Sanewall Manual

Making sense of firewalling using Sanewall

Sanewall Team

Release 1.1.2
Built 12 May 2013

Contents

List of Tables

Chapter 1

Introduction

1.1 Latest version

The latest version of this document will always be available here. There are PDF and HTML versions.

1.2 Who should read this manual

This manual is aimed at those who wish to create and maintain firewalls with Sanewall.

There is a lack of basic and tutorial information currently. Sanewall is a fork of FireHOL and its documentation can be used to learn the configuration language. See the FireHOL website.

1.3 Where to get help

The Sanewall website.

The mailing lists and archives.

The package comes with a complete set of manpages, a README and a brief INSTALL guide.

Chapter 2

RFC 4890 Recommendations

2.1 Introduction

RFC 4890 is entitled "Recommendations for Filtering ICMPv6 Messages in Firewalls".

The recommendations pertain to firewalling at router level, not necessarily hosts or bridges (which may need to treat some packets differently, e.g. NS/NA and RS/RA).

The sections below were extracted from the example implementation; each one describes how the recommendation can be achieved with Sanewall.

It is assumed that a policy of reject or deny is in place. If that is not the case then some packet types need dropping explicitly to meet the recommendations.

2.2 Allow outbound echo requests from prefixes which belong to the site

The router command (see router definition: sanewall-router(5)) should be used with an appropriate src rule parameter (see optional rule parameters: sanewall-rule-params(5)).

2.3 Allow inbound echo requests towards only predetermined hosts

The ping - Ping (ICMP echo) - complex service should be used in combination with an appropriate dst rule parameter (see optional rule parameters: sanewall-rule-params(5)).

2.4 Allow incoming and outgoing echo reply messages only for existing sessions

This is handled automatically by the ping - Ping (ICMP echo) - complex service .

2.5 Deny icmps to/from link local addresses

The router command (see router definition: sanewall-router(5)) should be used with an appropriate src and dst rule parameter (see optional rule parameters: sanewall-rule-params(5)). For example:

```
src not "${UNROUTABLE_IPS}" dst not "${UNROUTABLE_IPS}"
```

2.6 Drop echo replies which have a multicast address as a destination

The ping - Ping (ICMP echo) - complex service can be used with an appropriate src rule parameter (see optional rule parameters: sanewall-rule-params(5)). For example:

```
ipv6 route ping src not "${MULTICAST6_IPS}"
```

will prevent incoming echo-requests from multicast IPs and replies to them.

2.7 Allow incoming destination unreachable messages only for existing sessions

Ensure any routers have:

```
server ipv6error accept
```

Adding dst "$inner_prefix" ensures only public hosts receive the messages. See ipv6error - ICMPv6 Error Handling - complex service .

2.8 Allow outgoing destination unreachable messages

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.9 Allow incoming Packet Too Big messages only for existing sessions

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.10 Allow outgoing Packet Too Big messages

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.11 Allow incoming time exceeded code 0 messages only for existing sessions

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.12 Allow incoming time exceeded code 1 messages

The rule(s) suggested by Section 2.7 will also meet this recommendation.

Note
Note: in the example RFC script, non-established/related messages are allowed through for this type. It is not clear why since code 0 and not code 1 messages are listed as part of the establishment of communications. Code 1 messages are listed as less essential for propagation over the network.
The behaviour implemented here is as per destination unreachable messages, so the same as the incoming time exceeded code 0 messages example.

2.13 Allow outgoing time exceeded code 0 messages

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.14 Allow outgoing time exceeded code 1 messages

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.15 Allow incoming parameter problem code 1 and 2 messages for an existing session

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.16 Allow outgoing parameter problem code 1 and code 2 messages

The rule(s) suggested by Section 2.7 will also meet this recommendation.

2.17 Allow incoming and outgoing parameter problem code 0 messages

From the RFC it is not really necessary to allow these messages. Sanewall handles this automatically (by dropping them) unless you set up an explicit route for the packets using the icmpv6 type bad-header.

2.18 Drop NS/NA messages both incoming and outgoing

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
Hosts and bridges need to allow these messages. See ipv6neigh - IPv6 Neighbour discovery - complex service .

2.19 Drop RS/RA messages both incoming and outgoing

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
Hosts and bridges need to allow these messages. See ipv6router - IPv6 Router discovery - complex service .

2.20 Drop Redirect messages both incoming and outgoing

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type redirect -j DROP
```

2.21 Drop incoming and outgoing Multicast Listener queries (MLDv1 and MLDv2)

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 130 -j DROP
```

2.22 Drop incoming and outgoing Multicast Listener reports (MLDv1)

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 131 -j DROP
```

2.23 Drop incoming and outgoing Multicast Listener Done messages (MLDv1)

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 132 -j DROP
```

2.24 Drop incoming and outgoing Multicast Listener reports (MLDv2)

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 143 -j DROP
```

2.25 Drop router renumbering messages

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 138 -j DROP
```

2.26 Drop node information queries (139) and replies (140)

Sanewall handles this automatically unless you set up an explicit route for the packets.

Note
At some point Sanewall may have a helper command added to simplify allowing these messages on a host/bridge. Meantime this is an example of the relevant ip6tables command:

```
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 139 -j DROP
ip6tables -A icmpv6-filter -p icmpv6 --icmpv6-type 140 -j DROP
```

2.27 If there are mobile ipv6 home agents present on the trusted side allow

At some point Sanewall may have a helper command added to simplify this setup. Meantime this is an example of the relevant ip6tables commands from the RFC script:

```
#incoming Home Agent address discovery request
ip6tables -A icmpv6-filter -p icmpv6 -d $inner_prefix \
     --icmpv6-type 144 -j ACCEPT
#outgoing Home Agent address discovery reply
ip6tables -A icmpv6-filter -p icmpv6 -s $inner_prefix \
     --icmpv6-type 145 -j ACCEPT
#incoming Mobile prefix solicitation
ip6tables -A icmpv6-filter -p icmpv6 -d $inner_prefix \
     --icmpv6-type 146 -j ACCEPT
#outgoing Mobile prefix advertisement
ip6tables -A icmpv6-filter -p icmpv6 -s $inner_prefix \
     --icmpv6-type 147 -j ACCEPT
```

2.28 If there are roaming mobile nodes present on the trusted side allow

At some point Sanewall may have a helper command added to simplify this setup. Meantime this is an example of the relevant ip6tables commands from the RFC script:

```
#outgoing Home Agent address discovery request
ip6tables -A icmpv6-filter -p icmpv6 -s $inner_prefix \
```

```
     --icmpv6-type 144 -j ACCEPT
#incoming Home Agent address discovery reply
ip6tables -A icmpv6-filter -p icmpv6 -d $inner_prefix \
     --icmpv6-type 145 -j ACCEPT
#outgoing Mobile prefix solicitation
ip6tables -A icmpv6-filter -p icmpv6 -s $inner_prefix \
     --icmpv6-type 146 -j ACCEPT
#incoming Mobile prefix advertisement
ip6tables -A icmpv6-filter -p icmpv6 -d $inner_prefix \
     --icmpv6-type 147 -j ACCEPT
```

2.29 Drop everything else

Sanewall handles this automatically unless you set up an explicit route for the packets.

Chapter 3

Sanewall Reference

3.1 Sanewall program: sanewall

Name

sanewall — an easy to use but powerful iptables stateful firewall

Synopsis

`sanewall`

`sudo -E sanewall` panic [`IP`]

`sanewall` *`command`* [-- *`conf-arg`*...]

`sanewall` *`CONFIGFILE`* [start | debug | try] [-- *`conf-arg`*...]

Description

Running **sanewall** invokes **iptables(8)** to manipulate your firewall.

Run without any arguments, **sanewall** will present some help on usage.

When given *`CONFIGFILE`*, **sanewall** will use the named file instead of `/etc/sanewall/sanewall.conf` as its configuration. If no command is given, **sanewall** assumes `try`.

It is possible to pass arguments for use by the configuration file separating any *`conf-arg`* values from the rest of the arguments with **--**. The arguments are accessible in the configuration using standard **bash(1)** syntax e.g. $1, $2, etc.

Sanewall is a fork of FireHOL. existing FireHOL configurations should be compatible with Sanewall, but please see the section called "Compatibility" for any differences in behaviour.

Panic

To block all communication, invoke **sanewall** with the `panic` command.

Sanewall removes all rules from the running firewall and then DROPs all traffic on all iptables tables (mangle, nat, filter) and pre-defined chains (PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING).

DROPing is not done by changing the default policy to DROP, but by adding one rule per table/chain to drop all traffic. This allows systems which do not reset all the chains to ACCEPT when starting to function correctly.

When activating panic mode, Sanewall checks for the existence of the `SSH_CLIENT` shell environment variable, which is set by **ssh**. If it finds this, then panic mode will allow the established SSH connection specified in this variable to operate.

Note
In order for Sanewall to see the environment variable you must ensure that it is preserved. For **sudo** use the `-E` and for **su** omit the - (minus sign).

If `SSH_CLIENT` is not set, the `IP` after the panic argument allows you to give an IP address for which all established connections between the IP address and the host in panic will be allowed to continue.

Commands

start, restart
Activates the firewall configuration from `/etc/sanewall/sanewall.conf`.

Use of the term `restart` is allowed for compatibility with common init implementations.

try Activates the firewall, waiting for the user to type the word **`commit`**. If this word is not typed within 30 seconds, the previous firewall is restored.

stop
Stops a running iptables firewall by clearing all of the tables and chains and setting the default policies to ACCEPT. This will allow all traffic to pass unchecked.

condrestart
Starts the Sanewall firewall only if it is not already active. It does not detect a modified configuration file, only verifies that Sanewall has been started in the past and not stopped yet.

status
Shows the running firewall, using **/sbin/iptables -nxvL | less**.

save
Start the firewall and then save it using **/sbin/iptables-save** to `/etc/sysconfig/iptables`.

The required kernel modules are saved to an executable shell script `/var/spool/sanewall/last_save_modules.sh`, which can be called during boot if a firewall is to be restored.

Note
External changes may cause a firewall restored after a reboot to not work as intended where starting the firewall with Sanewall will work.
This is because as part of starting a firewall, Sanewall checks some changeable values. For instance the current kernel configuration is checked (for client port ranges), and RPC servers are queried (to allow correct functioning of the NFS service).

debug
Parses the configuration file but instead of activating it, Sanewall shows the generated iptables statements.

explain
Enters an interactive mode where Sanewall accepts normal configuration commands and presents the generated iptables commands for each of them, together with some reasoning for its purpose

Additionally, Sanewall automatically generates a configuration script based on the successful commands given.

Some extra commands are available in `explain` mode.

SPECIAL COMMANDS IN EXPLAIN MODE

help
Present some help

show
Present the generated configuration

quit
Exit interactive mode and quit

helpme, wizard
Tries to guess the Sanewall configuration needed for the current machine.

Sanewall will not stop or alter the running firewall. The configuration file is given in the standard output of **sAnewall**, thus **`sanewall helpme > /tmp/sanewall.conf`** will produce the output in `/tmp/sanewall.conf`.

The generated Sanewall configuration *must* be edited before use on your systems. You are required to take a number of decisions; the comments in the generated file will instruct you in the choices you must make.

Compatibility

Sanewall should be largely compatible with all existing FireHOL configurations.

If you are using any variable starting "FIREHOL_" in your configuration, you will need to rename it to "SANEWALL_". See control variables: sanewall-variables(5) for a list of all variables used to control Sanewall.

In addition the default values of SANEWALL_*_ACTIVATION_POLICY, for each of INPUT, OUTPUT and FORWARD have been changed to DROP. See the entries under control variables: sanewall-variables(5) for details and how to obtain the original behaviour.

Files

`/etc/sanewall/sanewall.conf`

See Also

Sanewall configuration: sanewall.conf(5)
control variables: sanewall-variables(5)
Sanewall Manual: sanewall-manual.pdf
Sanewall Online Documentation

3.2 Sanewall configuration: sanewall.conf

Name

sanewall.conf — Sanewall configuration file

Description

`/etc/sanewall/sanewall.conf` is the default configuration file for Sanewall program: sanewall(1). It defines the stateful firewall that will be produced.

A configuration file starts with an optional version indicator which looks like this:

```
version 5
```

See version config helper: sanewall-version(5) for full details.

A configuration file contains one or more **interface** definitions, which look like this:

```
interface eth0 lan
  client all accept # This host can access any remote service
  server ssh accept # Remote hosts can access SSH on local server
  # ...
```

The above definition has name "lan" and specifies a network interface (eth0). A definition may contain zero or more subcommands. See interface definition: sanewall-interface(5) for full details.

A configuration file contains one or more **router** definitions, which look like this:

```
DMZ_IF=eth0
WAN_IF=eth1
router wan2dmz inface ${WAN_IF} outface ${DMZ_IF}
  route http accept  # Hosts on WAN may access HTTP on hosts in DMZ
  server ssh accept  # Hosts on WAN may access SSH on hosts in DMZ
  client pop3 accept # Hosts in DMZ may access POP3 on hosts on WAN
  # ...
```

The above definition has name "wan2dmz" and specifies incoming and outgoing network interfaces (eth1 and eth0) using variables. A definition may contain zero or more subcommands. Note that a router is not required to specify network interfaces to operate on. See router definition: sanewall-router(5) for full details.

It is simple to add extra service definitions which can then be used in the same way as those provided as standard. See the section called “Adding Services”.

The configuration file is parsed as a **bash(1)** script, allowing you to set up and use variables, flow control and external commands.

Special control variables: sanewall-variables(5) may be set up and used outside of any definition as can the functions in the section called "Configuration Helper Commands" and the section called "Helper Commands".

