

WatchGuard® System Manager MSS Edition WEP Setup and Operation Guide

WatchGuard System Manager - MSS Edition 7.5



Notice to Users

Information in this guide is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of WatchGuard Technologies, Inc.

Copyright, Trademark, and Patent Information

Copyright© 1998 - 2007 WatchGuard Technologies, Inc. All rights reserved.

Complete copyright, trademark, patent, and licensing information can be found in the WatchGuard System Manager -- MSS Edition User Guide. A copy of this book is automatically installed into a subfolder of the installation directory called Documentation. You can also find it online at:

<http://www.watchguard.com/help/documentation/>

All trademarks or trade names mentioned herein, if any, are the property of their respective owners.

Guide Version: 7.5

ADDRESS:

505 Fifth Avenue South
Suite 500
Seattle, WA 98104

SUPPORT:

www.watchguard.com/support
support@watchguard.com
U.S. and Canada +877.232.3531
All Other Countries +1.206.613.0456

SALES:

U.S. and Canada +1.800.734.9905
All Other Countries +1.206.521.8340

ABOUT WATCHGUARD

WatchGuard is a leading provider of network security solutions for small- to mid-sized enterprises worldwide, delivering integrated products and services that are robust as well as easy to buy, deploy and manage. The company's Firebox X family of expandable integrated security appliances is designed to be fully upgradeable as an organization grows and to deliver the industry's best combination of security, performance, intuitive interface and value. WatchGuard Intelligent Layered Security architecture protects against emerging threats effectively and efficiently and provides the flexibility to integrate additional security functionality and services offered through WatchGuard. Every WatchGuard product comes with an initial LiveSecurity Service subscription to help customers stay on top of the security landscape with vulnerability alerts, software updates, expert security instruction and superior customer care. For more information, please call (206) 521-8340 or visit www.watchguard.com.

Contents

Contents	iii
CHAPTER 1 WEP Architecture	1
WEP in the Managed Security System	2
What the WEP Does	3
WEP Daemon Functions	3
<i>controld</i>	3
<i>Wepd</i>	3
<i>Notifyd</i>	4
<i>log_server</i>	4
WEP Utilities	5
<i>wglog API</i>	5
<i>Archive client</i>	5
WebBlocker and the WEP	5
CHAPTER 2 Installing and Configuring the WEP	7
System Requirements	7
Configuring DNS for the WEP	7
<i>Add DNS to the nsswitch.conf file</i>	8
<i>Create a resolv.conf file</i>	8
Installing the WEP Packages	8
<i>Installing through FTP</i>	9
<i>Installation with CD-ROM</i>	9
<i>Installation with NFS</i>	9
<i>Installing WGtools, WGwepdoc, WGwep, and WGkitdoc</i>	10
<i>Initializing variables in the configuration file</i>	11
<i>Starting the WEP</i>	11
Configuring the WEP Server	11

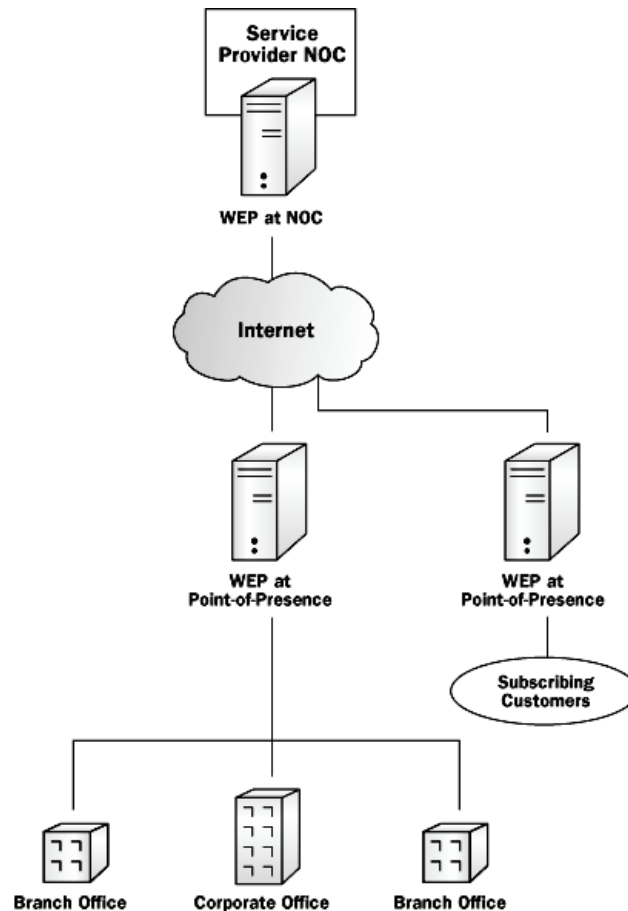
Starting and Stopping the WEP server	12
Communication Protocol	13
CHAPTER 3 Sending Commands to the WEP	15
Using Wepclient	15
<i>Using the client interactively</i>	15
<i>Using a script</i>	15
<i>Opening a connection using the MD5 hash</i>	16
Wepclient Commands	16
<i>Retrieving status</i>	16
<i>Getting and setting keys</i>	17
<i>Retrieving logs and files</i>	17
<i>Rolling over and deleting log files</i>	18
<i>Retrieving and uploading a Firebox configuration file</i>	19
<i>Management commands</i>	20
CHAPTER 4 Modifying the WEP Configuration File	21
Running wep-config-cmd	21
Command Options	22
Using the Archive Client	24
Configuring Logging	24
Custom Library for Control and Notifyd	25
Customizing Notifications	25
WepMonitor Variables	26
Using the dbfetch Command (WebBlocker)	26
Specifying a Connect Script	27
Specifying a Roll Log Script	27
CHAPTER 5 Using the Firebox Shell (fbsh) Command-Line Interface	29
Entering the Fbsh Environment	29
<i>Command Permissions</i>	29
Fbsh Syntax	30
Getting Online Help on Fbsh Commands	30
Manipulating Binary Files	31
Editing the Configuration File with Fbsh	31
Using Fbsh in Scripts	32
Fbsh Commands	33
<i>General fbsh commands</i>	34
<i>Firebox Status Commands</i>	34
<i>General Firebox commands</i>	37
<i>UDS commands</i>	37
Command Options	39

Using netdbg	40
<i>Installing netdbg components</i>	40
<i>Launching a netdbg utility</i>	40
<i>Redirecting output to TCP socket</i>	41
<i>Summary of commands</i>	41
CHAPTER 6 Using Rep_cmd to Create Reports	43
Report Output Types	43
<i>Exporting reports to HTML format</i>	43
<i>Exporting reports to NetIQ</i>	43
<i>Exporting reports to text files</i>	43
Command Syntax	44
Specifying Report Sections (Required for HTML Format)	45
<i>Report sections</i>	45
<i>Consolidated sections</i>	47
Specifying a Report Time Period (Optional)	48
Miscellaneous Parameters (Optional)	49
Example Command Lines	50
CHAPTER 7 C Language Application Program Interface (API)	51
Log Record API	51
<i>Compile/Link line</i>	52
<i>Get Log File Size</i>	52
<i>Read Log Records</i>	53
Function Descriptions	57
<i>wglog_open_local</i>	57
<i>wglog_open_remote</i>	58
<i>wglog_close</i>	58
<i>wglog_size</i>	59
<i>wglog_get_timezone</i>	60
<i>wglog_set_timezone</i>	61
<i>wglog_time_index</i>	61
<i>wglog_read</i>	63
<i>wglog_write</i>	64
<i>wglog_delete</i>	64
<i>wglog_purge</i>	65
CHAPTER 8 Custom Notification Filters	67
Notification Overview	67
Compile/Link line	67
<i>Example</i>	68
Function Descriptions	69
<i>Notify Filter</i>	69
<i>Proc Log Record</i>	69

Notification Environmental Variables	69
Index	73

WEP Architecture

The WatchGuard Event Processor (WEP) controls all communications between Global Policy Manager and a service provider's network of subscribing customers. The WEP software can be distributed anywhere throughout a service provider's network. It stores activity logs from up to a thousand or more Fireboxes and continuously monitors activity logs for specially defined trigger events. WEP software is installed separately on Sun Solaris workstations and managed centrally from Global Policy Manager.



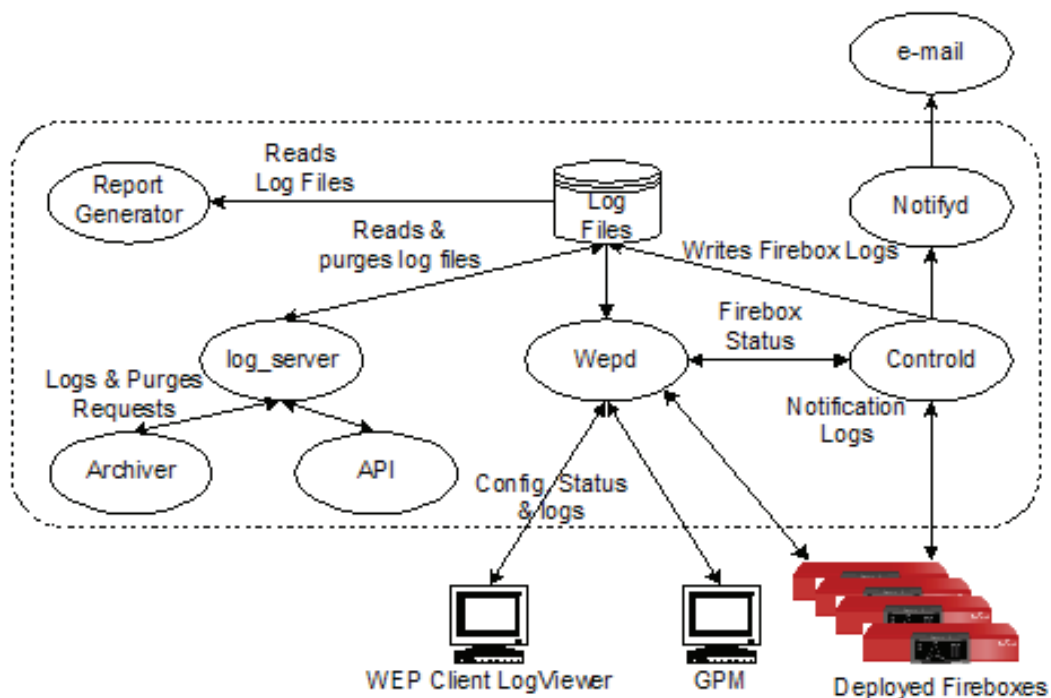
The WEP consists of the following software:

- WEP daemons: low-level programs that perform the event processing tasks. The primary daemons are control,d, wepd, notifyd, and log_server. A daemon also runs WebBlocker, the program that regulates Web surfing by site type and time of day.
- WEPtools: a toolkit of scripts and other software tools to make WEP server administration and configuration easier as well as automating Firebox reporting functions.
- WGdev: a C-language application program interface (API) that allows a C programmer to access and modify log records.

WEP in the Managed Security System

The WEP is a part of the WatchGuard Managed Security System, which is distributed among three hardware platforms. At the customer site is the Firebox, which implements a security policy derived from the configuration file stored in its flash memory. The Firebox engine is a proprietary processor board running a modified Linux kernel. At the WEP site is the WEP server running on a Sun Solaris platform. The WEP server stores and retrieves event logs from the Firebox, synchronizes the Fireboxes' clocks, issues notifications of specified events, and gathers Firebox status data for Global Policy Manager (GPM). The GPM administers the Fireboxes, creates configuration files for them, and downloads the new configurations to the Fireboxes. Included in the WEP package is a group of tools that enables the WEP operator to perform many of the same WEP operations from the Solaris console (using the command line) that are performed at GPM (using the GUI).

The following figure illustrates the basic WEP architecture along with its five process daemons—control, wepd, log_server, notifyd, and webblocker.



What the WEP Does

The WEP records and retrieves events. Each Firebox is configured to record certain events, such as port space probes and denied packets using specific protocols or ports. The Firebox encrypts logs of these events by sending a stream of log packets to its assigned WEP. The WEP receives the packets and saves them.

The GPM controls aspects of control and configuration of the Firebox via the WEP server. The GPM requests stored logs from the WEP. The GPM provides an operator with a real-time display showing which Fireboxes are currently logging to which WEPs.

Although logging of events is important, it is the retrieval of these events in a usable form that allows administrators and customers to monitor customer site security. A single WEP logs events for many Fireboxes. To monitor the security of a given customer Firebox installation, the WEP retrieves logs for a given Firebox and watches for traffic and attack patterns, attack sources, and security breaches, as well as configuration or device problems. The Firebox also logs (to the WEP) traffic patterns of the types of information passing through it.

These logs are retrieved on a Firebox-by-Firebox basis to compile reports that pinpoint attack sources or methods and configuration problems. By identifying weaknesses, the WEP helps customers develop more effective security policies.

WEP Daemon Functions

This section describes each of the WEP daemons and their individual tasks.

controld

Controld receives logs from the Firebox over the encrypted network connection and stores them in the Firebox log file and index. After the connection is formed on port 4107, controld and the logger validate the keys to use to encrypt the packets sent on the connection. This is the logging channel.

Controld also writes notification packets to notifyd. Controld loads a shared library when it starts that, by default, selects notification packets and forwards them to notifyd for further processing. This shared library mechanism allows you to write a custom shared library to select and process packets in addition to the notification logging packets.

For a WEP to receive logs from a Firebox, an IP-routable path must exist between the WEP and the Firebox. Port 4107 must be open on TCP/IP to both outgoing and incoming.

Controld knows the log directory where all Firebox logs are stored. The WEP subdirectory naming convention differentiates the log files for each Firebox. This is how one WEP server keeps track of logs coming from up to 500 concurrently logging Fireboxes.

Wepd

Wepd acts as a proxy, buffering several data flows from one host or process to another. Wepd performs the following functions:

Monitoring Firebox connections

Wepd receives status updates from controld about which Fireboxes are connected to the WEP at any given time. “Connected” means that the controld daemon on the WEP and the Firebox have established a logging session (TCP) on port 4107.

Providing connection status to GPM

GPM polls wepd to see which Fireboxes are connected.

Servicing client requests

Wepd listens to client applications via TCP/IP on port 4105. The client software sends its requests to the WEP where log files are transferred. Controld and the Firebox communicate over a single-DES encrypted channel. The control channel that wepd uses for all other connections is triple-DES encrypted. Client applications that communicate with wepd must use the MSS protocol.

Notifyd

Notifyd has a single purpose: to receive packets from the UNIX domain socket connecting it to a set of controld processes. Notifyd then calls the dynamically loaded shared library function that processes the packet. As shipped, this library function is used to send notifications. This library call is customizable to perform virtually any task based upon packet reception.

These messages are sent from the Firebox to controld, which forwards these packets to notifyd for further processing.

The security policies on the Firebox control the types of notifications for the Firebox. Administrators set service logging properties and specify notifications individually for each service. For instance, for the RealAudio service, an administrator might turn on logging for allowed or denied packets and request that the Firebox send notification on denied packets.

The controld daemon receives all log packets. It examines each packet after writing it to disk, and if the packet is a notification packet, forwards it to the notifyd daemon. The notifyd daemon examines the parameters for the service from which the notification packet originated and, if conditions are met, it launches notification scripts that send email, SMTP traps, or other forms of notification that an event has occurred.

Notifyd Notifications

When notifyd sends a message, its shared library launches a script that sends mail to the userid specified in the notifyuser file.

When notifyd receives a stream of log packets, it examines the notification interval specified in the configuration file. Because this interval is set in minutes, if the notification interval is set to two, the minimum time between two notification events is two minutes. The purpose behind this timing interval is to avoid excessive bandwidth devoted to redundant notifications. This grows in importance with the number of Fireboxes connected to the WEP.

Using ping as an example, WEP logging is turned on for allowed and denied packets, and notification is on for denied packets. The notification interval is set for one minute.

