command to pass to the system shell, and the resulting output to STDOUT is treated as the file contents to search. This can be particularly useful for unzipping automatically zipped log files before they are searched.

Examples:

```
%>./scour -yesno previous:12:00:00 num:3 file:/var/log/system.log
pattern:'su:.*to root'
```

Checks the log file 'system.log' for strings of the form 'su: <anything> to root' occurring within the previous 12 hours. In particular, the '-yesno'

option will result in a '0' being returned if there are not at least three such entries (num:3), and '1' if there are at least three such entries.

```
%>./scour -list -v previous:12:00:00 num:3 file:'!gunzip -c
/var/log/system.log.0.gz' pattern:'su:.*to root'
```

Look for the same pattern, but unzip the last archived system log in real-time and check the resulting text lines for the pattern.

```
%>./scour -list previous:36:00:00 num:3 file:/var/log/httpd/error_log
pattern:'permissions deny server execution'
```

Look for three or more occurrences of the pattern 'permissions deny server execution' in the error logs within the previous 36 hours for the Apache server, and list the resulting lines.

The scour command provides a great deal of additional flexibility to the 'status' rule. For example, it is possible to trigger an alarm if there are suspicious failed login attempts in the system log:

```
# check for unauthorised access
{ failed login attempts

    def: execute three_failed_in_hour ("%path/scour" -yesno --
        num:3 previous:01:00:00 pattern:'Authentication --
        failure' file:/var/log/system.log")

    def: execute three_failed_in_hour_zipped ("%path/scour" --
        -yesno --
        num:3 previous:01:00:00 pattern:'Authentication --
        failure' file:!/var/log/system.log.0.gz")

    def: execute keep-logstatus ("cp /var/log/system.log --
        '%path/%df.log'")

    def: execute keep-logstatus_zipped --
        ("cp /var/log/system.log.0.gz --
        '%path/%df.log.gz'")

    [severity 1] --
    [every 30] --
    [once until fail] --
    [initial] --
        check failed_login::status --
            three_failed_in_hour::baseline::-
            email(administrator), -
            execute(keep-logstatus)

    [severity 1] --
    [every 30] --
    [once until fail] --
    [initial] --
        check failed_login_archived::status --
            three_failed_in_hour_zipped::baseline::-
            email(administrator), -
            execute(keep-logstatus_zipped)

}
```

This rule checks the system log every 30 check cycles. The pattern being searched for is 'Authentication failure', and if there have been at least three such occurrences in the previous hour then a '1' will be output onto the STDOUT, otherwise a '0' will be produced. The status rule will note any changes in the output, and will trigger the alarms if any difference is present. One of the alarms is an 'execute' alarm that copies the current log file to a temporary location so that the evidence is preserved. Note that the previous system log is also checked by unzipping the contents of the compressed file in real-time and checking the entries. This is to ensure that if the rule is evaluated during a period when the system log has been recently rotated, that no entries will be missed.<sup>2</sup>

---

<sup>2</sup> This rule definition is based on a Mac OS X syslog configuration. For a Linux equivalent, the log location will be different, and the log entry string pattern will be different. The remainder of the rule definition will be identical.

---

## Appendix F: serverM Remote Monitor (srm)

The serverM remote monitor tool ('srm') is shipped with the default installation of serverM, and is designed to provide a service that enables a system administrator to listen for serverM remote log messages across the network, and to store those messages in either a central MySQL database, or a single central log file.

In order for the *srm* system to be able to listen on the correct port, the master configuration file of the serverM instance is configured with the central server's IP address and port, and the *srm* system is configured with matching details. Configuration details are discussed below.

### F1. serverM Master Configuration File and srm

The '*rlog*' parameters in the serverM master configuration file enables a serverM instance to send log messages and alarms across the network using the UDP network protocol. Each instance of serverM can be configured with the IP address of a single, central, log server. The configuration parameters are as follows:

- rlog*: <ip or domain>, <UDP port>, <optional key>
- rlog-severity*: <number>
- rlog-include*: <type of log messages to send>

The '*rlog*' parameter defines the IP address, or fully qualified domain name, of the central log server, the UDP port to send the messages on, and an option encryption key (a simple string of characters which is used to encrypt the log message using the Blowfish algorithm). If the optional key is not included then the messages are sent in plain text (which is not advised on security grounds).