Variables Available

The following variables are made available in the Sanewall configuration file and can be accessed as ${VARIABLE}.

UNROUTABLE_IPS
: This variable includes the IPs from both PRIVATE_IPS and RESERVED_IPS. It is useful to restrict traffic on interfaces and routers accepting Internet traffic, for example:

```
interface eth0 internet src not "${UNROUTABLE_IPS}"
```

PRIVATE_IPS
: This variable includes all the IP addresses defined as Private or Test by RFC 3330.

: You can override the default values by creating a file called /etc/sanewall/PRIVATE_IPS.

RESERVED_IPS
: This variable includes all the IP addresses defined by IANA as reserved.

: You can override the default values by creating a file called /etc/sanewall/RESERVED_IPS.

: Now that IPv4 address space has all been allocated there is very little reason that this value will need to change in future.

MULTICAST_IPS
: This variable includes all the IP addresses defined as Multicast by RFC 3330.

: You can override the default values by creating a file called /etc/sanewall/MULTICAST_IPS.

Adding Services

To define new services you add the appropriate lines before using them later in the configuration file.

The following are required:

```
server_myservice_ports="proto/sports"
client_myservice_ports="cports"
```

proto is anything **iptables(8)** accepts e.g. "tcp", "udp", "icmp", including numeric protocol values.

sports is the ports the server is listening at. It is a space-separated list of port numbers, names and ranges (from:to). The keyword any will match any server port.

cports is the ports the client may use to initiate a connection. It is a space-separated list of port numbers, names and ranges (from:to). The keyword `any` will match any client port. The keyword `default` will match default client ports. For the local machine (e.g. a **client** within an **interface**) it resolves to **sysctl** variable `net.ipv4.ip_local_port_range` (or `/proc/sys/net/ipv4/ip_local_port_range`). For a remote machine (e.g. a **client** within an **interface** or anything in a **router**) it resolves to the variable `DEFAULT_CLIENT_PORTS` (see control variables: sanewall-variables(5)).

The following are optional:

```
require_myservice_modules="modules"
require_myservice_nat_modules="nat-modules"
```

The named kernel modules will be loaded when the definition is used. The NAT modules will only be loaded if `SANEWALL_NAT` is non-zero (see control variables: sanewall-variables(5)).

For example, for a service named `daftnet` that listens at two ports, port 1234 TCP and 1234 UDP where the expected client ports are the default random ports a system may choose, plus the same port numbers the server listens at, with further dynamic ports requiring kernel modules to be loaded:

```
version 5

server_daftnet_ports="tcp/1234 udp/1234"
client_daftnet_ports="default 1234"
require_daftnet_modules="ip_conntrack_daftnet"
require_daftnet_nat_modules="ip_nat_daftnet"

interface eth0 lan0
    server daftnet accept

interface eth1 lan1
    client daftnet reject

router lan2lan inface eth0 outface eth1
    route daftnet accept
```

Where multiple ports are provides (as per the example), Sanewall simply determines all of the combinations of client and server ports and generates multiple iptables statements to match them.

To create more complex rules, or stateless rules, you will need to create a bash function prefixed `rules_` e.g. `rules_myservice`. The best reference is the many such functions in the main **sanewall** executable.

When adding a service which uses modules, or via a custom function, you may also wish to include the following:

```
ALL_SHOULD_ALSO_RUN="${ALL_SHOULD_ALSO_RUN} myservice"
```

which will ensure your service is set-up correctly as part of the **all** service.

Note
To allow definitions to be shared you can instead create files and install them in the `/etc/sanewall/services` directory with a `.conf` extension.
The first line must read:

```
FHVER 1:213
```

1 is the service definition API version. It will be changed if the API is ever modified. 213 refers to a FireHOL version and is retained for compatibility.
Sanewall will refuse to run if the API version does not match the expected one. The minor number is ignored.
At version 1:213, the API and therefore service definitions are compatible with FireHOL.

Definitions

interface definition: sanewall-interface(5)
router definition: sanewall-router(5)

Subcommands

policy command: sanewall-policy(5)
protection command: sanewall-protection(5)
server, route commands: sanewall-server(5)
client command: sanewall-client(5)
group command: sanewall-group(5)

Helper Commands

These helpers can be used in **interface** and **router** definitions as well as before them.

iptables helper: sanewall-iptables(5)
masquerade helper: sanewall-masquerade(5)

This helper can be used in **router** definitions as well as before any **router** or **interface**.

tcpmss helper: sanewall-tcpmss(5)

Configuration Helper Commands

These helpers should only be used outside of **interface** and **router** definitions (i.e. before the first interface is defined).

version config helper: sanewall-version(5)
action config helper: sanewall-action(5)
blacklist config helper: sanewall-blacklist(5)
classify config helper: sanewall-classify(5)
connmark config helper: sanewall-connmark(5)
dscp config helper: sanewall-dscp(5)
mac config helper: sanewall-mac(5)
mark config helper: sanewall-mark(5)
nat, snat, dnat, redirect config helpers: sanewall-nat(5)
transparent_proxy, transparent_squid helpers: sanewall-transparent_proxy(5)
tos config helper: sanewall-tos(5)
tosfix config helper: sanewall-tosfix(5)

See Also

Sanewall program: sanewall(1)
control variables: sanewall-variables(5)
services list: sanewall-services(5)
actions for rules: sanewall-actions(5)
Sanewall Manual: sanewall-manual.pdf
Sanewall Online Documentation

3.3 control variables: sanewall-variables

Name

sanewall-variables — Variables controlling Sanewall

Description

There are a number of variables that control the behaviour of Sanewall.

All variables may be set in the main Sanewall configuration file `/etc/sanewall/sanewall.conf`.

Variables which affect the runtime but not the created firewall may also be set as environment variables before running **sanewall**. These can change the default values but will be overwritten by values set in the configuration file. If a variable can be set by an environment variable it is specified below.

Sanewall also sets some variables before processing the configuration file which you can use as part of your configuration. These are described in Sanewall configuration: sanewall.conf(5).

Variables

DEFAULT_INTERFACE_POLICY

This variable controls the default action to be taken on traffic not matched by any rule within an interface. It can be overridden using policy command: sanewall-policy(5).

Packets that reach the end of an interface without an action of return or accept are logged. You can control the frequency of this logging by altering `SANEWALL_LOG_FREQUENCY`.

Default:

```
DEFAULT_INTERFACE_POLICY="DROP"
```

Example:

```
DEFAULT_INTERFACE_POLICY="REJECT"
```

DEFAULT_ROUTER_POLICY

This variable controls the default action to be taken on traffic not matched by any rule within a router. It can be overridden using policy command: sanewall-policy(5).

Packets that reach the end of a router without an action of return or accept are logged. You can control the frequency of this logging by altering `SANEWALL_LOG_FREQUENCY`.

Default:

```
DEFAULT_ROUTER_POLICY="RETURN"
```

Example:

```
DEFAULT_ROUTER_POLICY="REJECT"
```

UNMATCHED_INPUT_POLICY, UNMATCHED_OUTPUT_POLICY, UNMATCHED_FORWARD_POLICY

These variables control the default action to be taken on traffic not matched by any interface or router definition that was incoming, outgoing or for forwarding respectively. Any supported value from actions for rules: sanewall-actions(5) may be set.

All packets that reach the end of a chain are logged, regardless of these settings. You can control the frequency of this logging by altering SANEWALL_LOG_FREQUENCY.

Defaults:

```
UNMATCHED_INPUT_POLICY="DROP"
UNMATCHED_OUTPUT_POLICY="DROP"
UNMATCHED_FORWARD_POLICY="DROP"
```

Example:

```
UNMATCHED_INPUT_POLICY="REJECT"
UNMATCHED_OUTPUT_POLICY="REJECT"
UNMATCHED_FORWARD_POLICY="REJECT"
```

SANEWALL_INPUT_ACTIVATION_POLICY, SANEWALL_OUTPUT_ACTIVATION_POLICY, SANEWALL_FORWARD_ACTIVATION_POLICY, SANEWALL_ESTABLISHED_ACTIVATION_ACCEPT

These variables control the default action to be taken on traffic during firewall activation for incoming, outgoing and forwarding respectively. Acceptable values are ACCEPT, DROP and REJECT. They may be set as environment variables.

During activation, Sanewall creates temporary rules to ALLOW already established traffic (new connections honour the appropriate variable). Set SANEWALL_ESTABLISHED_ACTIVATION_ACCEPT to 0 to prevent this.

Unlike **FireHOL** which defaults all values to ACCEPT, **Sanewall** defaults all values to DROP.

If you wish to reinstate the old **FireHOL** behaviour, set these values to ACCEPT. Please do not do so if you are using all or any to match traffic; connections established during activation will continue even if they would not be allowed once the firewall is established.

Defaults:

```
SANEWALL_INPUT_ACTIVATION_POLICY="DROP"
SANEWALL_OUTPUT_ACTIVATION_POLICY="DROP"
SANEWALL_FORWARD_ACTIVATION_POLICY="DROP"
SANEWALL_ESTABLISHED_ACTIVATION_ACCEPT="1"
```

Example:

```
UNMATCHED_INPUT_POLICY="ACCEPT"
UNMATCHED_OUTPUT_POLICY="ACCEPT"
UNMATCHED_FORWARD_POLICY="ACCEPT"
SANEWALL_ESTABLISHED_ACTIVATION_ACCEPT="0"
```

SANEWALL_LOG_MODE

This variable controls method that Sanewall uses for logging.

Acceptable values are `LOG` (normal syslog) and `ULOG` (netfilter ulogd). When `ULOG` is selected, `SANEWALL_LOG_LEVEL` is ignored.

Default:

```
SANEWALL_LOG_MODE="LOG"
```

Example:

```
SANEWALL_LOG_MODE="ULOG"
```

To see the available options run: **`/sbin/iptables -j LOG --help`** or **`/sbin/iptables -j ULOG --help`**

SANEWALL_LOG_LEVEL

This variable controls the level at which events will be logged to syslog.

To avoid packet logs appearing on your console you should ensure klogd only logs traffic that is more important than that produced by Sanewall.

Use the following option to choose an iptables log level (alpha or numeric) which is higher than the `-c` of klogd.

iptables	klogd	description
emerg (0)	0	system is unusable
alert (1)	1	action must be taken immediately
crit (2)	2	critical conditions
error (3)	3	error conditions
warning (4)	4	warning conditions
notice (5)	5	normal but significant condition
info (6)	6	informational
debug (7)	7	debug-level messages

Table 3.1: iptables/klogd levels

Note
The default for klogd is generally to log everything (7 and lower) and the default level for iptables is to log as warnings (4).

SANEWALL_LOG_OPTIONS

This variable controls the way in which events will be logged to syslog.

Default:

```
SANEWALL_LOG_OPTIONS="--log-level warning"
```

Example:

```
SANEWALL_LOG_OPTIONS="--log-level info \
   --log-tcp-options --log-ip-options"
```

To see the available options run: **/sbin/iptables -j LOG --help**

SANEWALL_LOG_FREQUENCY, SANEWALL_LOG_BURST

These variables control the frequency that each logging rule will write events to syslog. SANEWALL_LOG_FREQUENCY is set to the maximum average frequency and SANEWALL_LOG_BURST specifies the maximum initial number.

Default:

```
SANEWALL_LOG_FREQUENCY="1/second"
SANEWALL_LOG_BURST="5"
```

Example:

```
SANEWALL_LOG_FREQUENCY="30/minute"
SANEWALL_LOG_BURST="2"
```

To see the available options run: **/sbin/iptables -m limit --help**

SANEWALL_LOG_PREFIX

This value is added to the contents of each logged line for easy detection of Sanewall lines in the system logs. By default it is empty.

Default:

```
SANEWALL_LOG_PREFIX=""
```

Example:

```
SANEWALL_LOG_PREFIX="SANEWALL:"
```

SANEWALL_DROP_INVALID

If set to 1, this variable causes Sanewall to drop all packets matched as INVALID in the **iptables(8)** connection tracker.

Note

You can use protection command: sanewall-protection(5) to control matching of INVALID packets and others on per-interface and per-router basis.

Default:

```
SANEWALL_DROP_INVALID="0"
```

Example:

```
SANEWALL_DROP_INVALID="1"
```

DEFAULT_CLIENT_PORTS

This variable controls the port range that is used when a remote client is specified. For clients on the local host, Sanewall finds the exact client ports by querying the kernel options.

Default:

```
DEFAULT_CLIENT_PORTS="1000:65535"
```

Example:

```
DEFAULT_CLIENT_PORTS="0:65535"
```

SANEWALL_NAT

If set to 1, this variable causes Sanewall to load the NAT kernel modules. If you make use of the NAT helper commands, the variable will be set to 1 automatically. It may be set as an environment variable.

Default:

```
SANEWALL_NAT="0"
```

Example:

```
SANEWALL_NAT="1"
```

SANEWALL_ROUTING

If set to 1, this variable causes Sanewall to enable routing in the kernel. If you make use of **router** definitions or certain helper commands the variable will be set to 1 automatically. It may be set as an environment variable.

Default:

```
SANEWALL_ROUTING="0"
```

Example:

```
SANEWALL_ROUTING="1"
```

SANEWALL_AUTOSAVE

This variable specifies the file that will be created when Sanewall program: sanewall(1) is called with the `save` argument. It may be set as an environment variable.

If the variable is empty, Sanewall will try to detect where to save the file. Currently `/etc/sysconfig/iptables` (RedHat) and `/var/lib/iptables/autosave` (Debian) are tried in order, based on the existence of the directory.