Each time the Firebox denies a ping packet, it generates a deny and a notification packet. The Firebox sends the packets to the WEP, where notifyd receives the deny packet and checks the notification interval. If more than one minute has passed since the previous notification (if any), notifyd sends an event notification.

The WEP contains only one notification interval, thus all the notification parameters for the WEP are set globally. If 20 Fireboxes log to the same WEP, all 20 share the same notification interval, and the same notification script is launched when a notification is determined to be warranted.

In this example, if the Firebox denied a ping every second, at the end of the one-minute interval, notifyd launches a script that shows a repeat count of 60 (1 per second) for that notification interval.

log_server

The log_server daemon is used as an access point for retrieving logs. One goal of the log server is to decouple client programs from the log file to modify the log file architecture without affecting the clients. Another goal of the log server is to merge log files stored on separate WEPs into one complete log file.

WEP Utilities

The following utilities are also instrumental in providing WEP functionality.

wglog API

The wglog API is used to access files directly on the local machine by internally making calls to the logdb API. The new logdb API organizes the log file data to make common requests efficiently. This logdb API does not create a circular buffer out of the log file. Log files continue to grow until disk space runs out. The disk space problem is handled by the wglog_purge/wglog_delete functions.

Archive client

The archive client uses the WGLog API to pull logs from a log server and write them to its own log database using the WGlog API's local access capability. It uses the API's delete function to remove the records from the source database. This is the recommended way to purge log files. To use the archive client, see the related manual page provided with your MSS build.

WebBlocker and the WEP

The WebBlocker database serves as a resource to enforce WebBlocker rules regarding the types of sites allowed. The WebBlocker daemon ensures that the version of the WebBlocker database stored on the WEP is the most current available. At preset intervals, the WebBlocker daemon uses the wbdb protocol to connect to the WatchGuard Web site and check the current version of the WebBlocker database. If the database on the WatchGuard site is newer than the one stored on the WEP, the webblocker daemon uses dbfetch to download the newer database to the WEP.

The webblocker daemon makes this check according to a parameter set during WEP registration. The frequency can be set from once every few seconds to once a day. If set to daily, every day the webblocker daemon sends out the dbfetch executable. The webblocker protocol is TCP/IP on port 4106. The WEP needs port 4106 open both incoming and outgoing to download the WebBlocker database.

The WebBlocker database is not the same as the one containing the rules for determining who can go to what kind of sites. This database contains information compiled through sites that continually update lists of sites and their contents. WatchGuard compiles that information into a binary database so it can be moved back and forth efficiently.

If an administrator does not want port 4106 open all the time, he or she can manually launch the database application on a weekly basis to check the WatchGuard site for updates to the database. If an administrator wants a frequent check and a manual launch then cron, or some other scheduling software, can launch dbfetch at a desired interval.

Periodically, the Firebox opens port 4103 to the WebBlocker database and makes a request very similar to the one that launches the update query between the WEP and the WatchGuard Web site. The webblocker daemon on the WEP listens to port 4103. The webblocker daemon returns a small header containing the MD5 sum of the WebBlocker database. When the Firebox receives this information, it checks to see if it matches the database stored in the memory. If the information it receives is different, or if the Firebox is booting for the first time and has nothing in memory, it follows with an immediate request to download the entire database using a binary protocol on port 4103. Port 4103 should be set to allow incoming and outgoing between the Firebox and the WEP.

Installing and Configuring the WEP

This chapter describes the procedures required to install and configure the WEP.

System Requirements

WatchGuard recommends that the WEP platform be dedicated solely to WEP operation, and that nothing run on the WEP host except WEP software. A particular recommendation is to not run X Windows or other servers on the WEP host, because these pose an unnecessary security risk.

The following table shows the minimum system requirements for WEP hosts.

Feature	Minumum requirement
Processor	Sun Ultra5 Sparc workstation
Processor speed	270 Mhz
RAM	128 MB
Operating system	Solaris 2.6 or 2.8 CoreInstall
Disk capacity	4GB However, this varies according to the number of Fireboxes connected to the WEP. SCSI disks are highly recommended.
/usr partition	500MB minimum
Other	Ethernet interface; CD-ROM drive
Additional recommended Sun patches	For current information on Sun patches, go to: http://sunsolve.sun.com/patches

Configuring DNS for the WEP

The current WEP supports Solaris version 2.6 or 2.8 software. The default installation of of these versions, however, does not offer users the option to configure DNS, which is required for basic Internet name res-

Installing the WEP Packages

olution. You must complete the DNS configuration before installing the basic Solaris software. However, DNS is not required for the WEP to function.

- 1 Change to the `/etc` directory:
`cd /etc`
- 2 Using the `ls` command, check for the presence of the `defaultrouter` file. If it does not exist, use a text editor to create a file with the following contents:
 - The IP address of the default gateway for the network where the WEP software resides.Save the file and exit.

Add DNS to the `nsswitch.conf` file

- 1 Open the `nsswitch.conf` file in a text editor.
- 2 Locate the line in the file that begins with the string `hosts:` and add the string `files dns` after it as follows:
`hosts: files dns`
Add a list of places for the computer to locate names to associate with addresses:
`hosts: files dns location [location] ...`
For example, if you are using the name services `nis` and `nis+`, the line would appear as follows:
`hosts: files dns nis nis+`
- 3 Save the file and exit.

Create a `resolv.conf` file

- 1 Change to the `/etc` directory:
`cd /etc`
- 2 Enter the string `search` on the first line of the file, followed by a list of active domains. These domains are usually local to your company.
Example: `search ssl.watchguard.com hackers.watchguard.com watchguard.com`
- 3 On the next line, enter the string `nameserver` followed by the primary DNS server on the network:
Example: `nameserver 192.168.1.6`
- 4 On the next line, enter the string `nameserver` followed by the secondary DNS server on the network, if the network has one.
- 5 Save the file as `resolv.conf` and close the file.
- 6 Enter `/etc/init.d/inetinit`
- 7 DNS should now be active. Test it by entering:
`nslookup hostname`
where `hostname` is the name of a known host on the network.

Installing the WEP Packages

As a part of the installation process, you are prompted for user and group (file) ownership information. WatchGuard recommends that you log in to the MSS system as a non-root user such as `mssuser`. Install all packages into the same base directory. When the installation is complete, all files in the WG packages are then located in the following directory:
`/base_directory/WGwep/*.`

You can access the installation package from several installation sources: FTP, CD-ROM, or NFS. Use the method most convenient for you.

Root access is required for a successful installation.

- 1 Become root
su -
- 2 Determine the installation source (CD-ROM, FTP, or NFS).

Installing through FTP

- 1 Use the following FTP commands to retrieve the compressed file:

```
wep1# cd /path_to_tar_files
wep1# ftp ftp.watchguard.com
Name: ftp_loginname_supplied_by_WatchGuard
Password: ftp_password_supplied_by_WatchGuard
>get MSS6.0.tar.Z
>quit
```

- 2 Use the following command to unpack the file before proceeding with the installation process:

```
wep1# cd /var/tmp
wep1# zcat /path_to_tar_files/MSS6.0.tar.Z | tar xfv -
wep1# pkgadd -d MSS6.0
```

Installation with CD-ROM

Solaris 2.6 or 2.8 installation with the Sun Volume Management Services

WatchGuard MSS should automatically be available under `/cdrom/mss6.0` after you insert the WatchGuard Event Processor CD-ROM.

Solaris 2.6 or 2.8 installation without the Sun Volume Management Services

If you are using a CD-ROM, insert it into the CD-ROM drive and execute these commands:

```
wep1# mkdir -p /cdrom/mss
wep1# mount -F hsfs -o ro /dev/dsk/c?t?d?s0 /cdrom/mss
wep1# cd /cdrom/mss
wep1# pkgadd -d MSS6.0
```

Installation with NFS

If no CD-ROM drive is attached to the Sun Solaris box, mount the WEP CD-ROM on a suitable NFS server and export the CD-ROM to the Solaris box. A suitable NFS server has the capability to mount CD-ROMs with Rock Ridge extensions. Please consult the Sun Support Channel for more information on Rock Ridge extensions and NFS exports.

WatchGuard WEP packages are interactive. Do not use the `-n` flag when implementing a `pkgadd`.

The following is an example of typical screen output :

The following packages are available:

```
WGdev      WGdev: WatchGuard Development Kit
           (sparc) 6.0
WGkitdoc   WEP: WatchGuard MSS Toolkit Man Pages
           (sparc) 6.0
WGtools    WEP: WatchGuard MSS Toolkit
           (sparc.sun4u) 6.0
WGwep      WEP: WatchGuard Event Processor
           (sparc.sun4u) 6.0
```

WGwepdoc WEP: WatchGuard Event Processor Man Pages
(sparc) 6.0

The following prompts appear:

```
Select package(s) you wish to process (or 'all' to process all packages).  
(default: all) [?,??,q]:  
Enter user to own WGdev files (root)  
Enter group ownership for WGdev files (root)  
Enter path to package base directory [?,q] /opt
```

The following output appears:

```
Using </opt> as the package base directory.  
## Processing package information.  
## Processing system information.  
## Verifying disk space requirements.  
## Checking for conflicts with packages already installed.  
## Checking for setuid/setgid programs.
```

Installing WGdev: WatchGuard Development Kit as <WGdev>

```
## Installing part 1 of 1.  
/opt/WGdev/PLATFORM  
/opt/WGdev/doc/logops.txt
```

<...>

```
/opt/WGdev/man/man3/wgerr.3  
[ verifying class <none> ]
```

Installation of <WGdev> was successful.

Installing WGtools, WGwepdoc, WGwep, and WGkitdoc

When you install these packages, use a non-root login (and group) such as wepuser for file ownership. This login attempts to run the log-roll, connect, and notification scripts because of its permission settings.

The installation of the WGWep package asks whether or not you want the WEP daemons to automatically start and stop at system startup or shutdown.

To install the other packages on the WEP server, just repeat the procedure for each package that you installed (WGtools, WGwepdoc, WGwep, and WGkitdoc). However, start the WEP process running in the background as root.

After responding to one or more prompts, the following output appears:

```
The following files are being installed with setuid and/or setgid  
permissions:  
/etc/init.d/wep <setuid root setgid root>
```

```

/opt/WGwep/bin/oob.sh <setuid root setgid root>
/opt/WGwep/bin/oobconfig.sh <setuid root setgid root>
/opt/WGwep/bin/start-oob <setuid root setgid root>
/opt/WGwep/bin/wep-config-cmd <setuid root setgid root>

```

When asked whether you want to install the files as setuid/setgid files, respond with `y`.

When the `Select package(s)` prompt appears again, enter `q` to quit.

To adjust the verbosity of the installation, refer to the admin man page.

Initializing variables in the configuration file

You must run the `wep-config-command` utility after installing the WEP. The WEP will not function unless you run this command. For information on how to run `wep-config-command`, see “Running `wep-config-cmd`” on page 21.

Starting the WEP

After the WEP installation process is complete, you need to start the WEP:

```
wep#1 /etc/init.d/wep start
```

Verify the installation with these commands:

```
wep1# wepclient -
> OpenConn 127.0.0.1 rw
```

Where `rw` is the WEP password. The following output appears:

```

Opening connection
Trying triple DES (port 4105)
Connected to 127.0.0.1, authenticating...authenticated
> StatusVersion
Getting Wep version.
status_version_rsp

StatusVersion=====
Build Number:?.?.B??? MSS Version:6.0

> CloseConn
Closing connection
> quit
wep1#

```

Configuring the WEP Server

After you install the WEP, you must configure it. Log in using the same account you used for installation. For example, if you installed the WEP software as `root`, log in as `root`.

- 1 Determine where the package is installed:

```
wephost# pkgparam WGwep 'BASEDIR'
wephost# /opt/WGwep/bin/wep-config-cmd
```

- 2 To make administration tasks easier, put the WGwep utilities into the shell command search path by putting the following into `.profile`:

```
PATH=$PATH:`pkgparam WGwep BASEDIR`/WGwep/bin; export PATH
```

Starting and Stopping the WEP server

- 3 To configure the WEP services, run `wep-config-cmd` as in the example below:

```
wephost# wep-config-cmd
```

The following output appears:

Package Base: `/opt/WGwep`

LineField Name: Field Value

```
-----  
1:   Package base directory: /opt/WGwep  
2:           Wep IP: aaa.xxx.yyy.zzz  
3:   Logging sub directory: /opt/WGwep/logs  
4:           Status filename: /opt/WGwep/status_file  
5:           Wep UDS filename: /opt/WGwep/status_uds  
6:   Notifier UDS filename: /opt/WGwep/notify_uds  
7:           Wep encryption key: gpm_encryption_key  
8:           Log encryption key: log_storage_encryption_key  
9:           Notify script file: /opt/WGwep/scripts/mail_report  
10:  Notify script username: nobody  
11:  Logging connected script: /opt/WGwep/scripts/simple_connect  
12:  Logfile backup script:  
13:  Connect script username: wepuser  
14:  Backup script username: wepuser  
15:           Cache syslogs: no  
-----
```

Enter (E)xit and save, (Q)uit without saving, or the line number to change:

This screen shows the configuration of the WEP IP address, WEP and Log keys, and the location of various files. WatchGuard recommends specifying a logdir on a different file system (partition) than the one the installation resides on. In item 3, if you do not have sufficient disk space for logging under `/opt/WGwep/logs` (in this example), define a different directory tree here. You must provide correct data for lines 2, 7, and 8 to start the WEP:

- Line 2 is the IP address of the WEP server configured in the Fireboxes.
- Line 7 is the encryption key used by GPM to connect with the Solaris WEP services.
- Line 8 is the encryption key used by the Fireboxes to connect with the `controld` daemon for log storage.
- Lines 10, 13, and 14 are ignored in non-root installations. The scripts run with an effective UID of the `wepuser`. If another UID is desired, the file ownership and (`setuid`) permission must be adjusted in the scripts.

Starting and Stopping the WEP server

After the WEP services are configured, `wepuser` starts, stops, and restarts the services. The three line commands, respectively, are:

```
wepuser# /etc/init.d/wep start  
wepuser# /etc/init.d/wep stop
```

```
wepuser# /etc/init.d/wep restart
```

The following are interdependencies to consider when running a WEP server.

Time Synchronization

Fireboxes synchronize their time setting with that of the WEP. Because the accuracy of the logs depends upon correct time, it is essential that the WEP maintain accurate time. Use a service package such as SUNWxntp to keep the time on the WEP running consistently. This service can also be configured for shared secrets. If a site uses two or more WEP servers, one WEP server is configured as the master NTP server, with the other WEP server acting as the slave. This setup ensures proper time synchronization for Firebox logs when the Firebox log location fails over from one WEP to another.

Configuring Sendmail

A feature of MSS is mail notification. However, configure sendmail for outgoing only, setting it to run under cron rather than in daemon mode. Do not configure it for incoming mail or for relaying mail.

Communication Protocol

When GPM sends a request to the WEP using the MSS protocol, the request is sent to the Firebox. The Firebox replies back to the WEP, translating the reply into MSS protocol and sending it to GPM. These events require that the WEP establish an encrypted channel between GPM and the WEP. For this to happen, the client must know the WEP control command key. These keys are stored in `/etc/WGwep.conf` on the WEP.

The WEP key is different from the Firebox keys. The Firebox keys allow the WEP to talk to the Firebox while the WEP client prompts for the Firebox keys. The GPM stores the command keys for all the Fireboxes with which it communicates in the NOC database on the WEP. The GPM stores Firebox keys in the NOC database on the GPM host. These keys are not stored on the Firebox.

To manage a Firebox, an administrator needs to know the read-only and the read/write keys entered during Firebox registration. For instance, to write a new security configuration to a Firebox, the WEP client issues the `PutConfig` command. This command sends the Firebox configuration key to the WEP as part of the command. `wepd` uses the key to establish an MPF with the Firebox. It then uses the MPF `put-file` command to save the configuration to the Firebox, reboot the Firebox, and load the new security policy.