The '*rlog-severity*' defines the maximum severity level of rules for which alarms will generate remote logging messages e.g. a severity level of '5' will mean that any rule with a severity pragma set to 5 or below will be remote logged (as will any rules *without* a severity pragma, which is assumed to be a severity level of '0' by default).

The '*rlog-include*' parameter defines the other types of messages that will be sent to the remote logging server, and may be a list of types (separated by a comma), or the keyword '*all*', which will include all types ('status', 'info', and 'error').

### F2. srm Centralised Logging Daemon and Configuration

The *srm* program uses a configuration file similar to the master configuration file for the serverM instance.

The *srm* configuration file – called *config.txt* can be located in any directory, and its path is indicated on the command line when the *srm*

daemon is started. This file is a simple text file, and can be edited with any text editor of your choice.

The configuration file contains a single line per parameter, and each line is of the form:

```
parameter-name:parameter-value
```

The configuration file may have any number of blank lines. Any text including, and following, a 'hash' symbol ('#') will be ignored. Parameter name and value pairs may not be split over multiple lines, and there must be only one parameter name and value pair per line.

The order in which the parameter name and value pairs appear in the file is not significant.

There are only six types of parameter that can be defined within the configuration file, and these are detailed below:

Parameter	Values	Description
log-location	Path to any directory e.g. /tmp	The directory which the <i>srn</i> log file will be stored (the logfile will have the name ' <i>srnlog</i> ')
screen	'on' or 'off'	Determines whether received log messages will be printed to the current terminal. <b>Note:</b> this parameter also determines whether or not the <i>srn</i> program runs as a daemon or interactively. If the 'screen' parameter is set to 'on' then the program will run interactively, otherwise it will run as a daemon
db-ip	IP address or FQDN of MySQL database instance.	This parameter is optional. If no IP or FQDN is provided then no logging to MySQL server will be attempted
db-user	Username	Username for MySQL database instance
db-pass	Password	Password for MySQL database instance
Key	Any string of characters	Encryption key (may be any string of characters)
rlog	<ip or domain>, port, key	The IP address of a sending serverM instance, along with the UDP port number that the messages will be sent to, and an optional encryption key. <b>Note:</b> this parameter may be repeated as often as required.



Note that if an encryption key is used by the *SRM* system, then all instances of serverM logging to this server must use the same key. The *srml* log file (*srmllog.txt*) records the operations of the *srml* system itself in addition to the remote log messages received from the serverM instances that it is monitoring. Like the serverM log file itself, the *srml* log can be copied or deleted as required. If the *srml* system fails to find an existing log file in the location specified then it will create one.

An example configuration file is therefore:

# example configuration file for srm V0.9

```
log-location:/var/log/
screen:on
db-ip:127.0.0.1
db-user:srm_user
db-pass:gh&8shFF!
Key: 36%6sGG2*f-ioO
rlog:127.0.0.1,4000
rlog:192.168.0.34,4000
rlog:192.168.0.36,4000
rlog:192.168.0.38,4000
rlog:192.168.0.40,4002
```

In this example, the *srml* system is expecting to receive log messages from up to five separate serverM instances. The server on which the *srml* system itself is running also has a serverM instance (127.0.0.1). All serverM instances are using encryption. All serverM instances, with the exception of 192.168.0.40, are communicating on UDP port 4000.

### F3. Starting and Stopping the srm System

Starting the *srml* system is undertaken at the command line:

```
%>./srm config.txt
```

The config file location parameter is the only parameter, and is required. If no configuration file is provided then the *srml* system will abort with an error message.

If there are any errors in the configuration file then the *srml* system will also abort with an informative error message.

### F4. System Requirements

The *srml* system ships with its own required Perl libraries in the subdirectory 'lib'. In addition to these libraries, a number of standard libraries are required. The two required libraries are:

```
IO::Socket::INET
DBI
Crypt::CBC
Crypt::Blowfish
```

These can be installed from CPAN ([www.cpan.org](http://www.cpan.org)) using your favourite Perl package manager.

## F5. Database Logging

The *srm* system is designed to log remote messages to a MySQL database instance called '*srm*'. Remote log messages are stored in a single MySQL table called '*log*'. Each message is parsed into the following fields:

- *id* - a unique ID number generated by the *srm* system for each entry in the table
- *server* - the name of the remote serverM instance *server*
- *time* - a human readable string representing the time at which the log message was generated on the remote serverM instance
- *timestamp* - an '*epoch*' timestamp of when the remote message was received by the *srm* system
- *checkcycle* - the checkcycle on the remote serverM instance when the log message was generated
- *type* - the type of log message
- *message* - the body of the message

Thus the '*id*' is the primary key of the database table and will be guaranteed to be unique for every *srm* log message.