Default:

```
SANEWALL_AUTOSAVE=""
```

Example:

```
SANEWALL_AUTOSAVE="/tmp/sanewall-saved.txt"
```

SANEWALL_LOAD_KERNEL_MODULES

If set to 0, this variable forces Sanewall to not load any kernel modules. It is needed only if the kernel has modules statically included and in the rare event that Sanewall cannot access the kernel configuration. It may be set as an environment variable.

Default:

```
SANEWALL_LOAD_KERNEL_MODULES="1"
```

Example:

```
SANEWALL_LOAD_KERNEL_MODULES="0"
```

SANEWALL_TRUST_LOOPBACK

If set to 0, the loopback device "lo" will not be trusted and you can write standard firewall rules for it.

Warning
If you do not set up appropriate rules, local processes will not be able to communicate with each other which can result in serious breakages.

By default "lo" is trusted and all `INPUT` and `OUTPUT` traffic is accepted (forwarding is not included).

Default:

```
SANEWALL_TRUST_LOOPBACK="1"
```

Example:

```
SANEWALL_TRUST_LOOPBACK="0"
```

SANEWALL_DROP_ORPHAN_TCP_ACK_FIN

If set to 1, Sanewall will drop all TCP connections with ACK FIN set without logging them.

In busy environments the iptables connection tracker removes connection tracking list entries as soon as it receives a FIN. This makes the ACK FIN appear as an invalid packet which will normally be logged by Sanewall.

Default:

```
SANEWALL_DROP_ORPHAN_TCP_ACK_FIN="0"
```

Example:

```
SANEWALL_DROP_ORPHAN_TCP_ACK_FIN="1"
```

SANEWALL_DEBUGGING

If set to a non-empty value, switches on debug output so that it is possible to see what processing Sanewall is doing.

Note

This variable can *only* be set as an environment variable, since it is processed before any configuration files are read.

Default:

```
SANEWALL_DEBUGGING=""
```

Example:

```
SANEWALL_DEBUGGING="Y"
```

WAIT_FOR_IFACE

If set to the name of a network device (e.g. eth0), Sanewall will wait until the device is up (or until 60 seconds have elapsed) before continuing.

Note

This variable can *only* be set as an environment variable, since it determines when the main configuration file will be processed.

A device does not need to be up in order to have firewall rules created for it, so this option should only be used if you have a specific need to wait (e.g. the network must be queried to determine the hosts or ports which will be firewalled).

Default:

```
WAIT_FOR_IFACE=""
```

Example:

```
WAIT_FOR_IFACE="eth0"
```

See also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
nat, snat, dnat, redirect config helpers: sanewall-nat(5)
administration tool for IPv4 firewalls: iptables(8)

3.4 interface definition: sanewall-interface

Name

sanewall-interface — create an interface definition

Synopsis

`interface` *`real-interface name`* [*`rule-params`*]

Description

An **interface** definition creates a firewall for protecting the host on which the firewall is running.

The default policy is DROP, so that if no subcommands are given, the firewall will just drop all incoming and outgoing traffic using this interface.

The behaviour of the defined interface is controlled by adding subcommands (listed in the section called "See Also").

Note
Forwarded traffic is never matched by the **interface** rules, even if it was originally destined for the firewall but was redirected using NAT. Any traffic to be passed through the firewall for whatever reason must be in a **router** (see router definition: sanewall-router(5)).

Parameters

`real-interface`

This is the interface name as shown by **`ip link show`**. Generally anything **iptables** accepts is valid.

The + (plus sign) after some text will match all interfaces that start with this text.

Multiple interfaces may be specified by enclosing them within quotes, delimited by spaces for example:

```
interface "eth0 eth1 ppp0" myname
```

`name`

This is a name for this interface. You should use short names (10 characters maximum) without spaces or other symbols.

A name should be unique for all Sanewall interface and router definitions.

---

rule-params

The set of rule parameters to further restrict the traffic that is matched to this interface.

See optional rule parameters: sanewall-rule-params(5) for information on the parameters that can be used. Some examples:

```
interface eth0 intranet src 192.0.2.0/24

interface eth0 internet src not "${UNROUTABLE_IPS}"
```

See Sanewall configuration: sanewall.conf(5) for an explanation of `${UNROUTABLE_IPS}`.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
router definition: sanewall-router(5)
policy command: sanewall-policy(5)
protection command: sanewall-protection(5)
client command: sanewall-client(5)
server, route commands: sanewall-server(5)
group command: sanewall-group(5)
iptables helper: sanewall-iptables(5)
masquerade helper: sanewall-masquerade(5)

3.5 router definition: sanewall-router

Name

sanewall-router — create a router definition

Synopsis

`router` *`name`* [*`rule-params`*]

Description

A **router** definition consists of a set of rules for traffic passing through the host running the firewall.

The default policy for router definitions is RETURN, meaning packets are not dropped by any particular router. Packets not matched by any router are dropped at the end of the firewall.

The behaviour of the defined router is controlled by adding subcommands (listed in the section called "See Also").

Parameters

`name`

This is a name for this router. You should use short names (10 characters maximum) without spaces or other symbols.

A name should be unique for all Sanewall interface and router definitions.

`rule-params`

The set of rule parameters to further restrict the traffic that is matched to this router.

See optional rule parameters: sanewall-rule-params(5) for information on the parameters that can be used. Some examples:

```
router mylan inface ppp+ outface eth0 src not ${UNROUTABLE_IPS}

router myrouter
```

See Sanewall configuration: sanewall.conf(5) for an explanation of `${UNROUTABLE_IPS}`.

Working with routers

Routers create stateful **iptables** rules which match traffic in both directions.

To match some client or server traffic, the input/output interface or source/destination of the request must be specified. All `inface`/`outface` and `src`/`dst` optional rule parameters: sanewall-rule-params(5) can be given on the router statement (in which case they will be applied to all subcommands for the router) or just within the subcommands of the router.

For example, to define a router which matches requests from any PPP interface and destined for eth0, and on this allowing HTTP servers (on eth0) to be accessed by clients (from PPP) and SMTP clients (from eth0) to access any servers (on PPP):

```
router mylan inface ppp+ outface eth0
  server http accept
  client smtp accept
```

Note
The `client` subcommand reverses any optional rule parameters passed to the **router**, in this case the inface and outface.

Equivalently, to define a router which matches all forwarded traffic and within the the router allow HTTP servers on eth0 to be accessible to PPP and any SMTP servers on PPP to be accessible from eth0:

```
router mylan
  server http accept inface ppp+ outface eth0
  server smtp accept inface eth0 outface ppp
```

Note
In this instance two `server` subcommands are used since there are no parameters on the router to reverse. Avoid the use of the `client` subcommand in routers unless the inputs and outputs are defined as part of the router.

Any number of routers can be defined and the traffic they match can overlap. Since the default policy is RETURN, any traffic that is not matched by any rules in one will proceed to the next, in order, until none are left.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
policy command: sanewall-policy(5)
protection command: sanewall-protection(5)
client command: sanewall-client(5)
server, route commands: sanewall-server(5)
group command: sanewall-group(5)
iptables helper: sanewall-iptables(5)
masquerade helper: sanewall-masquerade(5)
tcpmss helper: sanewall-tcpmss(5)

3.6 policy command: sanewall-policy

Name

sanewall-policy — set default action for a definition

Synopsis

`policy` *`action`*

Description

The **policy** subcommand defines the default policy for an interface or router.

The *`action`* can be any of the actions listed in actions for rules: sanewall-actions(5).

Note
Change the default policy of a router only if you understand clearly what will be matched by the router statement whose policy is being changed.
It is common to define overlapping router definitions. Changing the policy to anything other than the default `return` may cause strange results for your configuration.

Warning
Do not set a policy to `accept` unless you fully trust all hosts that can reach the interface. Sanewall CANNOT create valid "accept by default" firewalls. See this FireHOL bug report for some more information and history.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
actions for rules: sanewall-actions(5)

3.7 protection command: sanewall-protection

Name

sanewall-protection — add extra protections to a definition

Synopsis

`protection` [reverse] *`flood-protection-type`* [requests/period [burst]]

`protection` [reverse] strong [requests/period [burst]]

`protection` [reverse] bad-packets | *`packet-protection-type`*

Description

The **protection** subcommand sets protection rules on an interface or router.

Flood protections honour the options `requests/period` and `burst`. They are used to limit the rate of certain types of traffic.

The default rate Sanewall uses is 100 operations per second with a burst of 50. Run **`iptables -m limit --help`** for more information.

The protection type `strong` will switch on all protections (both packet and flood protections) except `all-floods`. It has aliases `full` and `all`.

The protection type `bad-packets` will switch on all packet protections but not flood protections.

You can specify multiple protection types by using multiple **protection** commands or in a single command by enclosing the types in quotes.

Note
On a router, protections are normally set up on *`inface`*.
The `reverse` option will set up the protections on *`outface`*. You must use it as the first keyword.

Packet protection types

invalid
: Drops all incoming invalid packets, as detected INVALID by the connection tracker.

  See also `SANEWALL_DROP_INVALID` in control variables: sanewall-variables(5) which allows setting this function globally.

---

fragments
Drops all packet fragments.

This rule will probably never match anything since **iptables(8)** reconstructs all packets automatically before the firewall rules are processed whenever connection tracking is running.

new-tcp-w/o-syn
Drops all TCP packets that initiate a socket but have not got the SYN flag set.

malformed-xmas
Drops all TCP packets that have all TCP flags set.

malformed-null
Drops all TCP packets that have all TCP flags unset.

malformed-bad
Drops all TCP packets that have illegal combinations of TCP flags set.

Flood protection types

icmp-floods [requests/period [burst]]
Allows only a certain amount of ICMP echo requests.

syn-floods [requests/period [burst]]
Allows only a certain amount of new TCP connections.

Be careful to not set the rate too low as the rule is applied to all connections regardless of their final result (rejected, dropped, established, etc).

all-floods [requests/period [burst]]
Allows only a certain amount of new connections.

Be careful to not set the rate too low as the rule is applied to all connections regardless of their final result (rejected, dropped, established, etc).

Examples

```
protection strong

protection "invalid new-tcp-w/o-syn"

protection syn-floods 90/sec 40
```

Bugs

When using multiple types in a single command, if the quotes are forgotten, incorrect rules will be generated without warning.

When using multiple types in a single command, Sanewall will silently ignore any types that come after a group type (`bad-packets`, `strong` and its aliases). Only use group types on their own line.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)

3.8 server, route commands: sanewall-server

Name

sanewall-server, sanewall-route — accept requests to a service

Synopsis

`server` *`service action`* [*`rule-params`*]

`route` *`service action`* [*`rule-params`*]

Description

The **server** subcommand defines a server of a service on an interface or router. Any `rule-params` given to a parent interface or router are inherited by the server.

For Sanewall a server is the destination of a request. Even though this is more complex for some multi-socket services, to Sanewall a server always accepts requests.

The **route** subcommand is an alias for **server** which may only be used in routers.

The *`service`* parameter is one of the supported service names from services list: sanewall-services(5). Multiple services may be specified, space delimited in quotes.

The *`action`* can be any of the actions listed in actions for rules: sanewall-actions(5).

The *`rule-params`* define a set of rule parameters to further restrict the traffic that is matched to this service. See optional rule parameters: sanewall-rule-params(5) for more details.

Examples

```
server smtp accept

server "smtp pop3" accept

server smtp accept src 192.0.2.1

server smtp accept log "mail packet" src 192.0.2.1
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
services list: sanewall-services(5)
actions for rules: sanewall-actions(5)
optional rule parameters: sanewall-rule-params(5)

3.9 client command: sanewall-client

Name

sanewall-client — accept replies from a service

Synopsis

`client` *`service action`* [*`rule-params`*]

Description

The **client** subcommand defines a client of a service on an interface or router. Any `rule-params` given to a parent interface or router are inherited by the client, but are reversed.

For Sanewall a client is the source of a request. Even though this is more complex for some multi-socket services, to Sanewall a client always initiates the connection.

The *`service`* parameter is one of the supported service names from services list: sanewall-services(5). Multiple services may be specified, space delimited in quotes.

The *`action`* can be any of the actions listed in actions for rules: sanewall-actions(5).

The *`rule-params`* define a set of rule parameters to further restrict the traffic that is matched to this service. See optional rule parameters: sanewall-rule-params(5) for more details.

Examples

```
client smtp accept

client "smtp pop3" accept

client smtp accept src 192.0.2.1

client smtp accept log "mail packet" src 192.0.2.1
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
services list: sanewall-services(5)
actions for rules: sanewall-actions(5)
optional rule parameters: sanewall-rule-params(5)

3.10 group command: sanewall-group

Name

sanewall-group — group commands with common options

Synopsis

`group` with [*`rule-params`*]

`group` end

Description

The **group** command allows you to group together multiple **client** and **server** commands.

Grouping commands with common options (see optional rule parameters: sanewall-rule-params(5)) allows the option values to be checked only once in the generated firewall rather than once per service, making it more efficient.

Nested groups may be used.