Some commands between the WEP and the Firebox require the read-only password. For example, the `status bar` command that retrieves information about the current status of the Firebox is a read-only operation. The `fbsh` command `status firebox` contains information about which Firebox to check. For more information about `fbsh`, see Chapter 5, “Using the Firebox Shell (fbsh) Command Line Interface.” In this case, `wepd` sends the read-only key to establish an MPF session with the Firebox. The Firebox replies using MPF format. That MPF is interpreted by the `wepd` daemon, and the message is translated into MSS protocol for the reply to the MSS client (the GPM, in this case).

To communicate with the WEP, you use a command-line client called `wepclient`. `Wepclient` sends commands to the WEP server and enables the operator to perform many of the WEP administration tasks typically performed from the Windows NT client console. All traffic between `wepclient` and the `wepd` daemon are encrypted with 3DES.

Using Wepclient

You can use the client either interactively or through a script.

Using the client interactively

To use the client interactively, at a WEP terminal, submit the following two commands:

```
% wepclient -  
% OpenConn WEP_hostname WEP_password
```

Where:

- *WEP_hostname* = the name or IP address of the WEP to which you are connecting
- *WEP_password* = the password required for access to that host

You can then use the `wepclient` commands described in “Wepclient Commands” on page 16.

Use the following command to close a `wepclient` session:

```
% CloseConn
```

Using a script

To submit `wepclient` commands using a script, use the `wepclient` command as follows:

```
% wepclient input_script_filename [output_filename]
```

Where:

- *input_script_filename* = the script that contains the list of commands available for `wepclient` to execute. You can have different command script sets available according to function or permission level required to execute.
- *output_filename* = an optional argument to write the results of the input script to the file you name at `output_filename`. If you do not specify this argument, `wepclient` writes to the

Wepclient Commands

standard output (usually your workstation screen). By writing to a file, you can collect status and error messages for a wepclient session.

The script named in *input_script_filename* must call the `OpenConn` command before using any of the commands described in “Wepclient Commands,” below:

```
OpenConn WEP_hostname WEP_password
```

Where:

- *WEP_hostname* = the name or IP address of the WEP to which you are connecting
- *WEP_password* = the password required for access to that host

Follow the wepclient commands with the following command to close the wepclient session:

```
% CloseConn
```

Opening a connection using the MD5 hash

Whether you are using wepclient commands interactively or through a script, you can use the following command, as an alternative to `OpenConn`, to open a connection to the WEP using the MD5 hash of the WEP’s password:

```
OpenConnWithHash WEP_hostname WEP_password
```

Where:

- *WEP_hostname* = the name or IP address of the WEP to which you are connecting
- *WEP_MD5_password* = the MD5 hash of the password required for access to that host

Wepclient Commands

You can use wepclient commands to determine WEP and Firebox status, get and set encryption keys, retrieve logs, roll over and delete log files, manage Firebox configuration files, and manage wepclient.

Retrieving status

You can use several commands to determine the status of various WEP server parameters on the Firebox.

Retrieving the MSS version

Use the `StatusVersion` command to retrieve the current build number and MSS version number of the WEP:

```
StatusVersion
```

Retrieving the Firebox list

Use the `StatusWep` command to get a list of Fireboxes attached to the WEP:

```
StatusWep
```

Retrieving Firebox status information

Use the `StatusFB` command to retrieve the specified Firebox’s status information. Use this command whether or not the Firebox is connected to the current WEP:

```
StatusFB firebox_ip status_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox for which you are retrieving information
- *status_passphrase* = the status passphrase for that Firebox

Retrieving Firebox software configuration

Use the `StatusComponents` command to retrieve a list of the software components that are installed on a specified Firebox:

```
StatusComponents firebox_ip status_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox for which you are retrieving information
- *status_passphrase* = the status passphrase for that Firebox

Retrieving a list of active and available log hosts

Use the `StatusLoghosts` command to retrieve a list of all available log hosts for a specific Firebox and the log host it is currently logging to:

```
StatusLoghosts firebox_ip status_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox for which you are retrieving information
- *status_passphrase* = the status passphrase for that Firebox

Getting and setting keys

These commands retrieve or enable you to set the encryption key for the logging channel between a Firebox and the WEP.

Getting the logging channel encryption key

Use `GetLogKey` to get the IMD5 key used to encrypt the logging channel:

```
GetLogKey
```

The command returns the key to your standard input unless you specified an output file when you began the wepclient session.

Verifying the passphrases for a Firebox

Use `SetKeys` to verify the character strings that you believe to be the status and configuration passphrases:

```
SetKeys firebox_ip status_passphrase configuration_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox whose passphrases you want to verify
- *status_passphrase* = the status passphrase of the Firebox
- *configuration_passphrase* = the configuration passphrase of the Firebox

Retrieving logs and files

Retrieving a Firebox's logs from a specific time period

`GetLocalLog` retrieves a specific Firebox's logs that occurred within a specific time period:

```
GetLocalLog firebox_ip start_time end_time local_logfile
```

Where:

- *firebox_ip* = the IP address of the Firebox whose logs you want to retrieve
- *start_time and end_time* = the number of seconds passed since 00:00:00 AM
- *local_logfile* = the file into which you want to place the retrieved logs

Consolidating logs spread among multiple WEPs

The `GetLog` command enables you to retrieve the logs for a specific Firebox for a specified time period. If this Firebox has multiple WEP servers for failover, you can use this command to consolidate the logs from all WEPs if you supply each WEP's IP address and key. You must supply the encryption keys in the syntax `wep=key` where the `wep` is the IP address and `key` is the MD5 WEP key for that particular WEP server. To make the results manageable, or to gather data to create a specific type of report, you can further specify which data types to retrieve from the Firebox's logs.

Note

This command is deprecated. The archive client is the preferred alternative to this command.

The syntax of the command is:

```
% GetLog firebox_ip start_time end_time local_logfile [type1:] wep=key  
wep=key ...
```

If you do not specify the data *type* fields, this command returns all logs. Otherwise specify one or the other of these types but not both:

```
LOGREC_SYSLOG_STR  
SEND_ALL_RECORDS
```

The following is an example of how the `GetLog` command might be used:

```
GetLog 192.224.128.11 9811 128343987929346 SEND_ALL_RECORDS  
144.158.221.14=12832*98had983h983hfd93~
```

Retrieving a log file for local use

The `GetLogFile` command enables you to retrieve the log file for a specific Firebox and write it to a new local log file. The syntax for the command is:

```
GetLogFile firebox_ip logfile_name [local_logfile_name]
```

Where:

- *firebox_ip* = the Firebox's IP address
- *logfile_name* = the name of the log file located in the WEP's Firebox IP directory
- The file is stored in the file `logdb` by default or in the file name you specify in the variable *local_logfile_name* if you choose

For example, to retrieve the log file named `192.224.128.11.logdb` for the Firebox with IP address `192.224.128.11` and then write the results to a local logfile named `128.11.log`, enter:

```
GetLogFile 192.224.128.11 192.224.128.11.logdb 128.11.log
```

Rolling over and deleting log files

Wepclient provides commands to maintain your active and archived log files. Rolling over a log file involves moving the current log file to an archived file and starting a new log file to receive new logs. The `wepclient` command `RollLogFile` instigates the rollover process. `DeleteLogFile` allows you to manually clean up excess log files in a given directory.

Rolling over a log file

Use the `RollLogFile` command to move the active `logdb` file to `logdb.bak` and start a new `logdb` file:

```
RollLogFile firebox_ip
```

Where:

- *firebox_ip* = the IP address of the Firebox whose logs you are rolling over

Deleting a log file

Wepclient's `DeleteLogFile` command is not the same as the “rm” command. If you do not specify a file name argument for `DeleteLogFile`, it deletes all log files in the current directory except for the default MSS log file, `logdb`. Use this command with discretion and name your arguments carefully.

The syntax for deleting log files is:

```
DeleteLogFile firebox_ip [logfile_name]
```

Where:

- *firebox_ip* = the IP address of the Firebox whose logs you are deleting
- *logfile_name* = (optional) the name you enter to specify which file to delete. If you do not specify this argument, then `DeleteLogFile` deletes all log files in the Firebox's directory except for the one named “logdb”.

Retrieving and uploading a Firebox configuration file

Use these commands to start a new configuration in a Firebox and activate it, or to recover a previous configuration and restart a Firebox.

Retrieving the configuration file for a Firebox

Use the `GetConfig` command to retrieve a Firebox's configuration file. The command is also used to make sure a specific Firebox has the latest configuration file in its flash memory. `GetConfig` retrieves the configuration file for a Firebox using its IP address. You can optionally specify a known configuration file, which `GetConfig` compares to the configuration file on the Firebox. If the two files are different, the configuration file in the Firebox is returned. If no lines are returned, you know that the configuration file you specified is the same as the active one on that Firebox.

The syntax for this command is:

```
GetConfig firebox_ip [previous_configfile] status_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox
- *previous_configfile* = (optional) the name of the configuration file on the local host that you want to compare to the one in the Firebox's flash memory
- *status_passphrase* = the status passphrase for that Firebox

Transferring a new configuration file to a Firebox

The `PutConfig` command uploads a configuration file onto the named Firebox. To be certain that you are uploading the intended configuration, you must provide both the new and previous configuration file names for `PutConfig` to compare. This assures that the previous configuration file you think you are replacing is actually the one currently on the Firebox.

```
PutConfig firebox_ip configfile previous_configfile configuration_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox you want to reconfigure
- *configfile* = the configuration file you want to upload as the new configuration
- *previous_configfile* = the local copy of the configuration file currently on the Firebox. If `previous_configfile` does not match the configuration file currently on the Firebox, the command returns an error.
- *configuration_passphrase* = the status passphrase required to make any changes to that Firebox

Rebooting a Firebox

The `Reboot` command causes the designated Firebox to reboot and therefore read and use the configuration in the user partition of the flash memory. You must reboot a Firebox whenever you change the configuration loaded into a Firebox's user partition of flash memory so the Firebox can read and use the new configuration.

```
Reboot firebox_ip configuration_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox
- *configuration_passphrase* = the configuration passphrase to that Firebox

Restarting a Firebox

To attempt to restart a Firebox without rebooting, enter:

```
Restart firebox_ip configuration_passphrase
```

Where:

- *firebox_ip* = the IP address of the Firebox
- *configuration_passphrase* = the configuration passphrase to that Firebox

Management commands

Wepclient offers commands for interactively pausing a command, putting a process to sleep, and timing out a process.

Pausing wepclient

To pause wepclient until you press a key, enter:

```
Pause
```

Putting wepclient to sleep

To put wepclient to sleep:

```
Sleep seconds
```

Where:

- *seconds* = the number of seconds (expressed as a positive integer) WEPclient is to sleep before it reactivates

Setting a timeout for wepclient commands

To set the default timeout that some of the commands use, enter this command:

```
Timeout timeunits
```

Where:

- *timeunits* = five seconds long

For example, to have the `RollLogFile` command time out after 30 seconds, enter:

```
Timeout 6
```

Modifying the WEP Configuration File

The WEP configuration file is called WGwep.conf. Because all WEP daemons and tools have a common view of the WEP environment, they share a common configuration file. This chapter describes all options recognized by the file including which arguments each expects, what the option does, and any default that may be assumed if the option is not used.

Running wep-config-cmd

You run the wep-config-cmd utility after an installation or upgrade to initialize variables in the WGwep.conf file. This utility modifies common configuration variables, most of which are used for system administration. You must run this utility after an installation even if no changes are required.

Command syntax is as follows:

```
wep-cmd-config [config_file]
```

Where:

config_file = The configuration file you want to modify.

Screen output similar to the following appears after you submit wep-config-cmd:

Package Base: /opt/WGwep

Line Field Name: Field Value

```
-----  
1: Package base directory: /opt/WGwep  
2: Wep IP: 192.168.49.80  
3: Logging sub directory: /opt/WGwep/logs  
4: Status filename: /opt/WGwep/status_file  
5: Wep UDS filename: /opt/WGwep/status_uds  
6: Notifier UDS filename: /opt/WGwep/notify_uds  
7: Wep encryption key: 000102030405060708090a0b0c0d0e0f  
8: Log encryption key: 01325e3351f355654a653e5d2e65f354  
9: Notify script file: /opt/WGwep/scripts/notify_script  
10: Notify script username: wepuser
```

Command Options

- 11: Logging connected script: /opt/WGwep/scripts/simple_connect
- 12: Logfile backup script: /opt/WGwep/scripts/backup_script
- 13: Connect script username: wepuser
- 14: Backup script username: wepuser
- 15: Cache syslogs: no

Enter (E)xit and save, (Q)uit without saving, or the line number to change:

Script file variables

Most relevant to customization are the script file variables on line numbers 9–12, above. To change the names or enable the script files, enter the full directory path to a script file. To disable a script from running, remove the script file name and path.

Modifying other WGwep.conf variables

Other variables not accessible from this utility must be modified directly by editing the /etc/WGwep.conf file in a text editor such as vi.

To enable debugging for the wepd process, remove the comment character (#) from the following using a text editor. Note, however, that activating this variable decreases WEP performance and fills the syslog file.

```
# server.wepd.debug_options: DBG_LOW DBG_TEST
```

Command Options

You can use a number of options with wep-config-cmd. Argument types are as follows:

- *boolean*—Can be one of the following: true, false; yes no; on, off; 1,0
- *string*—Any string of characters.
- *list*—Any sequence of strings separated by a white space.
- *pathname*—A fully qualified pathname, including the file name if applicable. Examples are /opt/WGwep/logs and /opt/WGwep/scripts/simple_connect
- *integer*—A decimal-value integer. The only allowable punctuation is a dash symbol (-) to signify negative numbers.

Command options are as follows:

```
server.wepd.debug_options: list
```

Sets the logging level of the wepd daemon. You can use values DBG_LOW and DBG_TEST for diagnosing WEP problems. The wepd daemon uses syslog facility local0 for logging. Use this option for debugging purposes only.

```
controld.backup_script: pathname
```

When set, this option causes this script to execute and passes the name of the old logdb filename as a parameter. This provides an archiving function for these files.