The file '*srm.sql*' is included in the serverM/*srm* distribution, and contains the SQL commands required to create the table '*log*' in the database '*srm*'. The file contents are shown below:

```
SET NAMES latin1;
SET FOREIGN_KEY_CHECKS = 0;

CREATE TABLE `log` (
  `id` bigint(20) NOT NULL auto_increment,
  `server` char(255) NOT NULL,
  `time` bigint(20) unsigned NOT NULL,
  `timestamp` char(40) NOT NULL,
  `checkcycle` bigint(20) unsigned NOT NULL,
  `type` char(12) NOT NULL,
  `message` text NOT NULL,
  PRIMARY KEY (`id`),
  KEY `servertype` (`server`,`type`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

An example table segment is shown below:

server	time	timestamp	checkcycle	type	message
Astrokan.local	1141502907	Sat 4 Mar 2006 : 20:08:27	87	status	Heartbeat: starting check-cycle
Astrokan.local	1141502908	Sat 4 Mar 2006 : 20:08:28	87	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.52 secs)
linux	1141502945	Sat 4 Mar 2006 : 18:28:23	81	status	Heartbeat: starting check-cycle
linux	1141502945	Sat 4 Mar 2006 : 18:28:23	81	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.09 secs)
Sentry.local	1141502951	Sat 4 Mar 2006 : 20:09:11	134	status	Heartbeat: starting check-cycle
Sentry.local	1141502952	Sat 4 Mar 2006 : 20:09:12	134	alarm	status rule rules check new volumes mounted occurred
Sentry.local	1141502953	Sat 4 Mar 2006 : 20:09:13	134	info	email alarm sent to someone@somewhere.com
Astrokan.local	1141502968	Sat 4 Mar 2006 : 20:09:28	88	status	Heartbeat: starting check-cycle
Astrokan.local	1141502968	Sat 4 Mar 2006 : 20:09:28	88	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.51 secs)
linux	1141503005	Sat 4 Mar 2006 : 18:29:23	82	status	Heartbeat: starting check-cycle
linux	1141503005	Sat 4 Mar 2006 : 18:29:23	82	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.09 secs)
Sentry.local	1141503013	Sat 4 Mar 2006 : 20:10:13	135	status	Heartbeat: starting check-cycle
Sentry.local	1141503014	Sat 4 Mar 2006 : 20:10:14	135	alarm	status rule rules check new volumes mounted occurred
Sentry.local	1141503015	Sat 4 Mar 2006 : 20:10:15	135	info	email alarm sent to someone@somewhere.com
Astrokan.local	1141503028	Sat 4 Mar 2006 : 20:10:28	89	status	Heartbeat: starting check-cycle
Astrokan.local	1141503028	Sat 4 Mar 2006 : 20:10:28	89	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.51 secs)
linux	1141503065	Sat 4 Mar 2006 : 18:30:23	83	status	Heartbeat: starting check-cycle
linux	1141503065	Sat 4 Mar 2006 : 18:30:23	83	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.08 secs)
Sentry.local	1141503075	Sat 4 Mar 2006 : 20:11:15	136	status	Heartbeat: starting check-cycle
Sentry.local	1141503076	Sat 4 Mar 2006 : 20:11:16	136	alarm	status rule rules check new volumes mounted occurred
Sentry.local	1141503077	Sat 4 Mar 2006 : 20:11:17	136	info	email alarm sent to someone@somewhere.com
Astrokan.local	1141503088	Sat 4 Mar 2006 : 20:11:28	90	status	Heartbeat: starting check-cycle
Astrokan.local	1141503088	Sat 4 Mar 2006 : 20:11:28	90	info	general (Service check) executed command /etc/init.d/cron start for rule
Astrokan.local	1141503090	Sat 4 Mar 2006 : 20:11:30	90	info	email alarm sent to someone@somewhere.com
linux	1141503125	Sat 4 Mar 2006 : 18:31:23	84	status	Heartbeat: starting check-cycle
linux	1141503125	Sat 4 Mar 2006 : 18:31:23	84	status	Heartbeat: check-cycle ended - no alarm messages sent (av:0.08 secs)

Figure F.1: Sample *srm* Log Table

Note that the message text is 'safe encoded' to replace characters that MySQL will not accept in a text field with safe alternatives.

## F.6 The *srm* Log File

The 'log-location' parameter within the *srm* configuration file determines where the *srm* log file will be stored. This file will contain information about the *srm* system itself (including startup and closedown times, which servers are being monitored etc.) The log file will also contain the messages received from remote serverM instances.

The file can be deleted or moved at will - the *srm* system will generate a new log file if one is not found in the expected location. This enables system administrators to undertake log file rotation and compression as required.

An example extract from an *srm* log is shown below:

```
Fri 10 Mar 2006 : 11:53:27 srm system is starting
Fri 10 Mar 2006 : 11:53:27 monitoring host '127.0.0.1' on UDP port
'4000'
Fri 10 Mar 2006 : 11:56:53 rec: {Sentry.local} Fri 10 Mar 2006 :
11:56:53 (4) [status] Heartbeat: starting chec
k-cycle
Fri 10 Mar 2006 : 11:57:53 rec: {Sentry.local} Fri 10 Mar 2006 :
11:57:53 (5) [status] Heartbeat: starting chec
k-cycle
Fri 10 Mar 2006 : 11:58:53 rec: {Sentry.local} Fri 10 Mar 2006 :
11:58:53 (6) [status] Heartbeat: starting chec
k-cycle
Fri 10 Mar 2006 : 11:59:53 rec: {Sentry.local} Fri 10 Mar 2006 :
11:59:53 (7) [status] Heartbeat: starting chec
k-cycle
```

```

Fri 10 Mar 2006 : 11:59:54 rec: {Sentry.local} Fri 10 Mar 2006 :
11:59:54 (7) [info] general (Status check) exe
cuted command cp /var/log/system.log
'/Users/david/Documents/opensource serverM/%df.log' for rule rules
{failed
  login attempts} check failed login
Fri 10 Mar 2006 : 11:59:55 rec: {Sentry.local} Fri 10 Mar 2006 :
11:59:55 (7) [info] email alarm sent to david@
port80.com
Fri 10 Mar 2006 : 12:00:55 rec: {Sentry.local} Fri 10 Mar 2006 :
12:00:55 (8) [status] Heartbeat: starting chec
k-cycle
Fri 10 Mar 2006 : 12:01:55 rec: {Sentry.local} Fri 10 Mar 2006 :
12:01:55 (9) [status] Heartbeat: starting chec
k-cycle
Fri 10 Mar 2006 : 12:02:55 rec: {Sentry.local} Fri 10 Mar 2006 :
12:02:55 (10) [status] Heartbeat: starting che
ck-cycle
Fri 10 Mar 2006 : 12:02:56 rec: {Sentry.local} Fri 10 Mar 2006 :
12:02:56 (10) [info] general (Service check) e
xecuted command /etc/init.d/cron start for rule rules cronis running
Fri 10 Mar 2006 : 12:02:58 rec: {Sentry.local} Fri 10 Mar 2006 :
12:02:58 (10) [info] email alarm sent to david
@port80.com
Fri 10 Mar 2006 : 12:03:58 rec: {Sentry.local} Fri 10 Mar 2006 :
12:03:58 (11) [status] Heartbeat: starting che
ck-cycle
Fri 10 Mar 2006 : 12:04:58 rec: {Sentry.local} Fri 10 Mar 2006 :
12:04:58 (12) [status] Heartbeat: starting che
ck-cycle
Fri 10 Mar 2006 : 12:05:58 rec: {Sentry.local} Fri 10 Mar 2006 :
12:05:58 (13) [status] Heartbeat: starting che
ck-cycle
Fri 10 Mar 2006 : 12:06:58 rec: {Sentry.local} Fri 10 Mar 2006 :
12:06:58 (14) [status] Heartbeat: starting che
ck-cycle

```

The first two entries are generated by the *srm* system, and show the startup time, and the list of servers that the system is listening to messages from. The remainder of the entries are received from the remote server (in this case the local host 127.0.0.1). These entries always contain the server name within braces. There are two timestamps - the first belongs to the *srm* system, and is the time of the entry being written to the log, and the second is received from the remote serverM instance, and is part of the remote log message.