Examples

This:

```
interface any world
    client all accept
    server http accept

    # Provide these services to trusted hosts only
    server "ssh telnet" accept src "192.0.2.1 192.0.2.2"
```

can be replaced to produce a more efficient firewall by this:

```
interface any world
    client all accept
    server http accept

    # Provide these services to trusted hosts only
    group with src "192.0.2.1 192.0.2.2"
        server ssh telnet
        server ssh accept
    group end
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
optional rule parameters: sanewall-rule-params(5)

3.11 version config helper: sanewall-version

Name

sanewall-version — set version number of configuration file

Synopsis

`version` 5

Description

The **version** helper command states the configuration file version.

If the value passed is newer than the running version of Sanewall supports, Sanewall will not run.

You do not have to specify a version number for a configuration file, but by doing so you will prevent Sanewall trying to process a file which it cannot handle.

The value that Sanewall expects is increased every time that the configuration file format changes.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)

3.12 action config helper: sanewall-action

Name

sanewall-action — set up custom filter actions

Synopsis

`action` chain *`name action`*

Description

The **action** helper command creates an iptables chain which can be used to control the action of other firewall rules once the firewall is running.

For example, you can setup the custom action ACT1, which by default is ACCEPT, but can be dynamically changed to DROP, REJECT or RETURN (and back) without restarting the firewall.

The *`name`* can be any chain name accepted by iptables. You should try to keep it within 5 and 10 characters.

Note
The *`name`*s created with this command are case-sensitive.

The *`action`* can be any of those supported by Sanewall (see actions for rules: sanewall-actions(5)). Only ACCEPT, REJECT, DROP, RETURN have any meaning in this instance.

Examples

To create a custom chain and have some rules use it:

```
action chain ACT1 accept

interface any world
    server smtp ACT1
    client smtp ACT1
```

Once the firewall is running you can dynamically modify the behaviour of the chain from the Linux command-line, as detailed below:

To insert a DROP action at the start of the chain to override the default action (ACCEPT):

```
iptables -t filter -I ACT1 -j DROP
```

To delete the DROP action from the start of the chain to return to the default action:

```
iptables -t filter -D ACT1 -j DROP
```

Note
If you delete all of the rules in the chain, the default will be to RETURN, in which case the behaviour will be as if any rules with the action were not present in the configuration file.

You can also create multiple chains simultaneously. To create 3 ACCEPT and 3 DROP chains you can do the following:

```
action chain "ACT1 ACT2 ACT3" accept
action chain "ACT4 ACT5 ACT6" drop
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
actions for rules: sanewall-actions(5)
administration tool for IPv4 firewalls: iptables(8)

3.13 blacklist config helper: sanewall-blacklist

Name

sanewall-blacklist — set up a unidirectional or bidirectional blacklist

Synopsis

`blacklist` [full | all] *ip*...

`blacklist` input | them | him | her | it | this | these *ip*...

Description

The **blacklist** helper command creates a blacklist for the *ip* list given (which can be in quotes or not).

If the type `full` or one of its aliases is supplied, or no type is given, a bidirectional stateless blacklist will be generated. The firewall will REJECT all traffic going to the IP addresses and DROP all traffic coming from them.

If the type `input` or one of its aliases is supplied, a unidirectional stateful blacklist will be generated. Connections can be initiated to such IP addresses, but the IP addresses will not be able to connect to the firewall or hosts protected by it.

Any blacklists will affect all router and interface definitions. They must be declared before the first router or interface.

Examples

```
blacklist full 192.0.2.1 192.0.2.2
blacklist input "192.0.2.3 192.0.2.4"
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)

3.14 classify config helper: sanewall-classify

Name

sanewall-classify — classify traffic for traffic shapping tools

Synopsis

`classify` *`class`* [*`rule-params`*]

Description

The **classify** helper command puts matching traffic into the specified traffic shaping class.

The *`class`* is a class as used by **iptables** and **tc** (e.g. MAJOR:MINOR).

The *`rule-params`* define a set of rule parameters to match the traffic that is to be classified. See optional rule parameters: sanewall-rule-params(5) for more details.

Any **classify** commands will affect all traffic matched. They must be declared before the first router or interface.

Examples

```
# Put all smtp traffic leaving via eth1 in class 1:1
classify 1:1 outface eth1 proto tcp dport 25
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
administration tool for IPv4 firewalls: iptables(8)
show / manipulate traffic control settings: tc(8)
Linux Advanced Routing & Traffic Control HOWTO

3.15 connmark config helper: sanewall-connmark

Name

sanewall-connmark — set a stateful mark on a connection

Synopsis

`connmark` *`value`* | save | restore *`chain`* [*`rule-params`*]

Description

The **connmark** helper command sets a mark on a whole connection. It applies to both directions.

Note
To set a mark on packets matching particular rules, regardless of any connection, see mark config helper: sanewall-mark(5).

The *`value`* is the mark value to set (a 32 bit integer). If you specify `save` then the mark on the matched packet will be turned into a connmark. If you specify `restore` then the matched packet will have its mark set to the current connmark.

The *`chain`* will be used to find traffic to mark. It can be any of the **iptables** built in chains belonging to the `mangle` table. The chain names are: INPUT, FORWARD, OUTPUT, PREROUTING and POSTROUTING. The names are case-sensitive.

The *`rule-params`* define a set of rule parameters to match the traffic that is to be marked within the chosen chain. See optional rule parameters: sanewall-rule-params(5) for more details.

Any **connmark** commands will affect all traffic matched. They must be declared before the first router or interface.

Examples

Consider a scenario with 3 ethernet ports, where eth0 is on the local LAN, eth1 connects to ISP 'A' and eth2 to ISP 'B'. To ensure traffic leaves via the same ISP as it arrives from you can mark the traffic:

```
# mark connections when they arrive from the ISPs
connmark 1 PREROUTING inface eth1
connmark 2 PREROUTING inface eth2
```

```
# restore the mark (from the connmark) when packets arrive from the LAN
connmark restore OUTPUT
connmark restore PREROUTING inface eth0
```

It is then possible to use the commands from iproute2 such as **ip**, to pick the correct routing table based on the mark on the packets.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
mark config helper: sanewall-mark(5)
administration tool for IPv4 firewalls: iptables(8)
show / manipulate routing, devices, policy routing and tunnels: ip(8)
Linux Advanced Routing & Traffic Control HOWTO

3.16 dscp config helper: sanewall-dscp

Name

sanewall-dscp — set the DSCP field in the packet header

Synopsis

`dscp` *`value`* | class *`classid chain`* [*`rule-params`*]

Description

The **dscp** helper command sets the DSCP field in the header of packets traffic, to allow QoS shaping.

Note
There is also a **dscp** parameter which allows matching DSCP values within individual rules (see optional rule parameters: sanewall-rule-params(5)).

Set *`value`* to a decimal or hexadecimal (0xnn) number to set an explicit DSCP value or use `class` *`classid`* to use an iptables DiffServ class, such as EF, BE, CSxx or AFxx (see **`iptables -j DSCP --help`** for more information).

The *`chain`* will be used to find traffic to mark. It can be any of the **iptables** built in chains belonging to the `mangle` table. The chain names are: INPUT, FORWARD, OUTPUT, PREROUTING and POSTROUTING. The names are case-sensitive.

The *`rule-params`* define a set of rule parameters to match the traffic that is to be marked within the chosen chain. See optional rule parameters: sanewall-rule-params(5) for more details.

Any **dscp** commands will affect all traffic matched. They must be declared before the first router or interface.

Examples

```
# set DSCP field to 32, packets sent by the local machine
dscp 32 OUTPUT

# set DSCP field to 32 (hex 20), packets routed by the local machine
dscp 0x20 FORWARD
```

```
# set DSCP to DiffServ class EF, packets routed by the local machine
#              and destined for port TCP/25 of 198.51.100.1
dscp class EF FORWARD proto tcp dport 25 dst 198.51.100.1
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
administration tool for IPv4 firewalls: iptables(8)
show / manipulate routing, devices, policy routing and tunnels: ip(8)
Linux Advanced Routing & Traffic Control HOWTO
optional rule parameters: sanewall-rule-params(5)

3.17 mac config helper: sanewall-mac

Name

sanewall-mac — ensure source IP and source MAC address match

Synopsis

`mac` *`IP macaddr`*

Description

Any **mac** commands will affect all traffic destined for the firewall host, or to be forwarded by the host. They must be declared before the first router or interface.

Note
There is also a **mac** parameter which allows matching MAC addresses within individual rules (see optional rule parameters: sanewall-rule-params(5)).

The **mac** helper command DROPs traffic from any *`IP`* address that was not sent using the *`macaddr`* specified.

When packets are dropped, a log is produced with the label "MAC MISSMATCH" (sic.). **mac** obeys the default log limits (see the section called “Logging” in optional rule parameters: sanewall-rule-params(5)).

Note
This command restricts an IP to a particular MAC address. The same MAC address is permitted send traffic with a different IP.

Examples

```
mac 192.0.2.1    00:01:01:00:00:e6
mac 198.51.100.1 00:01:01:02:aa:e8
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
optional rule parameters: sanewall-rule-params(5)

3.18 mark config helper: sanewall-mark

Name

sanewall-mark — mark traffic for traffic shaping tools

Synopsis

`mark` *`value chain`* [*`rule-params`*]

Description

The **mark** helper command sets a mark on packets that can be matched by traffic shaping tools for controlling the traffic.

Note
To set a mark on whole connections, see connmark config helper: sanewall-connmark(5). There is also a **mark** parameter which allows matching marks within individual rules (see optional rule parameters: sanewall-rule-params(5)).

The *`value`* is the mark value to set (a 32 bit integer).

The *`chain`* will be used to find traffic to mark. It can be any of the **iptables** built in chains belonging to the `mangle` table. The chain names are: INPUT, FORWARD, OUTPUT, PREROUTING and POSTROUTING. The names are case-sensitive.

The *`rule-params`* define a set of rule parameters to match the traffic that is to be marked within the chosen chain. See optional rule parameters: sanewall-rule-params(5) for more details.

Any **mark** commands will affect all traffic matched. They must be declared before the first router or interface.

Note
If you want to do policy based routing based on iptables marks, you will need to disable the Root Path Filtering on the interfaces involved (rp_filter in sysctl).

Examples

```
# mark with 1, packets sent by the local machine
mark 1 OUTPUT

# mark with 2, packets routed by the local machine
mark 2 FORWARD

# mark with 3, packets routed by the local machine, sent from
#              192.0.2.2 destined for port TCP/25 of 198.51.100.1
mark 3 FORWARD proto tcp dport 25 dst 198.51.100.1 src 192.0.2.2
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
connmark config helper: sanewall-connmark(5)
administration tool for IPv4 firewalls: iptables(8)
show / manipulate routing, devices, policy routing and tunnels: ip(8)
Linux Advanced Routing & Traffic Control HOWTO
optional rule parameters: sanewall-rule-params(5)

3.19 nat, snat, dnat, redirect config helpers: sanewall-nat

Name

sanewall-nat, sanewall-snat, sanewall-dnat, sanewall-redirect — set up NAT and port redirections

Synopsis

`snat` [to] *`target`* [*`rule-params`*]

`dnat` [to] *`target`* [*`rule-params`*]

`redirect` [to] *`portrange`* [*`rule-params`*]

`nat` to-source | to-destination | redirect-to *`target`* [*`rule-params`*]

`nat` redirect-to *`portrange`* [*`rule-params`*]

Description

Note
The *`rule-params`* are used only to determine the traffic that will be matched for NAT in these commands.

snat

The **snat** helper sets up a Source NAT rule for routed traffic by calling **nat to-source**. For example:

```
snat to 192.0.2.1 outface eth0 src 198.51.100.1 dst 203.0.113.1
```

dnat

The **dnat** helper sets up a Destination NAT rule for routed traffic by calling **nat to-destination**. For example:

```
dnat to 192.0.2.1 outface eth0 src 198.51.100.1 dst 203.0.113.1
```

redirect

The **redirect** helper redirects matching traffic to *portrange* on the local host by calling **nat redirect-to**. For example:

```
redirect-to 8080 inface eth0 src 198.51.100.0/24 proto tcp dport 80
```

nat

The **nat** helper takes one of the following sub-commands:

to-source *target*
: Defines a Source NAT (created in table NAT, chain POSTROUTING).

  target is the source address to be set in packets matching *rule-params*.

  If no rules are given, all forwarded traffic will be matched. inface should not be used in SNAT since the information is not available at the time the decision is made.

  target accepts any --to-source values that **iptables(8)** accepts. Run **iptables -j SNAT --help** to for more information. Multiple *targets* may be specified by separating with spaces and enclosing with quotes.

to-destination *target*
: Defines a Destination NAT (created in table NAT, chain POSTROUTING).

  target is the destination address to be set in packets matching *rule-params*.

  If no rules are given, all forwarded traffic will be matched. outface should not be used in DNAT since the information is not available at the time the decision is made.

  target accepts any --to-destination values that **iptables(8)** accepts. Run **iptables -j DNAT --help** to for more information. Multiple *targets* may be specified by separating with spaces and enclosing with quotes.

redirect-to *portrange*
: Redirect matching traffic to the local machine (created in table NAT, chain PREROUTING).

  portrange is the port range (from-to) or single port that packets matching *rule-params* will be redirected to.

  If no rules are given, all forwarded traffic will be matched. outface should not be used in REDIRECT since the information is not available at the time the decision is made.

Examples

```
# Send to 192.0.2.1
#  - all traffic arriving at or passing through the firewall
nat to-destination 192.0.2.1

# Send to 192.0.2.1
#  - all traffic arriving at or passing through the firewall
#  - which WAS going to 203.0.113.1
nat to-destination 192.0.2.1 dst 203.0.113.1

# Send to 192.0.2.1
#  - TCP traffic arriving at or passing through the firewall
#  - which WAS going to 203.0.113.1
nat to-destination 192.0.2.1 proto tcp dst 203.0.113.1

# Send to 192.0.2.1
#  - TCP traffic arriving at or passing through the firewall
#  - which WAS going to 203.0.113.1, port 25
nat to-destination 192.0.2.1 proto tcp dport 25 dst 203.0.113.1

# Other examples
nat to-source 192.0.2.1 outface eth0 src 198.51.100.1 dst 203.0.113.1
nat to-destination 192.0.2.2 outface eth0 src 198.51.100.2 dst 203.0.113.2
nat redirect-to 8080 inface eth0 src 198.51.100.0/24 proto tcp dport 80
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
optional rule parameters: sanewall-rule-params(5)
masquerade helper: sanewall-masquerade(5)

3.20 transparent_proxy, transparent_squid helpers: sanewall-transparent_proxy

Name

sanewall-transparent_proxy, sanewall-transparent_squid — set up a transparent proxy

Synopsis

`transparent_proxy` *`service port user`* [*`rule-params`*]

`transparent_squid` *`port user`* [*`rule-params`*]

Description

The **transparent_proxy** helper command sets up transparent caching for TCP traffic.