```
controld.connect_user: username
```

This option is active only when the WGwep package is installed with the root user as the owner of the files contained by the package. The administrator is prompted for the user and group ownership when the package is installed. The connect script is executed as the

username when this option is set. If not set, the connect script runs under the username of “nobody.”

`controld.rolllog_user`: username

This option is active only when the WGwep package is installed with the root user as the owner of the files contained by the package. The administrator is prompted for the user and group ownership when the package is installed. The backup script is executed as the username when this option is set. If not set, the connect script runs under the username of “nobody.”

`controld.cache_syslogs_locally`: boolean

When true, causes all syslogs received from each Firebox to be saved into a local syslogd-like text file within the logdb directory tree.

`controld.cleanup_timeout`: integer

The number of seconds allotted until the master controld process cleans up after any forked-off worker processes have exited (these are commonly referred to as “zombie” processes).
controld default: 5

`controld.connect_script`: pathname

The file that will be launched for every successful connection and disconnection of a Firebox. If this option is missing or empty, nothing will be launched. Installation default:
package_base/scripts/simple_connect

`controld.shared_lib.name`: pathname

The shared library name that controld uses to call any user-defined functions. The default value is notify_filter.so, and is installed in package_base/lib/notify_filter.so.

`controld.shared_lib.notif_funcname`: string

The name of the function that controld calls after a new log record is received from the Firebox. The default value is notify_filter. If the return value is 0, the log record is dropped. Otherwise, it is forwarded to notifyd.

`controld.debug`: boolean

If true, this indicates that extra debugging information from controld should be displayed.

`controld.debug_dump`: boolean

If true, this causes all packets sent and received by controld to be displayed. This requires compiled-in support. When enabled, it slows controld’s performance significantly.

`controld.fd_limit`: integer

This limits the maximum number of file descriptors each process under controld can use (up to the hard limit set within your system). Unless you are sure the system call select can handle more, setting this above 1024 can produce unexpected results. controld default = 1024

`controld.interactive`: boolean

When true, this indicates that controld should not run as a daemon process. Use this option for debugging purposes only.

`controld.key`: string

A sequence of 32 hex characters used to authenticate and encrypt all log packets received by controld. Each pair of characters represents the hexadecimal value of a byte in the MD5

sum of the original key created by `wep-config-cmd`. The control default is `f1f2f3f4f5f6f7f8f9f0fafbfcfdfeff`.

`controld.max_connections`: integer

The number of connections allowed against any worker at one time. For example, if this is set to 50, each worker is allowed 50 connections before refusing others. If one worker has reached its maximum but others have not, the new connection is taken up by a worker that still has connections remaining. The maximum number of simultaneous connections that can be handled is the product of this property's value multiplied by the `controld.num_workers` value. The control default is 50.

`controld.num_records`: integer

This is the maximum number of records that will accumulate in a logdb file. The log file will roll when this limit is reached. The default is eight million (8,000,000).

`controld.num_workers`: integer

This is the number of workers that should be executed to handle incoming logging connections. This property's value multiplied by the `controld.max_connections`' value will result in the maximum number of simultaneous workers. The default is 20.

Using the Archive Client

The archive client queries and consolidates Firebox log records both locally and remotely across many WEP servers. The archive client is written with the WGLog API; source code is provided for illustration purposes in the WGdev package to give users the ability to customize archiving strategies.

The stock archive client has these options:

- Query and print size (total number of log records on all WEPs for a particular Firebox)
- Read log records by numerical or time indices of a particular Firebox
- Print log records (abbreviated form)
- Write log records to file
- Mark log records for deletion
- Purge log records (Purge requests are ignored if fewer than 80 percent of the log records are marked for deletion.)

Note

For all other information about using the Archiver, refer to the man page "archive," which lists and describes each of the line commands available in the WEP Client.

Configuring Logging

The following two variables in the `WGwep.conf` file are used to balance the number of control processes that are accepting connections from Fireboxes as well as the number of Fireboxes serviced by each control:

```
controld.max_connections: 50
controld.num_workers: 10
```

The formula (`max_connections * num_workers`) gives the total number of Fireboxes from which this WEP accepts connections. You will probably have to adjust these numbers based upon your experience with the logging load on the WEP. For example, a configuration set to `num_workers = 1` and `max_connections = 500` tends to be CPU-bound as the load increases. Likewise, `num_workers = 500` and `max_connections = 1`, increases memory usage and might require that the WEP spend a significant percentage of its time swapping. Balancing the configuration to spread out the number of connected Fireboxes per process provides the best results. With `max_connections = 50` and `num_workers = 10`, it is possible to support 500 connections spread over 10 processes.

Three variables debug the controld process. By default these are commented out or not defined. Activating these variables can seriously degrade the WEP's performance:

```
controld.printf: yes
controld.debug: yes
controld.debug_dump: yes
```

Custom Library for Controld and Notifyd

Controld and notifyd can be configured to use a custom-built library. To configure controld to use this library you must set the configuration file variable to a custom built library. The default setting is:

```
controld.shared_lib.name: /opt/WGwep/lib/notify_filter.so
```

Customizing Notifications

Controld calls a `notify_filter` function defined in a shared library to determine which log records are forwarded on to notifyd. Notifyd in turn calls a `process_log_record` function defined in another shared library, which may be the same as the one used by controld, to actually process the log records. The default notify filter shared library is provided with source code in the WGdev package. This allows users to customize the specific behavior desired: which log records are processed and how they are processed if the Firebox is configured to notify a tagged record.

The default notify filter shared library filters only notification log records and forwards them to notifyd. Notifyd, in turn, launches the script `notifyd.script` by `notifyd.notify_user` based on the interval `notifyd.interval` using the shell `/bin/sh`. This script is customizable. There are two sample scripts provided in the WGwep package: `mail_report` and `notify_script`, which can be modified or used as starting points for building a customized notify script.

Specifying a custom-built library

To configure notifyd to use the custom-built library, set the following configuration file variable to the custom-built library (default setting shown):

```
notifyd.shared_lib.name: /opt/WGwep/lib/notify_filter.so
```

The default `notify_filter` library uses the following four variables:

- `notifyd.interval: 1`

The `notifyd.interval` parameter is expressed in minutes. This field controls the number of duplicate notifications sent within an interval.

- `notifyd.mailhost: hostname`

The `notifyd.mailhost` field is the mail server.

- `notifyd.mailto: username@hostname`

WepMonitor Variables

The `notifyd.mailto` field is the email address of the user that receives the email notification.

- `notifyd.notify_user`: username

This parameter names the user to receive the notification.

Debugging the notifyd process

Three variables debug the `notifyd` process. By default these are commented out or simply not defined. Because these variables are debugging tools, activating them compromises WEP performance:

```
notifyd.debug: yes
notifyd.debug_dump: yes
notifyd.printf: yes
```

WepMonitor Variables

The WepMonitor utility monitors the status of the WEP and generates an email message when the disk drive fills with log records. Three configuration parameters are used to control this utility:

- `wepmonitor.alerttypes`: MAIL

Selects either MAIL or SNMP to send an email message or a generic (linkdown) SNMP trap.

- `wepmonitor.percentfull`: 90

The threshold over which `wepmonitor` begins to generate messages. The `wepmonitor` also polls the `wepd` process to verify that `wepd` is running.

- `wepmonitor.pollinterval`: 60

The polling frequency in seconds; if set to zero the utility exits.

Using the dbfetch Command (WebBlocker)

Dbfetch is a command-line utility that connects to the specified host/port and downloads the WebBlocker database. This command creates a new file named `webblocker.db`. The previous WebBlocker database file is then named `webblocker.old`. The WebBlocker program launches `dbfetch` by default once per day.

Command syntax is as follows:

```
dbfetch [-debug] hostname [port]
```

where:

`-debug` = enables debug output

hostname = Either the DNS name or the IP address of the server that has the WebBlocker download server running on it.

port = the number of the port on which to receive the download. The default is port 4103.

WatchGuard strongly recommends using the default values in the `wb_config` file and using `dbfetch` to obtain the latest WebBlocker database.

The default values are:

- `AutoDownload`: 1
- `WB_DB_Filename`: `webblocker.db`

Specifying a Connect Script

Users can specify a shell script that executes whenever a Firebox connects to or disconnects from the WEP (controld) by way of the configuration variable `controld.connect_script`. The shell `/bin/sh` is executed to run the script as user “`controld.connect_user`”. The following environment variables are set by the parent process when this script is launched:

`mz_firebox`
IP address of the Firebox

`mz_notifier`
String specifying either connect or disconnect

If no script is specified, no script is launched. A default script located in `<package_base>/scripts/simple_connect` logs connect/disconnect messages to the syslog local0 facility.

Specifying a Roll Log Script

A roll log command is sent to one or more WEPs through the WEP client. When the WEP receives a roll log command, it forwards the command to controld, renames the `logdb` file to `logdb.bak`, creates a new (empty) `logdb` file, then launches the roll log script specified by the configuration variable “`controld.backup_script`”. The following environment variables are set by the parent process when this script is launched:

`MZ_FILE`
Path of the backed up `logdb` file.

`MZ_FIREBOX`
IP address of the Firebox interface that is sending the log packets. For remote Fireboxes, this is the external interface.

`MZ_OVERWRITE`
Set to 1 if a previously existing `logdb.bak` file was overwritten by the roll log operation. Set to 0 if no previously existing `logdb.bak` file was overwritten.

`MZ_RECCOUNT`
Number of log records in the backed-up log file.

`MZ_DATETIME`
Timestamp when log file was rolled over.

Using the Firebox Shell (fbsh) Command-Line Interface

Fbsh is a command-line interface to WatchGuard appliances. Its primary function is to communicate with those appliances over a secure channel to configure and obtain status information. It also provides additional functions such as key generation and bootstrap provisioning. The underlying protocol used to communicate with the Firebox is called the Mazama Packet Filter (MPF), which uses TCP port 4105 by default. Fbsh uses 3DES encryption, but it also supports single-DES for backward compatibility.

Entering the Fbsh Environment

Use the following command to enter the fbsh environment:

```
fbsh firebox [passphrase]
```

Where:

firebox = the Firebox name or IP address

[*passphrase*] = the Firebox status or configuration passphrase (for information on determining which to use, see “Command Permissions,” below). If the passphrase is omitted from the command line, the utility prompts for it. Because it is more secure, this is the preferred method for interactive use.

If successful, this command string results in a connection to the Firebox and the following prompt:

```
fbsh (firebox)>
```

You can now send fbsh commands (listed in “Fbsh Commands” on page 33) to the Firebox. Fbsh displays any output received from the Firebox on your terminal screen.

Command Permissions

Fbsh commands do not override the permissions set on the Firebox. If you are connected to a Firebox with the status passphrase and try to run an fbsh command that requires the configuration passphrase, the Firebox responds with an “Access Denied” message displayed by fbsh.

You will also receive an “Access Denied” message if you try to open a read-write connection to the Firebox while another user already has one open.

If you encounter unanticipated “Access Denied” messages, determine whether connection permissions or user clashes are the cause: connect to the Firebox with the configuration passphrase and issue this command:

```
status writers
```

This displays the IP address of the read/write connection if one exists. If there is none, you probably entered the passphrase incorrectly.

Be careful when you use fbsh with a read/write connection to a Firebox. If you make changes incorrectly, you could accidentally disable the Firebox or make its configuration ineffective. Therefore, if possible, practice the commands against a test Firebox. If you do not have access to a test Firebox, connect to a “live” Firebox with the status passphrase to gain experience with the command set.

Fbsh Syntax

Fbsh syntax resembles that of a typical UNIX shell interface. Syntax elements are the redirect, the semicolon, and the “take source from” character.

Redirect character (>)

The redirect character (>) tells fbsh to take the subsequent word as the name of an input file. For example, suppose you want to use the `get` command to copy the contents of the `wg.cfg` file into the `myfirebox.cfg` file on your local machine. You would use the following command line:

```
fbsh (firebox)> get wg.cfg > myfirebox.cfg
```

This creates a file called `myfirebox.cfg` in the same directory from which you ran fbsh.

You can use the redirect character with many other fbsh commands. For example, to create a file called `myfirebox.status_all` that contains all the output of the `status all` command, you would enter:

```
fbsh (firebox)> status all > myfb.status_all
```

Semicolon (;)

Another shell character supported by fbsh is the semicolon (;). This character parses the given command into multiple commands delineated by the semicolons. For example, to put a new configuration file on a Firebox and then reboot it in a single one-line command, enter:

```
fbsh (firebox)> put wg.cfg ; reboot
```

In this command, fbsh prompts for a file name from which to read, copies this file’s contents into the configuration file stored on the Firebox, and then reboots the Firebox to save the new settings in the configuration file.

“Take source from” (<)

The “take source from” character (<) gives fbsh a file name from which to read. Fbsh will normally prompt you for the source file, but the (<) character makes this unnecessary. For example, if you want to put a new configuration file on a Firebox and reboot it but want to circumvent fbsh prompting for the source file name `myfirebox.cfg`, you could use the following command string:

```
fbsh (firebox)> put wg.cfg < myfb.cfg ; reboot
```

Getting Online Help on Fbsh Commands

Fbsh provides online Help to familiarize you with its option set and to help you build a correctly structured argument. To get a list of all the help available, including an explanation of each item, enter:

```
fbsh (firebox)> help
```

To get help on a specific command, use the following command string:

```
fbsh (firebox)> help command
```

Where:

command = the command for which you want help.

Fbsh also has tab complete functionality helping you determine what commands are available. For example, to learn more about all the status commands, type the following:

```
fbsh (firebox)> help status
```

Manipulating Binary Files

The MPF protocol that handles commands between the WEP and a Firebox is string-based, meaning that to get a binary file over this interface, you must use a special command. When you put the IKE daemon (a process on the Firebox involved with the IPSec implementation) into “trace” mode, it writes binary data into a file in

`/tmp/ike.dat`. To obtain that file, you issue a command such as:

```
fbsh (myfb)> get_binfile /tmp/ike.dat > ike.dat
```

This command returns a series of digits upward on your display. It represents the number of bytes transferred—a progress bar of sorts.

Editing the Configuration File with Fbsh

You can use fbsh along with a text editor to emulate Firebox commands typically issued from GPM. Be aware, however, that this can be a tedious process, and requires a thorough knowledge of every line in the Firebox configuration file format. Also, without the GPM interface, you can easily write a mistaken command construct or use bad data.

A good use of fbsh is to make minor changes to a Firebox’s configuration, as follows:

- 1 Use the `get` command to copy `wg.cfg` to a local file.
- 2 Open `wg.cfg` in a text editor, make the change or changes, and save the file.
- 3 Use the `put` command to replace the configuration file on the Firebox, and then reboot the Firebox.

You can also modify `wg.cfg` using the `edit` command, as shown below. No file argument is required; `wg.cfg` is the default. Although the example shows fbsh launching the `vi` editor, you set either your `EDITOR` or `VISUAL` environment variables to point to the text editor of your choice. The editor, however, must accept a file name as an argument when launched. For fbsh-oriented tasks, a fast and simple editor keeps better pace with your responsibilities and reduces the chance of errors associated with programs that are complex and memory-intensive.

```
fbsh (firebox)> edit
```

```
Launching editor (vi)
```

Fbsh then begins a series of queries for you to answer, such as:

```
Local file to save as (<CR>=don't save):
```

```
Save to firebox ([y]|n)?
```

```
Restart, reboot, or nothing ([r],b,n)?
```

```
fbsh (firebox)>
```

In this exchange, fbsh first asks to save this file locally. The default is no, which you can specify by pressing Enter. Then you are asked to save the file to the Firebox. If you enter `y` or press Enter, it is saved. If

you enter `n`, then the file is not saved and the remaining prompts about saving to a Firebox and rebooting are not displayed.

If you do save the file, however, fbsh prompts to take another action. You can try a restart; if that is not possible, fbsh prompts you again for either a reboot or to intentionally do nothing. In this case choose the `n` option.

Using Fbsh in Scripts

Fbsh is also used in standard shell scripts, or perl, automating otherwise arduous tasks. Although this process requires some knowledge of shell scripting or perl, simple scripts allow you to create your own real-time monitoring tools.

The following is a simple shell script that tracks active connections through the Firebox:

```
#!/bin/sh
while [ 1 ] ; do
fbsh 199.108.47.254 dook-waddie -c 'status connections'
sleep 2
done
```

The following is a script to return information similar to that displayed by the Bandwidth Meter in GPM.

```
#!/usr/bin/perl -w
```

Note

The location of Perl in Solaris 2.6 is variable.