Note

The proxy application must be running on the firewall host at port *`port`* with the credentials of the local user *`user`* (which may be a space-delimited list enclosed in quotes) serving requests appropriate to the TCP port *`service`*.

The *`rule-params`* define a set of rule parameters to define the traffic that is to be proxied. See optional rule parameters: sanewall-rule-params(5) for more details.

For traffic destined for the firewall host or passing through the firewall, do not use the `outface` rule because the rules are applied before the routing decision and so the outgoing interface will not be known.

An empty *`user`* string ("") disables caching of locally-generated traffic. Otherwise, traffic starting from the firewall is captured, except traffic generated by the local user(s) *`user`*. The `inface`, `outface` and `src` *`rule-params`* are all ignored for locally-generated traffic.

The **transparent_squid** helper command sets up the special case for HTTP traffic with *`service`* implicitly set to 80.

Examples

```
transparent_proxy 80 3128 squid inface eth0 src 192.0.2.0/24
transparent_squid 3128 squid inface eth0 src 192.0.2.0/24

transparent_proxy "80 3128 8080" 3128 "squid privoxy root bin" \
  inface not "ppp+ ipsec+" dst not "a.not.proxied.server"
transparent_squid "80 3128 8080" "squid privoxy root bin" \
  inface not "ppp+ ipsec+" dst not "non.proxied.server"
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)

3.21 tos config helper: sanewall-tos

Name

sanewall-tos — set the Type of Service (TOS) of packets

Synopsis

`tos` *`value chain`* [*`rule-params`*]

Description

The **tos** helper command sets the Type of Service (TOS) field in packet headers.

Note
There is also a **tos** parameter which allows matching TOS values within individual rules (see optional rule parameters: sanewall-rule-params(5)).

The *`value`* can be an integer number (decimal or hexadecimal) or one of the descriptive values accepted by **iptables** (run **`iptables -j TOS --help`** for a list).

The *`chain`* will be used to find traffic to mark. It can be any of the **iptables** built in chains belonging to the `mangle` table. The chain names are: INPUT, FORWARD, OUTPUT, PREROUTING and POSTROUTING. The names are case-sensitive.

The *`rule-params`* define a set of rule parameters to match the traffic that is to be marked within the chosen chain. See optional rule parameters: sanewall-rule-params(5) for more details.

Any **tos** commands will affect all traffic matched. They must be declared before the first router or interface.

Examples

```
# set TOS to 16, packets sent by the local machine
tos 16 OUTPUT

# set TOS to 0x10 (16), packets routed by the local machine
tos 0x10 FORWARD

# set TOS to Maximize-Throughput (8), packets routed by the local
#              machine, destined for port TCP/25 of 198.51.100.1
tos Maximize-Throughput FORWARD proto tcp dport 25 dst 198.51.100.1
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
tosfix config helper: sanewall-tosfix(5)
administration tool for IPv4 firewalls: iptables(8)
optional rule parameters: sanewall-rule-params(5)

3.22 tosfix config helper: sanewall-tosfix

Name

sanewall-tosfix — apply suggested TOS values to packets

Synopsis

```
tosfix
```

Description

The **tosfix** helper command sets the Type of Service (TOS) field in packet headers based on the suggestions given by Erik Hensema in iptables and tc shapping tricks .

The following TOS values are set:

- All TCP ACK packets with length less than 128 bytes are assigned Minimize-Delay, while bigger ones are assigned Maximize-Throughput
- All packets with TOS Minimize-Delay, that are bigger than 512 bytes are set to Maximize-Throughput, except for short bursts of 2 packets per second

The **tosfix** command must be used before the first router or interface.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
tos config helper: sanewall-tos(5)
administration tool for IPv4 firewalls: iptables(8)

3.23 iptables helper: sanewall-iptables

Name

sanewall-iptables — include custom iptables commands

Synopsis

`iptables` *argument*...

Description

The **iptables** helper command passes all of its arguments to the real **iptables(8)** at the appropriate point during run-time.

Note
When used in an **interface** or **router**, the result will not have a direct relationship to the enclosing definition as the parameters passed are only those you supply.

You should not use **/sbin/iptables** directly in a Sanewall configuration as it will run before Sanewall activates its firewall. This means they it be applied to the running firewall, not the new firewall, so will be removed when the new firewall is activated.

The **iptables** helper is provided to allow you to hook in commands safely.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
administration tool for IPv4 firewalls: iptables(8)

3.24 masquerade helper: sanewall-masquerade

Name

sanewall-masquerade — set up masquerading (NAT) on an interface

Synopsis

`masquerade` *`real-interface`* [*`rule-params`*]

`masquerade` [reverse] [*`rule-params`*]

Description

The **masquerade** helper command sets up masquerading on the output of a real network interface (as opposed to a Sanewall interface definition).

If a *`real-interface`* is specified the command should be used before any interface or router definitions. Multiple values can be given separated by whitespace, so long as they are enclosed in quotes.

If used within an interface definition the definition's *`real-interface`* will be used.

If used within a router definition the definition's *`outface`*(s) will be used if specified. If the `reverse` option is gived, then the definition's *`inface`*(s) will be used if specified.

Unlike most commands, **masquerade** does not inherit its parent definition's *`rules-params`*, it only honour's its own. The `inface` and `outface` parameters should not be used (iptables does not support inface in the POSTROUTING chain and outface will be overwritten by Sanewall using the rules above).

Note

The masquerade always applies to the output of the chosen network interfaces.
`SANEWALL_NAT` will be turned on automatically (see control variables: sanewall-variables(5)) and Sanewall will enable packet-forwarding in the kernel.

Masquerading and SNAT

Masquerading is a special form of Source NAT (SNAT) that changes the source of requests when they go out and replaces their original source when they come in. This way a Linux host can become an Internet router for a LAN of clients having unroutable IP addresses. Masquerading takes care to re-map IP addresses and ports as required.

Masquerading is expensive compare to SNAT because it checks the IP address of the outgoing interface every time for every packet. If your host has a static IP address you should generally prefer SNAT.

Examples

```
# Before any interface or router
masquerade eth0 src 192.0.2.0/24 dst not 192.0.2.0/24

# In an interface definition to masquerade the output of its real- ↩
    interface
masquerade

# In a router definition to masquerade the output of its outface
masquerade

# In a router definition to masquerade the output of its inface
masquerade reverse
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
optional rule parameters: sanewall-rule-params(5)
nat, snat, dnat, redirect config helpers: sanewall-nat(5)

3.25 tcpmss helper: sanewall-tcpmss

Name

sanewall-tcpmss — set the MSS of TCP SYN packets for routers

Synopsis

`tcpmss` *`mss`* | auto

Description

The **tcpmss** helper command sets the MSS (Maximum Segment Size) of TCP SYN packets routed through the firewall. This can be used to overcome situations where Path MTU Discovery is not working and packet fragmentation is not possible.

A numeric *`mss`* will set MSS of TCP connections to the value given. Using the word `auto` will set the MSS to the MTU of the outgoing interface minus 40 (clamp-mss-to-pmtu).

If used within a router definition the MSS will be applied on the *`outface`*(s) of the router. If used before any router or interface definitions it will be applied to all traffic passing through the firewall.

Note
The **tcpmss** command cannot be used in an interface.

Examples

```
tcpmss auto

tcpmss 500
```

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
router definition: sanewall-router(5)
TCPMSS target in the iptables tutorial

3.26 optional rule parameters: sanewall-rule-params

Name

sanewall-rule-params, sanewall-src, sanewall-dst, sanewall-srctype, sanewall-dsttype, sanewall-inface, sanewall-outface, sanewall-physin, sanewall-physout, sanewall-custom, sanewall-log, sanewall-loglimit, sanewall-proto, sanewall-uid, sanewall-gid, sanewall-mac-param, sanewall-mark-param, sanewall-tos-param, sanewall-dscp-param — optional rule parameters

Synopsis

Common

`src` [not] *`host`*

`dst` [not] *`host`*

`srctype` [not] *`type`*

`dsttype` [not] *`type`*

`proto` [not] *`protocol`*

`mac` [not] *`macaddr`*

`dscp` [not] *`value`* | class *`classid`*

`mark` [not] *`id`*

`tos` [not] *`id`*

`custom` "*`iptables-options...`*"

Router Only

`inface` [not] *`interface`*

`outface` [not] *`interface`*

`physin` [not] *`interface`*

`physout` [not] *`interface`*

Interface Only

`uid` [not] *`user`*

`gid` [not] *`group`*

Logging

`log` "*`log text`*" [level *`loglevel`*]

`loglimit` "*`log text`*" [level *`loglevel`*]

Description

Optional rule parameters are accepted by many commands to narrow the match they make. Not all parameters are accepted by all commands so you should check the individual commands for exclusions.

All matches are made against the REQUEST. Sanewall automatically sets up the necessary stateful rules to deal with replies in the reverse direction.

Use the option `not` to match any value other than the one(s) specified.

The logging parameters are unusual in that they do not affect the match, they just cause a log message to be emitted. Therefore, the logging parameters don't support the `not` option.

Sanewall is designed so that if you specify a parameter that is also used internally by the command then a warning will be issued (and the internal version will be used).

Common

Use **src** and **dst** to define the source and destination IP addresses of the request respectively. *`host`* defines the IP or IPs to be matched. Examples:

```
server smtp accept src not 192.0.2.1
server smtp accept dst 198.51.100.1
server smtp accept src not 192.0.2.1 dst 198.51.100.1
```

Use **srctype** or **dsttype** to define the source or destination IP address type of the request. *`type`* is the address type category as used in the kernel's network stack. It can be one of:

UNSPEC
: an unspecified address (i.e. 0.0.0.0)

UNICAST
: a unicast address

LOCAL
: a local address

BROADCAST
: a broadcast address

ANYCAST
: an anycast address

MULTICAST
: a multicast address

BLACKHOLE
: a blackhole address

UNREACHABLE
an unreachable address

PROHIBIT
a prohibited address

THROW, NAT, XRESOLVE
undocumented

See **iptables(8)** or run **`iptables -m addrtype --help`** for more information. Examples:

```
server smtp accept srctype not "UNREACHABLE PROHIBIT"
```

Use **proto** to match by protocol. The *`protocol`* can be any accepted by iptables(8).

Use **mac** to match by MAC address. The *`macaddr`* matches to the "remote" host. In an **interface**, "remote" always means the non-local host. In a **router**, "remote" refers to the source of requests for `servers`. It refers to the destination of requests for `clients`. Examples:

```
# Only allow pop3 requests to the e6 host
client pop3 accept mac 00:01:01:00:00:e6

# Only allow hosts other than e7/e8 to access smtp
server smtp accept mac not "00:01:01:00:00:e7 00:01:01:00:00:e8"
```

Use **dscp** to match the DSCP field on packets. For details on DSCP *`value`*s and *`classid`*s, see dscp config helper: sanewall-dscp(5).

```
server smtp accept dscp not "0x20 0x30"
server smtp accept dscp not class "BE EF"
```

Use **mark** to match marks set on packets. For details on mark *`id`*s, see mark config helper: sanewall-mark(5).

```
server smtp accept mark not "20 55"
```

Use **tos** to match the TOS field on packets. For details on TOS *`id`*s, see tos config helper: sanewall-tos(5).