```
use strict;
my $maxtot=1;
my $maxsec=1;
my $lastsec=0;
my $lasttot=0;
my $lastrx=0;
my $lasttx=0;
my $maxkbs=0;
unless (($ARGV[0]) && ($ARGV[0] =~ /\d+/)) { print "Need to specify a time, please.\n"; exit 1;}
while (1) {
my @int = `fbsh 199.108.47.254 dook-waddie -c 'status interfaces'`;
my (
    $sec,
    $msec,
    ) = split(/\s+/, $int[0]);
my (
    $name,
    $tx,
    $txcoll,
    $txerror,
    $rx,
    $rxcoll,
```

```

    $rxerror
    ) = split(/\s+/, $int[1]);
my $kbs = (((($rx + $tx) - $lasttot) * 8) / ($sec - $lastsec)) / 1_024);
if ($kbs > $maxkbs) {
    $maxkbs = $kbs;
}
printf(<<EOL,
    RX: %lu (%.2f Kbit/sec)
    TX: %lu (%.2f Kbit/sec)
    Data: %lu
    Kb/sec: %.2f
    Mbit/sec: %.2f
    Max Kb/sec: %.2f
EOL
#
$rx,
(((($rx - $lastrx) * 8) / ($sec - $lastsec)) / 1_024),
$tx,
(((($tx - $lasttx) * 8) / ($sec - $lastsec)) / 1_024),
($rx + $tx) - $lasttot,
$kbs,
(((($rx + $tx) - $lasttot) * 8) / ($sec - $lastsec) / (1_024 * 1_024)),
$maxkbs,
);
$lastrx = $rx;
$lasttx = $tx;
$lasttot = ($rx + $tx);
$lastsec = $sec;
sleep $ARGV[0];

```

Fbsh Commands

Fbsh commands contain the following elements:

metacharacter

A character that separates a word/token, and may also have a special meaning in addition to being a separator. These include: comma (,) and blank.

token

Any sequence of characters not containing a blank string. Precede metacharacters with a backslash if they are not to be interpreted as such.

string

A sequence of characters. Metacharacters should be preceded by a backslash.

list

Any sequence of tokens or strings separated by blanks. If a string within the list contains a blank or blanks, then enclose the string with either single or double quotation marks.

stdin, stdout, stderr

Usually this is your terminal, unless you use redirection.

pipe

A pipe consists of two or more commands separated by the pipe metacharacter (`|`). Only the first command in a pipeline is an fbsh or a Firebox command. All remaining commands in the pipeline are interpreted as and processed by the UNIX shell.

General fbsh commands**edit**

Retrieves the stored configuration from the Firebox, stores it in a temporary file, launches the editor specified in the editor environment variable (or `/bin/vi`) and allows you to edit the file. You can then make changes to the configuration. When fbsh resumes control, it checks to see if the file was modified. If it was, you are asked whether you want to save the changes to the Firebox, overwriting the original configuration. If you answer yes, your changes are sent to the Firebox, and a sync command is issued.

exit

Exits the fbsh session. A synonym for this command is `bye`.

help [command]

Show online help, as described in “Getting Online Help on Fbsh Commands” on page 30.

Firebox Status Commands

The status commands are similar to one another. Most of them accept a single token as their argument, and return output of the command on stdout.

status all

Generates a detailed report similar to the concatenated output of many of the following status commands.

status arp

Prints the Firebox ARP table.

status authentications

Generates a list showing the mapping of authenticated users to their IP addresses and other information. The report is empty if no users are currently authenticated.

status biosver

Generates information helpful in identification of the BIOS revision. The information contains a 32-byte MD5 fingerprint of the BIOS area, followed by a string identifying the BIOS.

status components

Generates a list identifying each component present in the currently running Firebox flash image. One line is created for each component and contains the name of the component, version, and timestamp information. A boot and root component is always present on any running Firebox.

`status connections`

Generates a line for each connection in the CONNECTED state at the time the command is issued. Only state-oriented connections are reported in this manner; currently, the TCP protocol is the only state-oriented connection type that generates a report.

`status failover`

Displays statistics for the failover feature.

`status firebox`

Generates information relevant to the internal engineering organization responsible for Firebox core-technology development.

`status interfaces`

Generates a formatted table showing statistics for the three Firebox interfaces.

`status loghosts`

Generates a line detailing information about each configured log host.

`status masquerade`

Generates a line showing entries in the masquerading table. The first output line is a key to the format of the remaining lines.

`status mode`

Generates a report of the string user or system to indicate into which mode the Firebox started. The mode system is synonymous with Sys B while user is synonymous with Sys A. The light panel has either the Sys A or the Sys B light lit, and agrees with this output.

`status permstore`

Displays the usage and availability of permanent storage on the Firebox.

`status product`

Lists a number of product related properties such as the Firebox type and identification information.

`status ps`

Generates a list of processes and statistics running on the Firebox.

`status read-only`

Generates either a 0 or 1 indicating whether the Firebox is in read/write or read-only mode, respectively.

`status release`

Prints the software release name running on the Firebox.

`status spoofs`

This command generates information pertaining to spoofing.

`status standby`

Used only when the High Availability feature is enabled in the Firebox. It prints the device and IP address of the standby box. High Availability is not an MSS feature.

`status stuff`

Generates information pertinent to the Firebox.

`status syslogs`

Prints the latest set of syslogs currently available on the Firebox. Because the logs are stored in a circular buffer in the Firebox, you might see all logs. For a more reliable way to view logs, use `controlld`.

`status sysver`

Generates information identifying the version of the system area. This command is recognized only in Sys A or user mode.

`status temp-hostiles`

Generates a line for each host added to the temporary blocked sites list.

`status template`

Generates information pertaining to the template used as a basis for the configuration file currently running. If no template was used, none is displayed. Because templates are a client side feature, this information is retained only in the configuration file, to ease the task of tracking and configuration management.

`status version`

Generates information pertaining to the Firebox core-technology software, as well as MPF Protocol version information.

`status writers`

Displays the IP address or addresses with a read/write session established with the Firebox. The output is either an IP address, none, or self.

Write-Only commands

These commands are valid only when connected to a Firebox with the configuration passphrase.

`backup_flash`

Triggers the Firebox to save the entire flash image from the user/Sys A area to the backup area.

`drop_hostiles address [address...]`

Takes a list of one or more host names as its argument. Each token should be an IP address that is on the temporary blocked sites list. Information regarding the blocked sites is obtained via the `status hostiles` command.

`get_flash filename`

Attempts to retrieve the entire Sys A flash image and store it locally in file name.

`put wg.cfg`

Stores a new configuration file on the Firebox. The argument 'wg.cfg' is required. The data for the configuration is taken from stdin. If stdin has no pending data, then fbsh enters query mode and queries for the name of a file to use as stdin for the command.

`put_flash filename`

This is an advanced command for use by an administrator. This command requires an argument, which is the path to a file containing a complete Firebox flash image for the user area. It is not possible to save to the system area. If you omit the argument, fbsh enters query-mode and awaits input of a file name. Note: This command only transfers the flash image; the Firebox does not begin using the new image until the next time it is started.

`reboot`

Causes the Firebox Master Control Program to discontinue the disbursement of CPU-cycles to all resident tasks. Also provokes the supporting chipset to cause the reset line on the CPU to pull high momentarily, thus forcing the CPU to clear all registers, reinitialize all stack pointers, and reload the program counter with the value stored in ROM at the reset vector.

`restart`

Attempts to restart a Firebox. A restart does not drop established connections and is desirable whenever changes are made to a configuration that do not require a reboot. If the configuration is not “restartable,” this command displays a message, and you must restart for any changes to take effect.

`restore_flash`

Copies a previously stored Sys A flash image from the backup area into the Sys A area.

General Firebox commands

`del_permfile filename`

This is an advanced command for use by an administrator. Deletes the specified file in permanent storage on the Firebox.

`get filename`

Instructs the Firebox to locate *filename* and send its contents. The contents of the file are sent to stdout. To store the contents in a file use redirection.

`get_binfile filename`

This command instructs the Firebox to locate *filename* and send its contents in binary mode.

`get_permfile filename`

Gets a file from permanent storage on the Firebox.

`list_permfile filename`

Lists the contents of the permanent storage area on the Firebox.

`ls path`

Similar to the UNIX `ls` command; lists the files in the directory specified by *path*.

`put_permfile filename`

This is an advanced command for use by an administrator. Uploads a file to permanent storage on the Firebox.

UDS commands

The `uds` commands are generally used to communicate directly with the processes on the Firebox. They are used when a read-only connection is made to the Firebox. The `uds` commands require the following syntax:

```
fbsh (firebox) > uds uds_command
```

The `uds` command is as follows:

`dhcpcd_uds debug`

Turns debugging on in the DHCP daemon.

`dhcpcd_uds leases`

Prints a list of current leases.

`dhcpcd_uds ping`
Returns ping. Use this command as a fast way to find out whether the DHCP daemon is live and running.

`dhcpcd_uds restart`
Forces dhcpcd to restart, reloads the config props, and clears the lease database.

`dvcpcpl_uds restart`
Forces the DVCP client to contact the DVCP server and renew its lease on the configuration file.

`dvcpsv_uds get_info id`
Prints information for the given id such as the last modification time, last configuration download time, and so on.

`dvcpsv_uds list`
Lists all the available uds commands for DVCP server.

`dvcpsv_uds restart`
Restarts the DVCP server, forcing it to reload the configuration files and re-initialize everything.

`ipsec_uds allow_restart`
Toggles whether IKED allows itself to restart.

`ipsec_uds cfg`
Toggles a flag that forces dump of IKE configuration properties to logs. Follow this command with a restart.

`ipsec_uds del_sa [ip] all`
Deletes a given remote gateway's phase 2 Security Association (SA) or both phase 1 and phase 2 SAs if given the `all` option.

`ipsec_uds dev`
Enables or disables generation of extra device logs.

`ipsec_uds dump_sas [ip]`
Displays either a given remote gateway's SA (if an IP is entered) or all the current SAs.

`ipsec_uds dump_tunnels`
Causes iked to print a list of the configured tunnels.

`ipsec_uds ike`
Enables IKE debugging.

`ipsec_uds isakmp_debug [settings...]`
Allows the user to toggle the other ISAKMP debugging options (hash, skey, isakmp, off, all).

`ipsec_uds list`
Prints a list of available uds commands in iked.

`ipsec_uds max_trace_size [size]`
Sets the number of bytes the trace file is allowed to grow to before it is moved to the old dat file and a new one created. If the size is not specified, it returns the current setting.

`ipsec_uds rekey [tunnel_name|tunnel_id]`
Forces iked to rekey the specified tunnel.

`ipsec_uds restart`

Forces iked to exit and restart.

`ipsec_uds sync`

Forces iked to move the dat file to the old name and start a fresh dat file.

`ipsec_uds trace`

Toggles IKE tracing on and off. Turning the tracing on causes all IKE packets to be dumped in file `/tmp/ike.dat` that are available using the `get_binfile` command.

`ipsec_uds uptime`

Displays the number of seconds iked has run since the last reboot or restart.

`ipsec_uds version`

Prints the version of iked.

`netdbg_uds command`

The `netdbg` component is not always present on the Firebox. When it is, it is used to ping, traceroute, and `tcpdump` on the Firebox. For more information on using `netdbg`, see “Using `netdbg`” on page 40.

Command Options

The following options are available with `fbsh` commands:

`-v`

Toggles verbosity on or off.

`-version`

Specifies to display version information.

`-c command`

A command for `fbsh` to execute. If the command includes a blank space, you must include quotation marks around the command (“`status all`”).

`-single`

Specifies that `fbsh` is to use DES encryption instead of 3DES.

`-genkey passphrase`

Specifies the passphrase to use for connection to the Firebox. This option will also display the MD5 hash of passphrase. If this is the only option, `fbsh` will display the hash and exit.

`-connect hostname`

Specifies the hostname or IP address of the Firebox to connect to. However, this option is provided only for backward compatibility and is generally not necessary.

`passphrase`

Specifies the passphrase to use to connect to the Firebox. Unlike the `-genkey` option, this option does not display the MD5 hash of the passphrase. If neither this option, the `-genkey`, nor the `-k` options are not used, `fbsh` will use the value of the `FBPAS` environment variable. If no such variable is set, `fbsh` asks for the passphrase interactively.

`-k key`

The MD5 hash of the passphrase to use.

-b configuration_passphrase

Used for communicating with WatchGuard appliances using TCP/IP mode. This is useful during initial configuration or failure recovery. The protocol used is UDP Broadcast; therefore; the fbsh\ host and the WatchGuard appliance must typically be on the same LAN.

Using netdbg

Use the netdbg utility to issue a ping, trace route, or tcpdump command on the Firebox.

Installing netdbg components

The installation creates a subdirectory for each currently supported security appliance and encryption level in [installation directory]\Components directory. Copy netdbg.wg components into the appropriate WatchGuard directory for the type of Firebox you are working with. Using fbsh or Policy Manager, flash the Firebox with new software.

After flashing the Firebox in question, WatchGuard recommends removing the components as you will most likely not want to install the netdbg utilities on other units. In certain cases, the netdbg utilities might make the flash size too large.

Launching a netdbg utility

Use an fbsh connection to launch a netdbg utility using syntax similar to the following:

```
uds netdbg ping 10.32.0.32
```

This example tells firewalld to send a command to the netdbg process using a UNIX domain socket (uds) called /tmp/netdbg_uds. In the example, the command netdbg will launch is ping 10.32.0.32.

The fbsh prompt immediately returns after one of two messages appear on the console:

```
ERROR
```

```
or
```

```
73: ping
```

The second result indicates that netdbg successfully launched the ping, and the process identifier (PID) number is 73. This number is used to gather additional information regarding the running utility. For example, to view the current output, enter the following command:

```
uds netdbg dump 73
```

This command would dump the contents of the output file associated with the running utility.

Another useful command enables you to display a list of all the currently running and/or known netdbg process:

```
uds netdbg list
```

For which firewalld would return something like the following:

```
248: ping
```

```
249(dead): tcpdump
```

Here you see a running ping process and a completed tcpdump process. Eventually all processes die. Therefore, after you are done viewing the output, you need to drop the process from the list.

```
uds netdbg drop 249
```

Kill a process you no longer need running:

```
uds netdbg kill 248
```

Two additional commands return the known utilities list and the version information.

```
uds netdbg utils
```

```
uds netdbg version
```

Also note that there is a maximum file size for the temporary files that are capturing each utility's output. The default for this is 5Kb. This can be modified using the configuration file property `netdbg.max_file_size`.

Finally, you can also redirect the output to our logging channel instead of a file.

```
uds netdbg -s ping 10.32.0.32
```

Now there will be no file checking and the process can run forever. All of its output will go out on normal logging channels, and thus can be viewed using "status syslogs," LogViewer, or Traffic Monitor.

Redirecting output to TCP socket

To send output of a utility to an unprotected TCP socket, you can use a command similar to the following:

```
uds netdbg -o ping 10.32.0.32
```

The output from this command will either be an error or a description of which port to connect to in order to watch the given command's output. Use a telnet application to connect to the port. netdbg waits until something connects to the port before starting the command. One default that can be overridden is that netdbg only allows a connection to the port from the same address as the request.

Summary of commands

```
uds netdbg [-o, -e] [option]
```

-o

Open.

-s

Send all output to a syslog file.

Options

`allow_from_any`

Enable connections from anywhere to the Firebox.

`drop`

Drop a process from the list of running processes.

`dump processID`

Displays the contents of the output file associated with the running process. Identify the process by ID number.

`list`

Display a list of all currently running and/or known netdbg processes.

`max_file_size`

Change the default maximum file size. The default is 5K.

`ping [-c count] [-i wait] [-p pattern] [-s packetsize] [-t ttl] hostIP`

Send a ping packet to the IP address identified.

`tcpdump [-c count] [-i interface] [-s snaplen] [-w output_file] [-v verbose] [-x hex] [port] number [host] IPaddress`

`traceroute [-g gateway] [-i interface] [-m max_ttl] [-p port] [-s src_addr] [-t type of service] [-w waittime] hostIP [packetlen]`

Initiate a traceroute to the designated host IP address.

`utils`

Display the known utilities list.

`version`

Display the netdbg version number.

One of the best ways to provide hard data for accounting and management purposes is to generate detailed reports showing how the Internet connection is being used and by whom. Rep_cmd is the command-line equivalent of Historical Reports—a reporting tool in GPM that creates summaries and reports of Firebox log activity. You can customize these reports to include exactly the information you need in a form that is most useful to you. For example, you can define a precise time period for a report, consolidate report sections to show activity across a group of Fireboxes, and set properties to display the report data according to your preferences.

Report Output Types

You can export reports to three formats: HTML, NetIQ, and text.

Exporting reports to HTML format

If you export reports to HTML format, you can specify sections for the report as described in “Specifying Report Sections (Required for HTML Format)” on page 45. HTML is the default format for reports; no command-line switch is required to specify this format. However, if you want to select a graph type of other than the default 3-D pie chart, you must use the `-html graphtype` switch, as described in “Command Syntax,” below.

Exporting reports to NetIQ

NetIQ export converts the log file to the NetIQ Enhanced Log Format (WELF) with the default name `report-timestamp.log`. The time stamp is appended to the name so log files are not overwritten. The directory at `/reports/NetIQ` is the default location for the output file.

Exporting reports to text files

Text export converts the log file to a text (comma delimited) file. The same rules and options for the NetIQ option apply to text files.

Command Syntax

The syntax for `rep_cmd` depends on the output format of the report or whether you are building a report using data from a particular log file (`logdb`) or from a directory of log files for a specific Firebox:

For a single log file, HTML format with 3D pie charts