```
server smtp accept tos not "Maximize-Throughput 0x10"
```

Use **custom** to pass arguments directly to iptables(8). All of the parameters must be in a single quoted string. To pass an option to iptables(8) that itself contains a space you need to quote strings in the usual bash(1) manner. For example:

```
server smtp accept custom "--some-option some-value"
server smtp accept custom "--some-option 'some-value second-value'"
```

Router Only

Use **inface** and **outface** to define the *interface* via which a request is received and forwarded respectively. Use the same format as interface definition: sanewall-interface(5). Examples:

```
server smtp accept inface not eth0
server smtp accept inface not "eth0 eth1"
server smtp accept inface eth0 outface eth1
```

Use **physin** and **physout** to define the physical *interface* via which a request is received or send in cases where the inface or outface is known to be a virtual interface; e.g. a bridge. Use the same format as interface definition: sanewall-interface(5). Examples:

```
server smtp accept physin not eth0
```

Interface only

These parameters match information related to information gathered from the local host. They are silently ignored for incoming requests or requests that will be forwarded.

Use **uid** to match the operating system user sending the traffic. The *user* is a username, uid number or a quoted list of the two.

For example, to limit which users can access POP3 and IMAP:

```
client "pop3 imap" accept user not "user1 user2 user3"
```

This will allow all requests to reach the server but prevent replies unless the web server is running as apache:

```
server http accept user apache
```

Use **gid** to match the operating system group sending the traffic. The *group* is a group name, gid number or a quoted list of the two.

Note
The Linux kernel infrastructure to match PID/SID and executable names with **pid**, **sid** and **cmd** has been removed so these options can no longer be used.

Logging

Use **log** or **loglimit** to log matching packets to syslog. Unlike iptables(8) logging, this is not an action: Sanewall will produce multiple iptables commands to accomplish both the action for the rule and the logging.

Logging is controlled using the `SANEWALL_LOG_OPTIONS` and `SANEWALL_LOG_LEVEL` environment variables (see control variables: sanewall-variables(5)). **loglimit** additionally honours the `SANEWALL_LOG_FREQUENCY` and `SANEWALL_LOG_BURST` variables.

Specifying `level` (which takes the same values as `SANEWALL_LOG_LEVEL`) allows you to override the log level for a single rule.

Internal use

In addition to the commands in the synopsis, Sanewall provides **limit**, **sport** and **dport** which are used internally. These should not normally be used in configuration files unless you really understand what you are doing.

limit requires the arguments *`frequency`* and *`burst`* and will limit the matching of traffic in both directions.

sport requires an argument *`port`* which can be a name, number, range (FROM:TO) or a quoted list of ports. It specifies the source port of a request.

dport requires an argument *`port`* which can be a name, number, range (FROM:TO) or a quoted list of ports. It specifies the destination port of a request.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
client command: sanewall-client(5)
server, route commands: sanewall-server(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
mark config helper: sanewall-mark(5)
tos config helper: sanewall-tos(5)
dscp config helper: sanewall-dscp(5)
control variables: sanewall-variables(5)
administration tool for IPv4 firewalls: iptables(8)

3.27 actions for rules: sanewall-actions

Name

sanewall-actions, sanewall-accept, sanewall-deny, sanewall-drop, sanewall-reject, sanewall-return, sanewall-tarpit — rule actions

Synopsis

`accept`

`accept with limit` *`requests/period burst`* [overflow *`action`*]

`accept with recent` *`name seconds hits`*

`accept with knock` *`name`*

`reject` [with *`message`*]

`drop`

`deny`

`return`

`tarpit`

Description

These actions are the actions to be taken on traffic that has been matched by a particular rule.

Sanewall will also pass through any actions that **iptables(8)** accepts, however these definitions provide lowercase versions which accept arguments where appropriate and which could otherwise not be passed through.

Note
The **iptables(8)** LOG action is best used through the optional rule parameter `log` since the latter can be combined with one of these actions (Sanewall will generate multiple firewall rules to make this happen). For information on `log` and `loglimit`, see optional rule parameters: sanewall-rule-params(5).

The following actions are defined:

accept

accept allows the traffic matching the rules to reach its destination.

For example, to allow SMTP requests and their replies to flow:

```
server smtp accept
```

accept with limit

accept with limit allows the traffic, with new connections limited to `requests/period` with a maximum `burst`. Run **`iptables -m limit --help`** for more information.

The default `overflow` action is to REJECT the excess connections (DROP would produce time-outs on otherwise valid service clients).

Examples:

```
server smtp accept with limit 10/sec 100

server smtp accept with limit 10/sec 100 overflow drop
```

accept with recent

accept with recent allows the traffic matching the rules to reach its destination, limited per remote IP to `hits` per `seconds`. Run **`iptables -m recent --help`** for more information.

The `name` parameter is used to allow multiple rules to share the same table of recent IPs.

For example, to allow only 2 connections every 60 seconds per remote IP, to the smtp server:

```
server smtp accept with recent mail 60 2
```

Note

When a new connection is not allowed, the traffic will continue to be matched by the rest of the firewall. In other words, if the traffic is not allowed due to the limitations set here, it is not dropped, it is just not matched by this rule.

---

accept with knock

accept with knock allows easy integration with knockd, a server that allows you to control access to services by sending certain packets to "knock" on the door, before the door is opened for service.

The `name` is used to build a special chain knock_<`name`> which contains rules to allow established connections to work. If knockd has not allowed new connections any traffic entering this chain will just return back and continue to match against the other rules until the end of the firewall.

For example, to allow HTTPS requests based on a knock write:

```
server https accept with knock hidden
```

then configure knockd to enable the HTTPS service with:

```
iptables -A knock_hidden -s %IP% -j ACCEPT
```

and disable it with:

```
iptables -D knock_hidden -s %IP% -j ACCEPT
```

You can use the same knock `name` in more than one Sanewall rule to enable/disable all the services based on a single knockd configuration entry.

Note
There is no need to match anything other than the IP in knockd. Sanewall already matches everything else needed for its rules to work.

---

reject with *`message`*, reject
reject discards the traffic matching the rules and sends a rejecting message back to the sender.

When used with `with` the specific message to return can be specified. Run **`iptables -j REJECT --help`** for a list of the `--reject-with` values which can be used for `message`. See the section called "Reject With Messsages" for some examples.

The default (no `message` specified) is to send `tcp-reset` when dealing with TCP connections and `icmp-port-unreachable` for all other protocols.

For example:

```
UNMATCHED_INPUT_POLICY="reject with host-prohib"

policy reject with host-unreach

server ident reject with tcp-reset
```

drop, deny
drop discards the traffic matching the rules. It does so silently and the sender will need to timeout to conclude it cannot reach the service.

deny is a synonym for **drop**. For example, either of these would silently discard SMTP traffic:

```
server smtp drop

server smtp deny
```

return
return will return the flow of processing to the parent of the current command.

Currently, the only time **return** can be used meaningfully used is as a policy for an interface definition. Unmatched traffic will continue being processed with the possibility of being matched by a later definition. For example:

```
policy return
```

tarpit

tarpit captures and holds incoming TCP connections open.

Connections are accepted and immediately switched to the persist state (0 byte window), in which the remote side stops sending data and asks to continue every 60-240 seconds.

Attempts to close the connection are ignored, forcing the remote side to time out the connection after 12-24 minutes.

Example:

```
server smtp tarpit
```

Note
As the kernel conntrack modules are always loaded by Sanewall, some per-connection resources will be consumed. See this bug report for details.

The following actions also exist but should not be used under normal circumstances:

mirror

mirror returns the traffic it receives by switching the source and destination fields. REJECT will be used for traffic generated by the local host.

Warning
The MIRROR target was removed from the Linux kernel due to its security implications. MIRROR is dangerous; use it with care and only if you understand what you are doing.

redirect, redirect to-port *port*

redirect is used internally by Sanewall helper commands.

Only Sanewall developers should need to use this action directly.

Reject With Messsages

The following RFCs contain information relevant to these messages:

RFC 1812
RFC 1122
RFC 792

icmp-net-unreachable, net-unreach
ICMP network unreachable

Generated by a router if a forwarding path (route) to the destination network is not available.

From RFC 1812, section 5.2.7.1. See RFC 1812 and RFC 792.

Note
Use with care. The sender and the routers between you and the sender may conclude that the whole network your host resides in is unreachable, and prevent other traffic from reaching you.

---

icmp-host-unreachable, host-unreach
ICMP host unreachable

Generated by a router if a forwarding path (route) to the destination host on a directly connected network is not available (does not respond to ARP).

From RFC 1812, section 5.2.7.1. See RFC 1812 and RFC 792.

Note
Use with care. The sender and the routers between you and the sender may conclude that your host is entirely unreachable, and prevent other traffic from reaching you.

---

icmp-proto-unreachable, proto-unreach
ICMP protocol unreachable

Generated if the transport protocol designated in a datagram is not supported in the transport layer of the final destination.

From RFC 1812, section 5.2.7.1. See RFC 1812 and RFC 792.

icmp-port-unreachable, port-unreach
ICMP port unreachable

Generated if the designated transport protocol (e.g. TCP, UDP, etc.) is unable to demultiplex the datagram in the transport layer of the final destination but has no protocol mechanism to inform the sender.

From RFC 1812, section 5.2.7.1. See RFC 1812 and RFC 792.

Generated by hosts to indicate that the required port is not active.

icmp-net-prohibited, net-prohib
ICMP communication with destination network administratively prohibited

This code was intended for use by end-to-end encryption devices used by U.S. military agencies. Routers SHOULD use the newly defined Code 13 (Communication Administratively Prohibited) if they administratively filter packets.

From RFC 1812, section 5.2.7.1. See RFC 1812 and RFC 1122.

Note
This message may not be widely understood.

icmp-host-prohibited, host-prohib
ICMP communication with destination host administratively prohibited

This code was intended for use by end-to-end encryption devices used by U.S. military agencies. Routers SHOULD use the newly defined Code 13 (Communication Administratively Prohibited) if they administratively filter packets.

From RFC 1812, section 5.2.7.1. See RFC 1812 and RFC 1122.

Note
This message may not be widely understood.

tcp-reset
TCP RST

The port unreachable message of the TCP stack.

See RFC 1122.

Note
`tcp-reset` is useful when you want to prevent timeouts on rejected TCP services where the client incorrectly ignores ICMP port unreachable messages.

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)
interface definition: sanewall-interface(5)
router definition: sanewall-router(5)
optional rule parameters: sanewall-rule-params(5)

3.28 services list: sanewall-services

Name

sanewall-services — Sanewall service list

Services

This Wikipedia list of ports may be helpful if you need to define a new service.

AH - IPSEC AUTHENTICATION HEADER (AH) - SIMPLE SERVICE

Example
Configuration sample:

```
server AH accept
```

Server Ports
51/any

Client Ports
any

Links
Wikipedia

Notes
For more information see this Archive of the FreeS/WAN documentation and RFC 2402.

ALL - MATCH ALL TRAFFIC - COMPLEX SERVICE

Example
Configuration sample:

```
server all accept
```

Server Ports
all

Client Ports
all

Notes

Matches all traffic (all protocols, ports, etc) while ensuring that required kernel modules are loaded.

This service may indirectly setup a set of other services, if they require kernel modules to be loaded. The following complex services are activated:

ftp - File Transfer Protocol - simple service
irc - Internet Relay Chat - simple service

AMANDA - ADVANCED MARYLAND AUTOMATIC NETWORK DISK ARCHIVER - SIMPLE SERVICE

Server Ports

udp/10080

Client Ports

default

Netfilter Modules

nf_conntrack_amanda (CONFIG_NF_CONNTRACK_AMANDA)

Netfilter NAT Modules

nf_nat_amanda (CONFIG_NF_NAT_AMANDA)

Links

Homepage, Wikipedia

ANY - MATCH ALL TRAFFIC (WITHOUT MODULES OR INDIRECT) - COMPLEX SERVICE

Example

Configuration sample:

```
server any myname accept proto 47
```

Server Ports

all

Client Ports

all

Notes

Matches all traffic (all protocols, ports, etc), but does not care about kernel modules and does not activate any other service indirectly. In combination with the optional rule parameters: sanewall-rule-params(5) this service can match unusual traffic (e.g. GRE - protocol 47).

ANYSTATELESS - MATCH ALL TRAFFIC STATELESSLY - COMPLEX SERVICE

Example

Configuration sample:

```
server anystateless myname accept proto 47
```

Server Ports

all

Client Ports

all

Notes

Matches all traffic (all protocols, ports, etc), but does not care about kernel modules and does not activate any other service indirectly. In combination with the optional rule parameters: sanewall-rule-params(5) this service can match unusual traffic (e.g. GRE - protocol 47).

This service is identical to "any" but does not care about the state of traffic.

APCUPSD - APC UPS DAEMON - SIMPLE SERVICE

Example

Configuration sample:

```
server apcupsd accept
```

Server Ports

tcp/6544

Client Ports

default

Links

Homepage, Wikipedia

Notes

This service must be defined as "server apcupsd accept" on all machines not directly connected to the UPS (i.e. slaves).

Note that the port defined here is not the default port (6666) used if you download and compile APCUPSD, since the default conflicts with IRC and many distributions (like Debian) have changed this to 6544.

You can define port 6544 in APCUPSD, by changing the value of NETPORT in its configuration file, or overwrite this Sanewall service definition using the procedures described in the section called “Adding Services” of Sanewall configuration: sanewall.conf(5).

APCUPSDNIS - APC UPS DAEMON NETWORK INFORMATION SERVER - SIMPLE SERVICE

Example
Configuration sample:

```
server apcupsdnis accept
```

Server Ports
tcp/3551

Client Ports
default

Links
Homepage, Wikipedia

Notes
This service allows the remote WEB interfaces of APCUPSD, to connect and get information from the server directly connected to the UPS device.

APTPROXY - ADVANCED PACKAGING TOOL PROXY - SIMPLE SERVICE

Example
Configuration sample:

```
server aptproxy accept
```

Server Ports
tcp/9999

Client Ports
default

Links
Wikipedia

ASTERISK - ASTERISK PABX - SIMPLE SERVICE

Example
Configuration sample:

```
server asterisk accept
```

Server Ports
tcp/5038

Client Ports
default

Links
Homepage, Wikipedia

Notes
This service refers only to the manager interface of asterisk. You should normally enable sip - Session Initiation Protocol - simple service , h323 - H.323 VoIP - simple service , rtp - Real-time Transport Protocol - simple service , etc. at the firewall level, if you enable the relative channel drivers of asterisk.

CUPS - COMMON UNIX PRINTING SYSTEM - SIMPLE SERVICE

Example
Configuration sample:

```
server cups accept
```

Server Ports
tcp/631 udp/631

Client Ports
any

Links
Homepage, Wikipedia

CUSTOM - CUSTOM DEFINITIONS - CUSTOM SERVICE

Example
Configuration sample:

```
server custom myimap tcp/143 default accept
```

Server Ports
N/A

Client Ports
N/A

Notes
The full syntax is:

subcommand custom *name svr-proto/ports cli-ports* **action params**

This service is used by Sanewall to allow you create rules for services which do not have a definition.

subcommand, **action** and **params** have their usual meanings.

A `name` must be supplied along with server ports in the form *`proto/range`* and client ports which takes only a *`range`*.

To define services with the built-in extension mechanism to avoid the need for **custom** services, see the section called "Adding Services" of Sanewall configuration: sanewall.conf(5).

CVSPSERVER - CONCURRENT VERSIONS SYSTEM - SIMPLE SERVICE

Example

Configuration sample:

```
server cvspserver accept
```

Server Ports

tcp/2401

Client Ports

default

Links

Homepage, Wikipedia

DARKSTAT - DARKSTAT NETWORK TRAFFIC ANALYSER - SIMPLE SERVICE

Example

Configuration sample:

```
server darkstat accept
```

Server Ports

tcp/666

Client Ports

default

Links

Homepage

DAYTIME - DAYTIME PROTOCOL - SIMPLE SERVICE

Example

Configuration sample:

```
server daytime accept
```

Server Ports
tcp/13

Client Ports
default

Links
Wikipedia

DCC - DISTRIBUTED CHECKSUM CLEARINGHOUSE - SIMPLE SERVICE

Example
Configuration sample:

```
server dcc accept
```

Server Ports
udp/6277

Client Ports
default

Links
Wikipedia

Notes
See also this DCC FAQ.

DCPP - DIRECT CONNECT++ P2P - SIMPLE SERVICE

Example
Configuration sample:

```
server dcpp accept
```

Server Ports
tcp/1412 udp/1412

Client Ports
default

Links
Homepage

DHCP - DYNAMIC HOST CONFIGURATION PROTOCOL - COMPLEX SERVICE

Example

Configuration sample:

```
server dhcp accept
```

Server Ports

udp/67

Client Ports

68

Links

Wikipedia

Notes

The dhcp service is implemented as stateless rules.

DHCP clients broadcast to the network (src 0.0.0.0 dst 255.255.255.255) to find a DHCP server. If the DHCP service was stateful the iptables connection tracker would not match the packets and deny to send the reply.

Note that this change does not affect the security of either DHCP servers or clients, since only the specific ports are allowed (there is no random port at either the server or the client side).

Note also that the "server dhcp accept" or "client dhcp accept" commands should placed within interfaces that do not have src and / or dst defined (because of the initial broadcast).

You can overcome this problem by placing the DHCP service on a separate interface, without a src or dst but with a policy return. Place this interface before the one that defines the rest of the services.

For example:

```
          interface eth0 dhcp
          ~~~~policy return
          ~~~~server dhcp accept
          ~
          interface eth0 lan src "$mylan" dst "$myip"
          ~~~~client all accept
```

DHCPRELAY - DHCP RELAY - SIMPLE SERVICE

Example

Configuration sample:

```
server dhcprelay accept
```

Server Ports

udp/67

Client Ports
67

Links
Wikipedia

Notes
From RFC 1812 section 9.1.2:

In many cases, BOOTP clients and their associated BOOTP server(s) do not reside on the same IP (sub)network. In such cases, a third-party agent is required to transfer BOOTP messages between clients and servers. Such an agent was originally referred to as a BOOTP forwarding agent. However, to avoid confusion with the IP forwarding function of a router, the name BOOTP relay agent has been adopted instead.

For more information about DHCP Relay see section 9.1.2 of RFC 1812 and section 4 of RFC 1542

DICT - DICTIONARY SERVER PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server dict accept
```

Server Ports
tcp/2628

Client Ports
default

Links
Wikipedia

Notes
See RFC2229.

DISTCC - DISTRIBUTED CC - SIMPLE SERVICE

Example
Configuration sample:

```
server distcc accept
```

Server Ports
tcp/3632

Client Ports
default

Links
Homepage, Wikipedia

Notes
For distcc security, please check the distcc security design.

DNS - DOMAIN NAME SYSTEM - SIMPLE SERVICE

Example
Configuration sample:

```
server dns accept
```

Server Ports
udp/53 tcp/53

Client Ports
any

Links
Wikipedia

Notes
On very busy DNS servers you may see a few dropped DNS packets in your logs. This is normal. The iptables connection tracker will timeout the session and lose unmatched DNS packets that arrive too late to be useful.

ECHO - ECHO PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server echo accept
```

Server Ports
tcp/7

Client Ports
default

Links
Wikipedia

EMULE - EMULE (DONKEY NETWORK CLIENT) - COMPLEX SERVICE

Example

Configuration sample:

```
client emule accept src 192.0.2.1
```

Server Ports

many

Client Ports

many

Links

Homepage

Notes

According to eMule Port Definitions, Sanewall defines:

Accept from any client port to the server at tcp/4661
Accept from any client port to the server at tcp/4662
Accept from any client port to the server at udp/4665
Accept from any client port to the server at udp/4672
Accept from any server port to the client at tcp/4662
Accept from any server port to the client at udp/4672

Use the Sanewall client command: sanewall-client(5) command to match the eMule client.

Please note that the eMule client is an HTTP client also.

ESERVER - EDONKEY NETWORK SERVER - SIMPLE SERVICE

Example

Configuration sample:

```
server eserver accept
```

Server Ports

tcp/4661 udp/4661 udp/4665

Client Ports

any

Links

Wikipedia

ESP - IPSEC ENCAPSULATED SECURITY PAYLOAD (ESP) - SIMPLE SERVICE

Example

Configuration sample:

```
server ESP accept
```

Server Ports
50/any

Client Ports
any

Links
Wikipedia

Notes
For more information see this Archive of the FreeS/WAN documentation RFC 2406.

FINGER - FINGER PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server finger accept
```

Server Ports
tcp/79

Client Ports
default

Links
Wikipedia

FTP - FILE TRANSFER PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server ftp accept
```

Server Ports
tcp/21

Client Ports
default

Netfilter Modules
nf_conntrack_ftp (CONFIG_NF_CONNTRACK_FTP)

Netfilter NAT Modules
nf_nat_ftp (CONFIG_NF_NAT_FTP)

Links
Wikipedia

Notes
The FTP service matches both active and passive FTP connections.

GIFT - GIFT INTERNET FILE TRANSFER - SIMPLE SERVICE

Example
Configuration sample:

```
server gift accept
```

Server Ports
tcp/4302 tcp/1214 tcp/2182 tcp/2472

Client Ports
any

Links
Homepage, Wikipedia

Notes
The gift Sanewall service supports:

Gnutella listening at tcp/4302
FastTrack listening at tcp/1214
OpenFT listening at tcp/2182 and tcp/2472

The above ports are the defaults given for the corresponding giFT modules.

To allow access to the user interface ports of giFT, use the giftui - giFT Internet File Transfer User Interface - simple service .

GIFTUI - GIFT INTERNET FILE TRANSFER USER INTERFACE - SIMPLE SERVICE

Example
Configuration sample:

```
server giftui accept
```

Server Ports
tcp/1213

Client Ports
default

Links
Homepage, Wikipedia

Notes
This service refers only to the user interface ports offered by giFT. To allow gift accept P2P requests, use the gift - giFT Internet File Transfer - simple service .

GKRELLMD - GKRELLM DAEMON - SIMPLE SERVICE

Example
Configuration sample:

```
server gkrellmd accept
```

Server Ports
tcp/19150

Client Ports
default

Links
Homepage, Wikipedia

GRE - GENERIC ROUTING ENCAPSULATION - SIMPLE SERVICE

Example
Configuration sample:

```
server GRE accept
```

Server Ports
47/any

Client Ports
any

Netfilter Modules
nf_conntrack_proto_gre (CONFIG_NF_CT_PROTO_GRE)

Netfilter NAT Modules
nf_nat_proto_gre (CONFIG_NF_NAT_PROTO_GRE)

Links
Wikipedia

Notes

Protocol No 47.

For more information see RFC RFC 2784.

H323 - H.323 VoIP - SIMPLE SERVICE

Example

Configuration sample:

```
server h323 accept
```

Server Ports

tcp/1720

Client Ports

default

Netfilter Modules

nf_conntrack_h323 (CONFIG_NF_CONNTRACK_H323)

Netfilter NAT Modules

nf_nat_h323 (CONFIG_NF_NAT_H323)

Links

Wikipedia

HEARTBEAT - HeartBeat - SIMPLE SERVICE

Example

Configuration sample:

```
server heartbeat accept
```

Server Ports

udp/690:699

Client Ports

default

Links

Homepage

Notes

This Sanewall service has been designed such a way that it will allow multiple heartbeat clusters on the same LAN.

HTTP - HYPERTEXT TRANSFER PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server http accept
```

Server Ports
tcp/80

Client Ports
default

Links
Wikipedia

HTTPALT - HTTP ALTERNATE PORT - SIMPLE SERVICE

Example
Configuration sample:

```
server httpalt accept
```

Server Ports
tcp/8080

Client Ports
default

Links
Wikipedia

Notes
This port is commonly used by web servers, web proxies and caches where the standard http - Hypertext Transfer Protocol - simple service port is not available or can or should not be used.

HTTPS - SECURE HYPERTEXT TRANSFER PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server https accept
```

Server Ports
tcp/443

Client Ports
default

Links
Wikipedia

HYLAFAX - HYLAFAX - COMPLEX SERVICE

Example
Configuration sample:

```
server hylafax accept
```

Server Ports
many

Client Ports
many

Links
Homepage, Wikipedia

Notes
This service allows incoming requests to server port tcp/4559 and outgoing from server port tcp/4558.

The correct operation of this service has not been verified.

USE THIS WITH CARE. A HYLAFAX CLIENT MAY OPEN ALL TCP UNPRIVILEGED PORTS TO ANYONE (from port tcp/4558).

IAX - INTER-ASTERISK EXCHANGE - SIMPLE SERVICE

Example
Configuration sample:

```
server iax accept
```

Server Ports
udp/5036

Client Ports
default

Links
Homepage, Wikipedia

Notes
This service refers to IAX version 1. There is also iax2 - Inter-Asterisk eXchange v2 - simple service .

IAX2 - INTER-ASTERISK EXCHANGE V2 - SIMPLE SERVICE

Example
Configuration sample:

```
server iax2 accept
```

Server Ports
udp/5469 udp/4569

Client Ports
default

Links
Homepage, Wikipedia

Notes
This service refers to IAX version 2. There is also iax - Inter-Asterisk eXchange - simple service .

ICMP - INTERNET CONTROL MESSAGE PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server icmp accept
```

Server Ports
icmp/any

Client Ports
any

Links
Wikipedia

ICMP - INTERNET CONTROL MESSAGE PROTOCOL - SIMPLE SERVICE

Alias
See icmp - Internet Control Message Protocol - simple service

ICMPV6 - INTERNET CONTROL MESSAGE PROTOCOL V6 - SIMPLE SERVICE

Alias
See icmpv6 - Internet Control Message Protocol v6 - simple service

ICMPV6 - INTERNET CONTROL MESSAGE PROTOCOL V6 - SIMPLE SERVICE

Example
Configuration sample:

```
server icmpv6 accept
```

Server Ports
icmpv6/any

Client Ports
any

Links
Wikipedia

ICP - INTERNET CACHE PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server icp accept
```

Server Ports
udp/3130

Client Ports
3130

Links
Wikipedia

IDENT - IDENTIFICATION PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server ident reject with tcp-reset
```

Server Ports
tcp/113

Client Ports
default

Links
Wikipedia

IMAP - INTERNET MESSAGE ACCESS PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server imap accept
```

Server Ports
tcp/143

Client Ports
default

Links
Wikipedia

IMAPS - SECURE INTERNET MESSAGE ACCESS PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server imaps accept
```

Server Ports
tcp/993

Client Ports
default

Links
Wikipedia

IPSECNATT - NAT TRAVERSAL AND IPSEC - SIMPLE SERVICE

Server Ports
udp/4500

Client Ports
any

Links

Wikipedia

IPV6ERROR - ICMPV6 ERROR HANDLING - COMPLEX SERVICE

Example

Configuration sample:

```
server ipv6error accept
```

Server Ports

N/A

Client Ports

N/A

Notes

A number of icmpv6 error types are not treated equally inbound and outbound.

The ipv6error rule wraps all of them in the following way:

allow incoming messages only for existing sessions
allow outgoing messages always

The following icmpv6 messages are handled:

destination-unreachable
packet-too-big
ttl-zero-during-transit
ttl-zero-during-reassembly
unknown-header-type
unknown-option

Sanewall sets up both rules the same way. In a router with inface being internal and outface being external the following will meet the recommendations of RFC 4890:

```
server ipv6error accept
```

Interfaces should always have:

```
server ipv6error accept
```

set.

Do not use:

```
client ipv6error accept
```

unless you are controlling traffic on an router interface where outface is the internal destination.

IPV6NEIGH - IPV6 NEIGHBOUR DISCOVERY - COMPLEX SERVICE

Example

Configuration sample:

```
client ipv6neigh accept
server ipv6neigh accept
```

Server Ports

N/A

Client Ports

N/A

Notes

ICMPv6 does a great deal of automatic configuration and discovery. There is no ARP and no need to manually configure routers. To enable this functionality the network neighbour and router solicitation/advertisement messages should be enabled on each interface.

These rules are stateless since advertisement can happen automatically as well as on solicitation.

Neighbour discovery (incoming) should always be enabled:

```
server ipv6neigh accept
```

Neighbour advertisement (outgoing) should always be enabled:

```
client ipv6neigh accept
```

The rules should not be used to pass packets across a firewall (e.g. in a router) unless it is a bridge.