```
rep_cmd logfile section [section...] [-html graphtype] [optional parameters]
```

Where:

logfile = the full pathname of the `logdb` file providing the data for the report. This argument is required.

section = a report section, as described in “Specifying Report Sections (Required for HTML Format)” on page 45. At least one report section must be specified in the command string.

`-html graphtype` = a graph type other than the default 3-D pie chart, as follows:

- `-html 0` = No graph
- `-html 1` = Pie chart
- `-html 2` = 3-D pie chart
- `-html 3` = Bar chart
- `-html 4` = 3-D bar chart

optional parameters = one or more optional parameters, as described in “Specifying a Report Time Period (Optional)” on page 48 and “Miscellaneous Parameters (Optional)” on page 49.

For multiple log files, HTML format

```
rep_cmd firebox_ip logpath logdir section [section...] [optional parameters]
```

Where:

firebox_ip = the IP address of the Firebox for which you want to generate a report.

logpath logdir = the pathname and directory containing the log files for which you want to build a report.

section = a report section, as described in “Specifying Report Sections (Required for HTML Format)” on page 45. At least one report section must be specified in the command string.

optional parameters = one or more optional parameters, as described in “Specifying a Report Time Period (Optional)” on page 48 and “Miscellaneous Parameters (Optional)” on page 49.

For a single log file, NetIQ or text format

```
rep_cmd logfile output_type [optional parameters]
```

Where:

logfile = the full pathname of the `logdb` file providing the data for the report. This argument is required.

output_type = Either `-NetIQ` or `-text`.

optional parameters = one or more optional parameters, as described in “Specifying a Report Time Period (Optional)” on page 48 and “Miscellaneous Parameters (Optional)” on page 49.

For multiple log files, NetIQ or text format

```
rep_cmd firebox_ip logpath logdir output_type [optional parameters]
```

Where:

firebox_ip = the IP address of the Firebox for which you want to generate a report.

logpath logdir = the pathname and directory containing the log files for which you want to build a report.

output_type = either `-NetIQ` or `-text`.

optional parameters = one or more optional parameters, as described in “Specifying a Report Time Period (Optional)” on page 48 and “Miscellaneous Parameters (Optional)” on page 49.

Specifying Report Sections (Required for HTML Format)

You specify report sections to define the type of information—such as packet-filtered traffic, user authentication data, or denied packets—to include in your report. If you are using HTML format, you must specify at least one report section in every `rep_cmd` command string.

If you are creating a report for more than one Firebox, you can consolidate certain sections to summarize particular types of information. Consolidated sections summarize the activity of all devices being monitored as a group as opposed to individual devices.

The information in each section is described below.

Report sections

Report sections can be divided into two basic types:

- **Summary**—Sections that rank information by bandwidth or connections.
- **Detailed**—Sections that display all activity with no summary ranking.

The following is a listing of the different types of report sections and consolidated sections, and the command-line switch required for each:

Firebox Statistics

A summary of statistics on one or more log files for a single Firebox. Use the following switch to generate a report that includes this information:

`-fireboxstats`

Authentication Detail

A detailed list of authenticated users sorted by connection time. Fields include: authenticated user, host, start date of authenticated session, start time of authenticated session, end time of authenticated session, and duration of session. Use the following switch:

`-authentications`

Time Series – Packet Filtered

A table of all accepted connections distributed along user-defined intervals and sorted by time. If you choose the entire log file or specific time parameters, the default time interval is daily. Otherwise, the time interval is based on your selection. Use the following switch:

`-time-series`

Host Summary – Packet Filtered

A table of internal and external hosts passing packet-filtered traffic through the Firebox sorted either by bytes transferred or number of connections. Use the following switch:

`-by-host`

Service Summary

A table of traffic for each service sorted by connection count. Use the following switch:

`-by-service`

Session Summary – Packet Filtered

A table of the top incoming and outgoing sessions, sorted either by byte count or number of connections. The format of the session is: client -> server : service. Rep_cmd attempts to resolve the server port to a table to represent the service name. If resolution fails, rep_cmd displays the port number. Use the following switch:

`-by-session`

Time Summary – Proxied Traffic

A table of all accepted connections distributed along user-defined intervals and sorted by time. If you choose the entire log file or specific time parameters, the default time interval is daily. Otherwise, the time interval is based on your selection. Use the following switch:

`-by-proxy`

Host Summary – Proxied Traffic

A table of internal and external hosts passing proxied traffic through the Firebox, sorted either by bytes transferred or number of connections. Use the following switch:

`-proxy-by-host`

Proxy Summary

Proxies ranked by bandwidth or connections. Use the following switch:

`-proxysummary`

Session Summary – Proxied Traffic

A table of the top incoming and outgoing sessions, sorted either by byte count or number of connections. The format of the session is: client -> server : service. Use the following switch:

`-proxy-by-session`

HTTP Summary

Tables of the most popular external domains and hosts accessed using the HTTP proxy, sorted by byte count or number of connections. Use the following switch:

`-httpsummary`

HTTP Detail

Tables for incoming and outgoing HTTP traffic, sorted by time stamp. The fields are Date, Time, Client, URL Request, and Bytes Transferred. Use the following switch:

`-httpdetail`

SMTP Summary

A table of the most popular incoming and outgoing email addresses, sorted by byte count or number of connections. Use the following switch:

`-smtpsummary`

SMTP Detail

A table of incoming and outgoing SMTP proxy traffic, sorted by time stamp. The fields are: Date, Time, Sender, Recipient(s), and Bytes Transferred. Use the following switch:

`-smtpdetail`

FTP Detail

Tables for incoming and outgoing FTP traffic, sorted by time stamp. The fields are Date, Time, Client, Server, FTP Request, and Bandwidth. Use the following switch:

`-ftpdetail`

Denied Outgoing Packet Detail

A list of denied outgoing packets, sorted by time. The fields are Date, Time, Type, Client, Client Port, Server, Server Port, Protocol, and Duration. Use the following switch:

`-deniedpacketdetail`

Denied Incoming Packet Detail

A list of denied incoming packets, sorted by time. The fields are Date, Time, Type, Client, Client Port, Server, Server Port, Protocol, and Duration. Use the following switch:

`-deniedinpacketdetail`

Denied Packet Summary

Multiple tables, each representing data on a particular host originating denied packets. Each table includes time of first and last attempt, type, server, port, protocol, and number of attempts. If only one attempt is reported, the last field is blank. Use the following switch:

`-deniedpacketsummary`

Denied Service Detail

A list of times a service was attempted to be used but was denied. The list does not differentiate between Incoming and Outgoing. Use the following switch:

`-deniedservicedetail`

WebBlocker Detail

A list of URLs denied due to WebBlocker implementation, sorted by time. The fields are Date, Time, User, Web Site, Type, and Category. Use the following switch:

`-deniedurls`

Denied Authentication Detail

A detailed list of failures to authenticate, sorted by time. The fields are Date, Time, Host, and User.

`-deniedauthentications`

Consolidated sections

These sections, which contain the data for all logs for all Fireboxes, are available if you specify more than one Firebox. They are not relevant if you are running a report for just one Firebox.

Network Statistics

A summary of statistics on one or more log files for all devices being monitored. Use the following switch:

`-g-fireboxstats`

Time Summary – Packet Filtered

A table of all accepted connections distributed along user-defined intervals and sorted by time. If you choose the entire log file or specific time parameters, the default time interval is daily. Otherwise, the time interval is based on your selection. Use the following switch:

`-g-time-series`

Host Summary – Packet Filtered

A table of internal and external hosts passing packet-filtered traffic, sorted either by bytes transferred or number of connections. Use the following switch:

`-g-by-host`

Specifying a Report Time Period (Optional)

Service Summary

A table of traffic for all services sorted by connection count. Use the following switch:

`-g-by-service`

Session Summary – Packet Filtered

A table of the top incoming and outgoing sessions, sorted either by byte count or number of connections. `Rep_cmd` attempts to resolve the server port to a table to represent the service name. If resolution fails, `rep_cmd` displays the port number. Use the following switch:

`-g-by-session`

Time Summary – Proxied Traffic

A table of all accepted proxied connections distributed along user-defined intervals and sorted by time. If you choose the entire log file or specific time parameters, the default time interval is daily. Otherwise, the time interval is based on your selection. Use the following switch:

`-g-proxy-time-series`

Host Summary – Proxied Traffic

A table of internal and external hosts passing proxied traffic, sorted either by bytes transferred or number of connections. Use the following switch:

`-g-proxy-by-host`

Proxy Summary

Proxies ranked by bandwidth or connections. Use the following switch:

`-g-by-proxy`

Session Summary – Proxied Traffic

A table of the top incoming and outgoing sessions sorted either by byte count or number of connections. The format of the session is: client -> server : service. If proxied, connections show the service in all capital letters. If resolution fails, Historical Reports displays the port number. Use the following switch:

`-g-proxy-by-session`

HTTP Summary

Tables of the most frequented external domains and hosts accessed using the HTTP proxy, sorted by byte count or number of connections. Use the following switch:

`-g-httpsummary`

Specifying a Report Time Period (Optional)

Time parameters allow you to limit your report to data from specific time periods. For example, to create a text report using log entries from the entire last month, you would use the following command string:

```
rep_cmd mylogfile -text -lastmonth
```

The time parameters are as follows:

`all`

Entire file or files (default)

`-today`

From midnight to 11:59:59pm today

- yesterday
From midnight to 11:59:59pm yesterday
 - thisweek
From Sunday to Saturday of the present week
 - lastweek
From Sunday to Saturday last week
 - thismonth
From the first to the end of the month
 - lastmonth
Last month
 - thisyear
From midnight January 1 to 11:59:59pm December 31
 - lastyear
From midnight January 1 to 11:59:59pm December 31 for last year
 - specify
Specify time range (used in conjunction with `-starttime` and `-endtime`, below)
 - starttime
Specifies the beginning of a time range. This switch must be followed with a date time string in the format "`dd\mm\yyyy hh:mm:ss`". (You must include the quotation marks.) Time is in 24-hour notation (10:00 p.m. = 22:00:00).
 - endtime
Specifies the end of a time range. This switch must be followed with a date time string in the format "`dd\mm\yyyy hh:mm:ss`". (You must include the quotation marks.) Time is in 24-hour notation (10:00 p.m. = 22:00:00).
- Example: If you want to create a report using log file entries from noon on June 11, 2002 to 6:00 a.m. on June 16, 2002, you would use the following command string:
- ```
rep_cmd logdb.wg1 -fireboxstats -specify -starttime "06\11\2002 12:00:00" -endtime "06\16\2002 06:00:00"
```
- gmtime  
Sets all timestamps in Greenwich Mean Time. The default is management station local time.

---

## Miscellaneous Parameters (Optional)

---

The following parameters can also be included in the `rep_cmd` command string:

- reportname  
A name for the report, such as `MyReport`. The default is `report-timestamp`.
- logpath  
The location for the log files. The default is `/logs`.
- outputpath  
A destination directory for the report created by `rep_cmd`. The default is `/reports` for HTML and text reports, `/reports/NetIQ` for NetIQ reports.

---

## Example Command Lines

- dns  
Enables DNS lookup on IP addresses
- auth  
Resolves IP addresses for user names if authentication is used and the authentication logs are in the log file.
- overwrite  
Overwrites previous reports in the /NetIQ directory.

---

## Example Command Lines

To create a NetIQ report using the default report directory and file name:

```
rep_cmd 172.16.0.251 logdb.wgl -NetIQ
```

To create a NetIQ report written to the file export.log:

```
rep_cmd 172.16.0.251 logdb.wgl -NetIQ export.log
```

To create an HTML report against logdb.wgl showing Firebox statistics, packet filtered by host, packet filtered by service, and by proxy:

```
rep_cmd logdb.wgl -fireboxstats -by-host -by-service
-by-proxy
```

To create the same report but with 2-D bar charts instead of the default 3-D pie charts:

```
rep_cmd -html 3 logdb.wgl -fireboxstats -by-host -by-service
-by-proxy
```

To create a text report using the default directory and file name:

```
rep_cmd 172.16.0.251 logdb.wgl -text
```

To create a text report written to the file export.txt:

```
rep_cmd 172.16.0.251 logdb.wgl -text export.txt
```

# C Language Application Program Interface (API)

---

This chapter contains examples of two usages of the C API. The first example enables the programmer to write applications that can pull sequences of log records from the WEP servers, or from locally stored files. Using the second example, a programmer can write a library to process log records while they are being received from the Firebox by the WEP.

This API provides tools to perform the following functions:

- Customize the processing of log records
- Write applications for archiving records
- Change the format of the records to a format used by a third-party program
- Include additional notification options for records that are being received while the Firebox is running

## Log Record API

---

The WGLog API provides an interface to log records. Logs are available either locally from a logdb file, or by way of a client/server connection to a log server. An encrypted connection to the log server is established on port 4108. Request packets are sent to:

- Request information about the size of the log file
- Request information about relative record indexes for start and end time ranges
- Delete records
- Request the time zone string in use by the Firebox
- Retrieve records based on relative record indexes

The interface starts with one of two open functions. One open function is used for accessing files locally where the program is running. The other open function is used to connect to a log server running on the WEP to download files from that log server. If there are multiple log servers, this function also consolidates the log records between the various log servers while they are running. Because all functions after the open function are unaware of where the log records are stored, they have no dependency on which open function was used. The goal is to enable an application to generically process records without having separate applications for local and remote processing.

All functions are written with a C program language interface, and are thread safe.

### Compile/Link line

```
cc -c -g -xCC -DSUNOS_MODE -DLOGREC_INLIB -I../../include -o example.o
example.c
cc -o example -g example.o -L../../lib -lsocket -lxnet -lwgerr -lcskt -
llogdb -lwglog
```

---

**Note**

---

The makefiles included in the example directory are customized for the Sun compiler. These need to be modified if you are using GNU compilers. Comment out the Sun section and uncomment the GNU section within the makefile.

---

The first example retrieves the size of the consolidated set of log files. This is one of the simplest requests that can be made against a log server. The second example retrieves 9001 records starting with record 1000 and ending with record 10000.

### Get Log File Size

The following codeblock opens the connection to the log\_server process. The first parameter consists of an IP address and MD5 key separated by an equal sign. It specifies the log\_server IP address and the MD5 key used to encrypt the connection between the log\_server and the client. The second parameter follows the same format but represents the IP address and key used by the log\_server to connect to other log\_servers (separated by spaces) for consolidating log records. The third parameter specifies the timeout value for the request in seconds. The fourth parameter specifies whether the connection is read-only (WG\_READONLY) or read/write (WG\_READWRITE). The last parameter is the address of the handle variable that will be passed back for use with the remaining API functions.

```
status = wglog_open_remote
 ("192.168.44.1=f423596ab129af22018420c0e2f0a11b",
 "192.168.44.2=92f92043010438c9093f0c0b92320b9f
192.168.44.3=92f92043010438c9093f0c0b92320b9f",
 TIMEOUT_SECONDS,
 WG_READWRITE,
 &handle);
if (status == WG_SUCCESS)
{
 firebox_ip.s_addr = inet_addr("192.168.30.2");
```

The following codeblock requests the size of the logfile. The first parameter is the handle returned by an open call. The second parameter is the Firebox IP address in binary/network byte order. The third parameter is the address of the size variable that will be loaded with the number of records currently in the specific Firebox log file. The last two fields select a filter for the request. If the filter had been used in this case, the only valid option would be to pass in a filter for syslog records which would return the number of syslogs in the log file. Other options would return all records in the log file regardless of which filter was used.

```
status = wglog_size(handle,
 firebox_ip,
 &size,
 NULL,
 0);

if (status != WG_SUCCESS)
```

```
printf("Error returned:%s\n", wg_strerror(status));
 else
 printf("The current number of records in the main log file is:%ld\n", size);
```

The following code closes the connection with the log\_server:

```
status = wglog_close(handle);
 }

 if (status != WG_SUCCESS)
```

The following code translates error codes into their string equivalents:

```
printf("Error returned:%s\n", wg_strerror(status));

 return 0;
}
```

## Read Log Records

The following code opens the connection to the log\_server process. The first parameter consists of an IP address and MD5 encryption key separated by an equal sign. It specifies the log\_server IP address and the MD5 key used to encrypt the connection between the log\_server and the client. The second parameter follows the same format but represents the IP address and key used by the log\_server to connect to other log\_servers for consolidating log records. The third parameter specifies the timeout value for the request in seconds. The fourth parameter specifies whether the connection is read-only (WG\_READONLY) or read/write (WG\_DUALCREATE). The last parameter is the address of the handle variable that will be passed back for use with the remaining API functions.

```
status = wglog_open_remote
 ("192.168.44.1=f423596ab129af22018420c0e2f0a11b",
 "192.168.44.2=92f92043010438c9093f0c0b92320b9f",
 "192.168.44.3=92f92043010438c9093f0c0b92320b9f",
 TIMEOUT_SECONDS,
 WG_READWRITE,
 &handle);

if (status == WG_SUCCESS)
{
 wg_erno_t status = WG_FAIL;
 struct log_record log_array[MAXIMUM_RECORDS];
 unsigned long num_records = 0;
 int current_sum = 0;
 int get_recs = 0;

 do
 {
```

```
get_recs = (end_record - start_record + 1) - current_sum;
get_recs = (get_recs < MAXIMUM_RECORDS) ? get_recs : MAXIMUM_RECORDS;
```

```
if (get_recs)
{
```

The following code requests the records from the logfile. The first parameter is the handle returned by an open call. The second parameter is the Firebox IP address in binary/network byte order. The third parameter is the offset into the file that will be the first record read for this function call. The fourth and fifth fields are used to select a filter for the request. In this case, if the filter had been used, the only valid option would be to pass in a filter for syslog records which would return the number of syslogs in the log file. Other options would return the total records in the log file regardless of which filter was used. The sixth field is the array where the records are to be placed when retrieved from the file. The seventh field is the number of records requested. The last field is the address of the variable that contains the actual number of records that the function was able to retrieve.

```
 status = wgllog_read(handle,
 firebox_ip,
 start_record + current_sum,
 log_types,
 num_types,
 log_array,
 get_recs,
 &num_records);

 if (status == WG_SUCCESS)
 {
 int index;

 current_sum += num_records;
 for (index = 0; index < num_records; index++)
 {
 if (!logrec_ntoh(&log_array[index]))
```

The following code calls a function to display the record's content:

```
 logrec_print(stdout, &log_array[index]);
 else
 printf("Invalid record.\n");
 }
 }
} while(num_records == get_recs && get_recs && status == WG_SUCCESS);

if (status != WG_SUCCESS)
printf("Error returned:%0s\n", wg_strerror(status));
```

The following code closes the connection with the log\_server:

```

 status = wgllog_close(handle);
}

if (status != WG_SUCCESS)

```

The following code translates error codes into their string equivalents:

```

 printf("Error returned:%s\n", wg_strerror(status));

 return 0;
}

```

The following function displays the record content. This function contains a call to `logrec_dump()` which is not included in the example. `logrec_dump()` simply displays the field in a hexdump format.

```

static void logrec_print(FILE* logfile, struct log_record* logrec)
{
 logrec_print_header(logfile, logrec);

 switch(logrec->type)
 {
 case LOGREC_SYSLOG_STR:
 if (logrec->u.syslog.pid == 0xffffffff)
 fprintf(logfile, "level:%lu pid:KRN ",
 logrec->u.syslog.level);
 else
 fprintf(logfile, "level:%lu pid:%lu ",
 logrec->u.syslog.level,
 logrec->u.syslog.pid);
 fprintf(logfile, "buffer:%s",
 logrec->u.syslog.buffer);
 break;

 case LOGREC_URL:
 case LOGREC_BLOCK_URL:
 case LOGREC_AUTH:
 case LOGREC_AUTH_DONE:
 case LOGREC_NOAUTH:
 case LOGREC_BANDWIDTH:
 case LOGREC_MSG:
 case LOGREC_OPTION_REC:
 case LOGREC_OPTION_END:
 case LOGREC_OPTIONS:
 case LOGREC_ACK:

```

```
case LOGREC_DEBUG:
case LOGREC_NOTIF:
case LOGREC_HTTP_START:
case LOGREC_HTTP_END:
case LOGREC_SMTP_START:
case LOGREC_SMTP_RCPT:
case LOGREC_SMTP_END:
case LOGREC_FTP_START:
 case LOGREC_FTP_END:
case LOGREC_RA_START:
case LOGREC_RA_INFO:
case LOGREC_RA_END: fprintf(logfile, "buffer:%s", logrec->u.buffer);
 break;

 case FW_TRACE:
case FW_NOTIFY:
case FW_LOG:
case FW_PRINT:
case FW_DUMP:
case FW_ICMP_MSG:
case FW_DEFAULT:
case FW_OPTIONS:
case FW_SPOOF:
case FW_HOSTILE_SITES:
case FW_HOSTILE_DPORTS:
case FW_OUT_TCP_STATE:
case FW_IN_TCP_STATE:
case FW_IN_TCP:
case FW_IN_UDP:
case FW_IN_ICMP:
case FW_IN_IP:
case FW_IN_DYNAMIC:
case FW_IN_ANY:
 fprintf(logfile,
 "opcode:%ud id:%ud index:%ud len:%ud real_index:%ud service:%s packet_lead:",
 logrec->u.packet.log.opcode,
 logrec->u.packet.log.id,
 logrec->u.packet.log.index,
 logrec->u.packet.log.len,
 logrec->u.packet.log.real_index,
 logrec->u.packet.service);
 logrec_dump(logfile, logrec->u.packet.log.packet_lead, 30);
 break;

default:
 fprintf(logfile, "Unknown record type:%d\n", logrec->type);
}
```

```

 fprintf(logfile, "\n");
}

```

## Function Descriptions

This section describes the API functions.

### wglog\_open\_local

This function is used to open files that are accessible through the local file system. The `wglog_open_local` function does not actually open the files; it stores the information passed to it in the handle. When one of the other functions is called, the path and mode passed into this function are used to actually open the local file.

If the Firebox IP value is zero in the other functions, then the path is interpreted to mean a complete path to a logdb file. If, however, the Firebox IP value is set, then the path is interpreted as the base of a directory tree configured as a set of numbers equal to the pure IP address (dot notation, not URL or friendly name) of the Firebox. The logdb file is expected to be located at the end of the IP named directory tree. For example, if the Firebox IP is 205.80.5.1 and the path is `/opt/WGwep/logs`, then the file should be located here:

```
/opt/WGwep/logs/205/80/5/1/logdb
```

`wglog_open_local` defers the actual opening of the log file because once a call to the open function is made, different functions can access different Firebox IP addresses without closing the handle. Internally, the local version of the open call closes an existing file and opens a new file if the Firebox IP changes between two requests.

After this function returns a `WG_SUCCESS` value, you must call the `wglog_close` function to free the handle. Otherwise, there will be a memory leak and possibly a file descriptor leak.

### Syntax

```

wg_errno_t wglog_open_local(const char* path,
 WGLOG_MODE mode,
 WGLOG_HANDLE* handle_p);

```

### Parameters

`path`

The path to a logdb file, or a path to the base of the tree of logdb files.

`mode`

Set to `WG_READONLY` for read-only access or `WG_READWRITE` for read/write access, which includes writing records and purging records.

`handle_p`

A pointer to a `WGLOG_HANDLE` variable which will be set if the `wglog_open_local` function returns `WG_SUCCESS`.

### Error Codes

`WG_SUCCESS`

`WG_FAIL`

WG\_FAIL\_MEM

### **wglog\_open\_remote**

This function opens a connection to a log server process. Unlike the `wglog_open_local` function, this function directly opens the connection immediately. All other functions except for the `wglog_open_local` function use the handle passed back by this function to send requests to the log server

#### **Syntax**

```
wg_errno_t wglog_open_remote(const char* log_server_access_string,
 const char* log_server_list,
 int timeout_seconds,
 WGLOG_MODE mode,
 WGLOG_HANDLE* handle_p);
```

#### **Parameters**

`log_server_access_string`

In the format `ip=MD5`, where `ip` is the IP address (in dot notation) for the log server that this API will connect directly to. `MD5` is the MD5 hash of the password used to access the log server.

`log_server_list`

In the format `ip=MD5`, as described above. This list consists of one or more `ip=MD5` pairs separated by spaces. Each `ip=MD5` pair represents an additional log server that will be used in the consolidation process to consolidate log records that the Firebox might have logged to. This list typically matches the Firebox configuration log server list of Fireboxes less the log server in the `log_server_access_string` parameter.

`timeout_seconds`

The number of seconds in which an inactive connection will time out.

`mode`

Set to `WG_READONLY` for read-only access or `WG_READWRITE` for read/write access.

`handle_p`

Pointer to a `WGLOG_HANDLE` variable which will be set if the `wglog_open_local` function returns `WG_SUCCESS`.

#### **Error Codes**

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_BADKEY

WG\_FAIL\_CONNECT

WG\_FAIL\_BIND

WG\_FAIL\_BADPARAM

### **wglog\_close**

This function closes the connection to the remote server or closes the last local file used, and deallocates the memory associated with the handle.

**Syntax**

```
wg_errno_t wglog_close(WGLOG_HANDLE handle);
```

**Parameters**

*handle*

The parameter that was passed back from one of the two open calls. It should be called only when the connection no longer needs to be used and the open call returned a WG\_SUCCESS status.

**Note**

Failure to call this function results in a memory leak, as well as a possible file descriptor leak.

**Error Codes**

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_BADPARAM

WG\_FAIL\_CLOSE

**wglog\_size**

This function returns either the total number of records in the log file, or the total number of syslogs in the log file if the log\_type is set to LOGREC\_SYSLOG\_STR.

A log file when the connection has been established to a remote log server with a set of alternate log servers specified in the wglog\_open\_remote function is actually the consolidated set of log records among all of the referenced log servers. Therefore the size for three log servers will be the sum of a size call to each log server individually.

**Syntax**

```
wg_errno_t wglog_size(WGLOG_HANDLE handle,
 struct in_addr firebox_ip,
 unsigned long* size,
 const LOGDB_TYPE* log_type,
 int num_log_types);
```

**Parameters**

*handle*

Parameter is initialized in one of the two open function calls.

*firebox\_ip*

IP address of the Firebox that the request applies to. This field is in network byte order.

*size*

Pointer to a variable allocated by the caller. The function puts the number of records in the log file in this field. If multiple log servers are being consolidated this value represents the sum of the records from all of the log servers specified in the open call.

log\_type

Field that is a pointer to an array of LOGDB\_TYPE values. If this array contains one entry with LOGREC\_SYSLOG\_STR as the LOGDB\_TYPE, then the size information reflects the number of syslog records in the log file, not the total number of log records in the log file.

num\_log\_types

Field that is the number of elements in the log\_type array above.

### **Error Codes**

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_READ

WG\_FAIL\_WRITE

WG\_FAIL\_MEM

### **wglog\_get\_timezone**

This function retrieves the timezone string from the logdb file for the specified Firebox IP address. The Firebox passes its timezone to controld (the WEP log process) each time it establishes a connection. Controld stores this value in the log file when a log file is first created. This method gives the reader of the log file the ability to access the timezone string even if the reader does not have access to the Firebox configuration file.

### **Syntax**

```
wg_errno_t wglog_get_timezone(WGLOG_HANDLE handle,
struct in_addr firebox_ip,
char* timezone,
int timezone_len);
```

### **Parameters**

handle

Parameter initialized in one of the two open function calls.

firebox\_ip

IP address of the Firebox that the request applies to. This field is in network byte order.

timezone

Timezone value found in the header of the *firebox\_ip* logdb file. This field must be allocated by the caller and at least *timezone\_len* bytes long

timezone\_len

Parameter must be greater than or equal to WGLOG\_TIMEZONE\_LEN. It represents the length of the timezone field.

### **Error Codes**

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_READ

WG\_FAIL\_WRITE

WG\_FAIL\_MEM  
 WG\_FAIL\_TIMESHORT  
 WG\_FAIL\_LOCK  
 WG\_BAD\_FILE

### **wglog\_set\_timezone**

This function is designed to set the timezone string into the header of a logdb file. It works only on handles opened using the `wglog_open_local` function. Calling this function on a `wglog_open_remote` connection will return an error.

#### **Syntax**

```
wg_errno_t wglog_set_timezone(WGLOG_HANDLE handle,
 struct in_addr firebox_ip,
 const char* timezone,
 int timezone_len);
```

#### **Parameters**

`handle`

Parameter initialized in one of the two open function calls.

`firebox_ip`

IP address of the Firebox that the request applies to. This field is in network byte order.

`timezone`

Timezone string that will be set in the header of the logdb file. It must not be longer than `timezone_len`.

`timezone_len`

Parameter must be greater than or equal to `WGLOG_TIMEZONE_LEN`. It represents the length of the timezone field.

#### **Error Codes**

WG\_SUCCESS  
 WG\_FAIL  
 WG\_FAIL\_TIMESHORT  
 WG\_FAIL\_LOCK  
 WG\_BAD\_FILE  
 WG\_BAD\_BADPARAM

### **wglog\_time\_index**

This function retrieves the log record indexes at or immediately following `time1` and at or immediately before `time2`. If only one time value is needed, set `time2` to zero.

If no records are found between the two time ranges and the times are before the earliest time in the file, an error is returned. If no records are found between the two time ranges, however, and the times are after the latest time in the file, the last record will be returned for both time indexes. The only way to verify that an instance of a single record between the two time ranges wasn't found is by calling `wglog_read` and inspecting the actual time for that record. This is a temporary workaround for the current release.

**Syntax**

```
wg_errno_t wglog_time_index(WGLOG_HANDLE handle,
 struct in_addr firebox_ip,
 const LOGDB_TYPE* log_types,
 int num_log_types,
 time_t time1,
 time_t time2,
 unsigned long* time1_record_index,
 unsigned long* time2_record_index);
```

**Parameters**

handle

Parameter initialized in one of the two open function calls.

firebox\_ip

IP address of the Firebox that the request applies to. This field is in network byte order.

log\_types

Pointer to an array of LOGDB\_TYPE values. If this array contains one entry with LOGREC\_SYSLOG\_STR as the LOGDB\_TYPE, the time indexes reflect the syslog records in the log file.

num\_log\_types

Number of elements in the log\_type array above.

time1

The time in seconds since the first second of 1970. It is represented in GMT time.

time2

The time in seconds since the first second of 1970. It is represented in GMT time.

time1\_record\_index

The record in the log file that matches *time1* or, if a log record does not have the exact time as *time1*, then the first record available after time1.

time2\_record\_index

The record in the log file that matches *time2* or, if a log record does not have the exact time as time2, then the first record available before *time2*.

**Error Codes**

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_READ

WG\_FAIL\_WRITE

WG\_FAIL\_MEM

WG\_FAIL\_BADPARAM

WG\_FAIL\_LOCK

WG\_BAD\_FILE

**wglog\_read**

This function retrieves log records from record\_number back up to record\_number - *log\_array\_len*.

**Syntax**

```
wg_errno_t wglog_read(WGLOG_HANDLE handle,
 struct in_addr firebox_ip,
 unsigned long first_record_index,
 const LOGDB_TYPE* log_types,
 int num_log_types,
 log_record* log_array,
 int log_array_len,
 unsigned long* num_log_records);
```

**Parameters**

*handle*

Parameter initialized in one of the two open function calls.

*firebox\_ip*

IP address of the Firebox that the request applies to. This field is in network byte order.

*first\_record\_index*

Index of the first record to be retrieved.

*log\_types*

Pointer to an array of LOGDB\_TYPE values. The read call will return only records matching this array, or all records if this array is NULL.

*num\_log\_types*

Number of elements in the log\_type array above. Set this field to zero if all records are wanted.

*log\_array*

Address of an array allocated by the caller. Log records read by this function will be placed here. For performance reasons, it is better to make this array relatively large. 100 or 1000 elements are better than using an array that is 1, 5, or 10 records long.

*log\_array\_len*

Number of elements in the array above.

*num\_log\_records*

Used to return the actual number of records read by this function. If a request to read 1000 records is made but only 12 records are available, then this value will be set to 12.

**Error Codes**

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_LOCK

WG\_FAIL\_BADPARAM

WG\_FAIL\_READ

WG\_FAIL\_WRITE  
WG\_FAIL\_MEM  
WG\_BAD\_FILE

**wglog\_write**

This function writes a log record to a local logdb file. It works only on handles opened by way of the `wglog_open_local` function. Calling this function on a `wglog_open_remote` connection will return an error.

**Syntax**

```
wg_errno_t wglog_write(WGLOG_HANDLE handle,
 log_record* log);
```

**Parameters**

handle

Parameter initialized in one of the two open function calls.

log

Pointer to a single log record that will be written to the log file. This function supports a very limited form of write. You are allowed to use this function only on the `wglog_open_local` opened handle, with the path set to an absolute file name. Access to the base of the log tree is not supported for this release for the write function.

**Error Codes**

WG\_SUCCESS  
WG\_FAIL  
WG\_FAIL\_LOCK  
WG\_BAD\_HANDLE  
WG\_FAIL\_BADPARAM  
WG\_FAIL\_RMTSRV  
WG\_BAD\_FILE  
WG\_FAIL\_MEM

**wglog\_delete**

This function marks a number of consecutive records starting at index 0 for deletion. This call does not delete records; it simply sets the point up to which the records will be deleted when the purge function is invoked. This function can be called multiple times setting the deletion point higher each time.

**Syntax**

```
wg_errno_t wglog_delete(WGLOG_HANDLE handle,
 struct in_addr firebox_ip,
 unsigned long num_records);
```

**Parameters**

handle

Parameter initialized in one of the two open function calls.

firebox\_ip

IP of the Firebox that the request applies to. This field is in network byte order.

num\_records

Sets the total number of deleted records in the log file. This function can be called repeatedly, setting this value higher and higher. If the number is set lower than the *num\_records* of a previous call, an error is returned.

### Error Codes

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_READ

WG\_FAIL\_WRITE

WG\_FAIL\_MEM

WG\_BAD\_FILE

WG\_FAIL\_BADPARAM

WG\_FAIL\_LOCK

### wglog\_purge

This function physically removes all records marked for deletion. It works only if all consecutive records from 0 to 80% of the file are marked; otherwise an error status is returned.

### Syntax

```
wg_errno_t wglog_purge(WGLOG_HANDLE handle,
 struct in_addr firebox_ip);
```

### Parameters

handle

Parameter initialized in one of the two open function calls.

firebox\_ip

IP address of the Firebox that the request applies to. This field is in network byte order.

### Error Codes

WG\_SUCCESS

WG\_FAIL

WG\_FAIL\_READ

WG\_FAIL\_WRITE

WG\_DELMARK\_TOOSMALL

WG\_NO\_DELMARK

WG\_FAIL\_LOCK

WG\_FAIL\_BADPARAM

WG\_FAIL\_MEM

WG\_BAD\_FILE



---

This appendix describes the WEP custom notification filters.

## Notification Overview

---

Two processes, `controld` and `notifyd`, each dynamically load a shared library function that can be customized by the user. WatchGuard supplies a simple default shared library function that sends out notifications from the `notifyd` server for certain log packets. Some of the possible ways to use this feature are to set up a separate log archival method, or to send out customized notifications for your site. The intent behind using these two functions is to move the heavy notification processing out of the `controld` process and over to the `notifyd` process.

`Controld` loads the `notify_filter` function to identify which records should be passed over to the `notifyd` process. The goal when writing a custom function is to return a 1 for each record that might need to be processed by `notifyd`. Otherwise the function should return a zero. It is important that this function do minimal processing on the packet. Any delays in this function could cause packets to be lost on the connections between `controld` and the Fireboxes that it is connected to. Specifically, this function must not call any subroutines that could cause it to block. This function is used to scrape off unneeded records that the `proc_log_record` function, called from `notifyd`, will discard anyway. By stripping out the extra records, the number of records sent through the UNIX domain socket to the `notifyd` process is reduced, reducing the burden on the system when it experiences heavy loads.

`Notifyd` loads the `proc_log_record` function to do custom processing of log records. This function is used to further process the records sent over from `controld`. The users can do nearly anything they want to the record in this routine. However, it is important to keep this function fairly lean.

## Compile/Link line

---

```
cc -c -xCC -DSUNOS_MODE -I../include -o notify_filter.o notify_filter.c
cc -G -o notify_filter.so notify_filter.o
```

## Example

### Notify Filter

This example function is used to discard all records except for those that are of the LOGREC\_NOTIF type.

```
int notify_filter(unsigned long addr,
 const log_record* log_rec)
{
 switch(log_rec->type)
 {
 case LOGREC_NOTIF:
1) return 1;
 break;

 default:
2) return 0;
 }
}
```

- 1 Return a one to signal to controld that the record should be forwarded to notifyd.
- 2 Return a zero to signal to controld to discard the record. The record will still be sent to the logdb file; this function only affects how notifyd receives the record.

### Proc Log Record

This function completes the processing on the log record. In this example the only records received are of the type LOGREC\_NOTIF. All notification type records for this example are processed using the send\_notification function.

```
int proc_log_record(unsigned long addr, const log_record* log_rec)
{
 static int firsttime = 1;

1) if (firsttime)
 {
 firsttime = 0;
 if (proc_log_record_init())
 return 1;
 }

2) return send_notification(addr, log_rec);
}
```

- 1 Initialize custom notification logic.
- 2 Send the notification. The implementation for this function has been left out of the example. View the complete example in the file examples/controld/notify\_filter.c located in the WGdev package.

---

## Function Descriptions

---

### Notify Filter

This function is called by controld.

#### Syntax

```
int notify_filter(unsigned long addr,
 const log_record* log_rec);
```

#### Parameters

*addr*

Address of Firebox that generated the packet in network byte order.

*log\_rec*

Log record in host byte order.

#### Return Code

Return 1 to forward the log record to notifyd.

Return 0 to discard the packet.

The packet is recorded in the logdb file regardless of the return value of this function.

### Proc Log Record

This function is called by notifyd. All arguments are in host byte order.

#### Syntax

```
int proc_log_record(unsigned long addr,
 const log_record* log_rec);
```

#### Parameters

*addr*

Address of Firebox that generated the packet in host byte order.

*log\_rec*

Log record in host byte order.

#### Return Code

-1 = Break a notifyd connection with controld

1 = Log the error message "unable to process %s message from %s"

0 = Successful operation

---

## Notification Environmental Variables

---

Notification programs pass several environmental variables that communicate various attributes of the IP packet that trigger notification.

In addition, a number of environmental variables describing the state of the firewall daemon are set.

These variables include:

**Environment****BASEDIR**

If BASEDIR is set, its value is used to determine where to start looking for the configuration file.

**Script Environment**

The environmental variables set for any script that is launched from controld are as follows:

**MZ\_FIREBOX**

The address in dotted IP notation of the Firebox from which the notification originated.

**MZ\_NOTIFIER**

The name of the notification, which can be:

- {“options.spoofing.”,”spoofing”}
- {“options.ipoptions.”,”ipoptions”}
- {“options.default.incoming.”,”default”}
- {“options.default.outgoing.”,”o\_default”}
- {“options.hostile\_site.”,”hostile\_site”}
- {“options.hostile\_port.”,”hostile\_port”}
- {“options.probe.address.”,”prob”}
- {“options.probe.port.”,”o\_prob”}

**MZ\_MAILTO**

The email address that controld has been configured to use.

**MZ\_MAILHOST**

The mail server controld has been configured to use.

**MZ\_COUNT**

The number of times a duplicate notice was received within the interval controld has been configured to use.

**MZ\_DISPOSITION**

The notification descriptive ‘type’ (for example, allow or deny).

**MZ\_DIRECTION**

The direction of the packet; value is either ‘in’ or ‘out’.

**MZ\_INTERFACE**

The name of the interface that received the reported packet.

**MZ\_LENGTH**

The number of bytes of the reported packet.

**MZ\_PROTOCOL**

The protocol of the reported packet.

**MZ\_TTL**

The value taken from the time-to-live field from a reported TCP packet.

MZ\_SADDR

The IP address taken from the source address field from the reported packet.

MZ\_DADDR

The IP address taken from the destination address field from the reported packet.

MZ\_SPORT

The source port number from a reported TCP or UDP packet.

MZ\_DPORT

The destination port number from a reported TCP or UDP packet.

MZ\_TYPE

The type of a reported ICMP packet.

MZ\_CODE

The code of a reported ICMP packet.



---

# Index

## A

archive client 5, 24

## B

binary files, transferring 31

## C

CD-ROM, installing WEP packages through 9

CloseConn command 15, 16

commands

CloseConn 15, 16

DeleteLogFile 19

Get 18

GetConfig 19

GetLocalLog 17

GetLog 18

GetLogKey 17

OpenConn 15, 16

OpenConnWithHash 16

Pause 20

PutConfig 19

Reboot 20

RollLogFile 18

SetKeys 17

Sleep 20

StatusComponents 17

StatusLoghosts 17

StatusVersion 16

StatusWep 16

Timeout 20

compile/link line

described 67

with notify filter 68

configuration file

editing with fbsh 31

initializing variables in 11

modifying 21-27

retrieving using Wepclient 19

uploading using wepclient 19

controld

and WGdev 67

custom library for 25

debugging 25

described 3

notifications 25

roll log script 27

controld.max\_connections 24

controld.num\_workers 24

custom notification filters 67

## D

daemons. See WEP daemons

dbfetch 26

DeleteLogFile command 19

DNS, configuring for WEP 7

## E

event retrieval, from logs 3

## F

failover, and time synchronization 13

fbsh

'take source from' character 30

and binary files 31

and Firebox permissions 29

and perl 32

and scripting 32

command syntax 30

commands 33-40

described 29

editing configuration file with 31

entering environment 29

general commands 37

Help for 30

redirect character 30

semicolon 30

status commands 34

tracking active Firebox connections 32

uds commands 37

filters, custom notification 67

Firebox shell. See fbsh

Fireboxes

attached to WEP 16

command keys 13

event retrieval 3

loading new configuration file using wepclient 19

monitoring connection with wepd 3

permissions and fbsh 29

rebooting using wepclient 20

retrieving configuration file 19

retrieving log hosts for 17

retrieving logs file of 18

retrieving logs of 17

retrieving software configuration of 17

retrieving status information on 16

tracking connections with fbsh 32

verifying passphrases for 17

FTP, installing WEP packages through 9

## G

GetConfig command 19  
GetLocalLog command 17  
GetLog command 18  
GetLogFile command 18  
GetLogKey command 17

## I

IP addresses, WEP 12

## L

log files  
  converting to text files 43  
  converting to WebTrends Enhanced Log Format 43  
  deleting 19  
  retrieving 18  
  rolling over 18  
log hosts for Firebox 17  
log\_server 4  
logging  
  and archive client 24  
  configuring 24  
  get log file size API 52  
  read log records API 53  
  traffic patterns 3  
logs  
  consolidating from multiple WEPs 18  
  retrieving with Wepclient 17

## M

Mazama Packet Filter (MPF) 29

## N

NFS, installing WEP packages through 9  
notification  
  customizing 25  
  environmental variables for 69  
notification filters, custom 67  
Notify Filter function 68, 69  
notifyd  
  and WGdev 67  
  custom library for 25  
  custom-built library 25  
  debugging 26  
  described 4  
  notifications 4  
nsswitch.conf 8

## O

OpenConn command 15, 16  
OpenConnWithHash command 16

## P

Pause command 20  
Proc Log Record function 68, 69  
PutConfig command 19

## R

Reboot command 20  
rep\_cmd  
  command syntax for 44–45  
  described 43  
  example command lines 50  
reports  
  authentication details 45  
  consolidated sections 47  
  consolidating sections 45  
  creating with rep\_cmd 43–50  
  customizing 43  
  denied incoming/outgoing packet detail 47  
  denied packet summary 47  
  denied service detail 47  
  exporting to HTML format 43  
  exporting to text files 43  
  exporting to WebTrends 43  
  Firebox statistics 45  
  FTP detail 46  
  host summary 45, 46  
  HTTP detail 46  
  HTTP summary 46, 48  
  network statistics 47  
  output types 43  
  proxy summary 46  
  sections in 45  
  service summary 45  
  session summary 46  
  SMTP summary 46  
  specifying report sections 45  
  time periods for 48  
  time summary 46, 48  
  WebBlocker detail 47  
resolv.conf 8  
RollLogFile command 18

## S

scripts  
  using fbsh in 32  
  using wepclient with 15  
sendmail 13  
SetKeys command 17  
Sleep command 20  
StatusComponents command 17  
StatusFB command 16  
StatusLoghosts command 17  
StatusVersion command 16  
StatusWep command 16

## T

time synchronization 13  
Timeout command 20

---

## W

WatchGuard Event Processor. See WEP

wb\_config, default values 26

WebBlocker

and WEP 5

downloading database with dbfetch 26

WebTrends Enhanced Log Format 43

WEP

and Firebox keys 13

and WebBlocker 5

command key 13

compile/link line 52

components of 2

configuring 11

configuring DNS for 7

configuring sendmail 13

configuring services for 12

current build number 16

described 1

event recording 3

event retrieval 3

executing script when Firebox connects 27

Fireboxes attached to 16

functions of 3

in MSS 2

installing 8–11

installing packages 9

installing packages using CD-ROM 9

installing packages using NFS 9

IPaddress of 12

log record API 51

MSS version number 16

retrieving Firebox status 16

roll log command 27

running WEP server 12

server requirements for 7

starting 11

stopping 12

time synchronization 13

WEP daemons

controld 3

described 2

functions of 3

log\_server 4

notifyd 4

webblocker 5

Wepd 3

wepclient

commands 16

deleting log files 19

described 15

getting logging channel MD5 key 17

loading configuration files 19

management commands 20

opening session 15

opening session with MD5 hash of password 16

pausing 20

putting to sleep 20

retrieving Firebox configuration 19

retrieving log hosts 17

retrieving logs and files 17

retrieving status 16

rolling over log file 18

setting encryption for logging channel 17

setting timeout for commands 20

using 15–20

using interactively 15

using with a script 15

wep-config-cmd 11

command syntax 21

wep-config-command

command options 22

wepd

and communication protocol 13

connection status to GPM 3

described 3

monitoring Firebox connections 3

servicing client requests 4

WepMonitor

described 26

variables 26

WEPTools 2

WGdev 2, 67

WGkitdoc 10

wglog API 5

wglog\_close 58

wglog\_delete 64

wglog\_get\_timezone 60

wglog\_open\_local 57

wglog\_open\_remote 58

wglog\_purge 65

wglog\_read 63

wglog\_set\_timezone 61

wglog\_size 59

wglog\_time\_index 61

wglog\_write 64

WGtools 10

WGwep 10

WGwep.conf

commenting 22

modifying 21

WGwep.conf. See also configuration file

WGwepdoc 10

## X

X Windows 7