IPV6ROUTER - IPV6 ROUTER DISCOVERY - COMPLEX SERVICE

Example

Configuration sample:

```
client ipv6router accept
```

Server Ports

N/A

Client Ports

N/A

Notes

ICMPv6 does a great deal of automatic configuration and discovery. There is no ARP and no need to manually configure routers. To enable this functionality the network neighbour and router solicitation/advertisement messages should be enabled on each interface.

These rules are stateless since advertisement can happen automatically as well as on solicitation.

Router discovery (incoming) should always be enabled:

```
client ipv6router accept
```

Router advertisement (outgoing) should be enabled on a host that routes:

```
server ipv6router accept
```

The rules should not be used to pass packets across a firewall (e.g. in a router) unless it is a bridge.

IRC - INTERNET RELAY CHAT - SIMPLE SERVICE

Example

Configuration sample:

```
server irc accept
```

Server Ports

tcp/6667

Client Ports

default

Netfilter Modules

nf_conntrack_irc (CONFIG_NF_CONNTRACK_IRC)

Netfilter NAT Modules

nf_nat_irc (CONFIG_NF_NAT_IRC)

Links

Wikipedia

ISAKMP - INTERNET SECURITY ASSOCIATION AND KEY MANAGEMENT PROTOCOL (IKE) - SIMPLE SERVICE

Example

Configuration sample:

```
server isakmp accept
```

Server Ports

udp/500

Client Ports

any

Links
Wikipedia

Notes
For more information see the Archive of the FreeS/WAN documentation

JABBER - EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server jabber accept
```

Server Ports
tcp/5222 tcp/5223

Client Ports
default

Links
Wikipedia

Notes
Allows clear and SSL client-to-server connections.

JABBERD - EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL (SERVER) - SIMPLE SERVICE

Example
Configuration sample:

```
server jabberd accept
```

Server Ports
tcp/5222 tcp/5223 tcp/5269

Client Ports
default

Links
Wikipedia

Notes
Allows clear and SSL client-to-server and server-to-server connections.

Use this service for a jabberd server. In all other cases, use the jabber - Extensible Messaging and Presence Protocol - simple service .

L2TP - LAYER 2 TUNNELING PROTOCOL - SIMPLE SERVICE

Server Ports
udp/1701

Client Ports
any

Links
Wikipedia

LDAP - LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server ldap accept
```

Server Ports
tcp/389

Client Ports
default

Links
Wikipedia

LDAPS - SECURE LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server ldaps accept
```

Server Ports
tcp/636

Client Ports
default

Links
Wikipedia

LPD - LINE PRINTER DAEMON PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server lpd accept
```

Server Ports
tcp/515

Client Ports
any

Links
Wikipedia

Notes
LPD is documented in RFC 1179.

Since many operating systems incorrectly use the non-default client ports for LPD access, this definition allows any client port to access the service (in addition to the RFC defined 721 to 731 inclusive).

MICROSOFT_DS - DIRECT HOSTED (NETBIOS-LESS) SMB - SIMPLE SERVICE

Example
Configuration sample:

```
server microsoft_ds accept
```

Server Ports
tcp/445

Client Ports
default

Notes
Direct Hosted (i.e. NETBIOS-less SMB)

This is another NETBIOS Session Service with minor differences with netbios_ssn - NETBIOS Session Service - simple service . It is supported only by Windows 2000 and Windows XP and it offers the advantage of being independent of WINS for name resolution.

It seems that samba supports transparently this protocol on the netbios_ssn - NETBIOS Session Service - simple service ports, so that either direct hosted or traditional SMB can be served simultaneously.

Please refer to the netbios_ssn - NETBIOS Session Service - simple service for more information.

MMS - MICROSOFT MEDIA SERVER - SIMPLE SERVICE

Example
Configuration sample:

```
server mms accept
```

Server Ports
tcp/1755 udp/1755

Client Ports
default

Netfilter Modules
See here.

Netfilter NAT Modules
See here.

Links
Wikipedia

Notes
Microsoft's proprietary network streaming protocol used to transfer unicast data in Windows Media Services (previously called NetShow Services).

MSN - MICROSOFT MSN MESSENGER SERVICE - SIMPLE SERVICE

Example
Configuration sample:

```
server msn accept
```

Server Ports
tcp/1863 udp/1863

Client Ports
default

MSNP - MSNP - SIMPLE SERVICE

Example
Configuration sample:

```
server msnp accept
```

Server Ports
tcp/6891

Client Ports
default

MS_DS - DIRECT HOSTED (NETBIOS-LESS) SMB - SIMPLE SERVICE

Alias
See microsoft_ds - Direct Hosted (NETBIOS-less) SMB - simple service

MULTICAST - MULTICAST - COMPLEX SERVICE

Example
Configuration sample:

```
server multicast reject with proto-unreach
```

Server Ports
N/A

Client Ports
N/A

Links
Wikipedia

Notes
The multicast service matches all packets sent to 224.0.0.0/4 using IGMP or UDP.

MYSQL - MYSQL - SIMPLE SERVICE

Example
Configuration sample:

```
server mysql accept
```

Server Ports
tcp/3306

Client Ports
default

Links
Homepage, Wikipedia

NETBACKUP - VERITAS NETBACKUP SERVICE - SIMPLE SERVICE

Example
Configuration sample:

```
server netbackup accept
client netbackup accept
```

Server Ports
tcp/13701 tcp/13711 tcp/13720 tcp/13721 tcp/13724 tcp/13782 tcp/13783

Client Ports
any

Links
Wikipedia

Notes
To use this service you must define it as both client and server in NetBackup clients and NetBackup servers.

NETBIOS_DGM - NETBIOS DATAGRAM DISTRIBUTION SERVICE - SIMPLE SERVICE

Example
Configuration sample:

```
server netbios_dgm accept
```

Server Ports
udp/138

Client Ports
any

Links
Wikipedia

Notes
See also the samba - Samba - complex service .

Keep in mind that this service broadcasts (to the broadcast address of your LAN) UDP packets. If you place this service within an interface that has a dst parameter, remember to include (in the dst parameter) the broadcast address of your LAN too.

NETBIOS_NS - NETBIOS NAME SERVICE - SIMPLE SERVICE

Example
Configuration sample:

```
server netbios_ns accept
```

Server Ports
udp/137

Client Ports
any

Links
Wikipedia

Notes
See also the samba - Samba - complex service .

NETBIOS_SSN - NETBIOS SESSION SERVICE - SIMPLE SERVICE

Example
Configuration sample:

```
server netbios_ssn accept
```

Server Ports
tcp/139

Client Ports
default

Links
Wikipedia

Notes
See also the samba - Samba - complex service .

Please keep in mind that newer NETBIOS clients prefer to use port 445 (microsoft_ds - Direct Hosted (NETBIOS-less) SMB - simple service) for the NETBIOS session service, and when this is not available they fall back to port 139 (netbios_ssn). Versions of samba above 3.x bind automatically to ports 139 and 445.

If you have an older samba version and your policy on an interface or router is DROP, clients trying to access port 445 will have to timeout before falling back to port 139. This timeout can be up to several minutes.

To overcome this problem you can explicitly REJECT the microsoft_ds - Direct Hosted (NETBIOS-less) SMB - simple service with a tcp-reset message:

server microsoft_ds reject with tcp-reset

NFS - NETWORK FILE SYSTEM - COMPLEX SERVICE

Example

Configuration sample:

```
client nfs accept dst 192.0.2.1
```

Server Ports

many

Client Ports

N/A

Links

Wikipedia

Notes

The NFS service queries the RPC service on the NFS server host to find out the ports nfsd, mountd, lockd and rquotad are listening. Then, according to these ports it sets up rules on all the supported protocols (as reported by RPC) in order the clients to be able to reach the server.

For this reason, the NFS service requires that:

the firewall is restarted if the NFS server is restarted
the NFS server must be specified on all nfs statements (only if it is not the localhost)

Since NFS queries the remote RPC server, it is required to also be allowed to do so, by allowing the portmap - Open Network Computing Remote Procedure Call - Port Mapper - simple service too. Take care that this is allowed by the running firewall when Sanewall tries to query the RPC server. So you might have to setup NFS in two steps: First add the portmap service and activate the firewall, then add the NFS service and restart the firewall.

To avoid this you can setup your NFS server to listen on pre-defined ports, as documented in NFS Howto. If you do this then you will have to define the the ports using the procedure described in the section called "Adding Services" of Sanewall configuration: sanewall.conf(5).

NIS - NETWORK INFORMATION SERVICE - COMPLEX SERVICE

Example

Configuration sample:

```
client nis accept dst 192.0.2.1
```

Server Ports

many

Client Ports

N/A

Links

Wikipedia

Notes

The nis service queries the RPC service on the nis server host to find out the ports ypserv and yppasswdd are listening. Then, according to these ports it sets up rules on all the supported protocols (as reported by RPC) in order the clients to be able to reach the server.

For this reason, the nis service requires that:

the firewall is restarted if the nis server is restarted
the nis server must be specified on all nis statements (only if it is not the localhost)

Since nis queries the remote RPC server, it is required to also be allowed to do so, by allowing the portmap - Open Network Computing Remote Procedure Call - Port Mapper - simple service too. Take care that this is allowed by the running firewall when Sanewall tries to query the RPC server. So you might have to setup nis in two steps: First add the portmap service and activate the firewall, then add the nis service and restart the firewall.

This service was added to FireHOL by Carlos Rodrigues. His comments regarding this implementation, are:

These rules work for client access only!

Pushing changes to slave servers won't work if these rules are active somewhere between the master and its slaves, because it is impossible to predict the ports where yppush will be listening on each push.

Pulling changes directly on the slaves will work, and could be improved performance-wise if these rules are modified to open fypxfrd. This wasn't done because it doesn't make that much sense since pushing changes on the master server is the most common, and recommended, way to replicate maps.

NNTP - NETWORK NEWS TRANSFER PROTOCOL - SIMPLE SERVICE

Example

Configuration sample:

```
server nntp accept
```

Server Ports

tcp/119

Client Ports

default

Links

Wikipedia

NNTPS - SECURE NETWORK NEWS TRANSFER PROTOCOL - SIMPLE SERVICE

Example

Configuration sample:

```
server nntps accept
```

Server Ports
tcp/563

Client Ports
default

Links
Wikipedia

NRPE - NAGIOS NRPE - SIMPLE SERVICE

Server Ports
tcp/5666

Client Ports
default

Links
Wikipedia

NTP - NETWORK TIME PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server ntp accept
```

Server Ports
udp/123 tcp/123

Client Ports
any

Links
Wikipedia

NUT - NETWORK UPS TOOLS - SIMPLE SERVICE

Example
Configuration sample:

```
server nut accept
```

Server Ports
tcp/3493 udp/3493

Client Ports
default

Links
Homepage

NXSERVER - NOMACHINE NX SERVER - SIMPLE SERVICE

Example
Configuration sample:

```
server nxserver accept
```

Server Ports
tcp/5000:5200

Client Ports
default

Links
Wikipedia

Notes
Default ports used by NX server for connections without encryption.

Note that nxserver also needs the ssh - Secure Shell Protocol - simple service to be enabled.

This information has been extracted from this The TCP ports used by nxserver are 4000 + DISPLAY_BASE to 4000 + DISPLAY_BASE + DISPLAY_LIMIT. DISPLAY_BASE and DISPLAY_LIMIT are set in /usr/NX/etc/node.conf and the defaults are DISPLAY_BASE=1000 and DISPLAY_LIMIT=200.

For encrypted nxserver sessions, only ssh - Secure Shell Protocol - simple service is needed.

OPENVPN - OPENVPN - SIMPLE SERVICE

Server Ports
tcp/1194 udp/1194

Client Ports
default

Links
Homepage, Wikipedia

ORACLE - ORACLE DATABASE - SIMPLE SERVICE

Example
Configuration sample:

```
server oracle accept
```

Server Ports
tcp/1521

Client Ports
default

Links
Wikipedia

OSPF - OPEN SHORTEST PATH FIRST - SIMPLE SERVICE

Example
Configuration sample:

```
server OSPF accept
```

Server Ports
89/any

Client Ports
any

Links
Wikipedia

PING - PING (ICMP ECHO) - COMPLEX SERVICE

Example
Configuration sample:

```
server ping accept
```

Server Ports
N/A

Client Ports
N/A

Links
Wikipedia

Notes

This services matches requests of protocol ICMP and type echo-request (TYPE=8) and their replies of type echo-reply (TYPE=0).

The ping service is stateful.

POP3 - POST OFFICE PROTOCOL - SIMPLE SERVICE

Example

Configuration sample:

```
server pop3 accept
```

Server Ports

tcp/110

Client Ports

default

Links

Wikipedia

POP3S - SECURE POST OFFICE PROTOCOL - SIMPLE SERVICE

Example

Configuration sample:

```
server pop3s accept
```

Server Ports

tcp/995

Client Ports

default

Links

Wikipedia

PORTMAP - OPEN NETWORK COMPUTING REMOTE PROCEDURE CALL - PORT MAPPER - SIMPLE SERVICE

Example

Configuration sample:

```
server portmap accept
```

Server Ports
udp/111 tcp/111

Client Ports
any

Links
Wikipedia

POSTGRES - POSTGRESQL - SIMPLE SERVICE

Example
Configuration sample:

```
server postgres accept
```

Server Ports
tcp/5432

Client Ports
default

Links
Wikipedia

PPTP - POINT-TO-POINT TUNNELING PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server pptp accept
```

Server Ports
tcp/1723

Client Ports
default

Netfilter Modules
nf_conntrack_pptp (CONFIG_NF_CONNTRACK_PPTP), nf_conntrack_proto_gre (CONFIG_NF_CT_PROTO_GRE)

Netfilter NAT Modules
nf_nat_pptp (CONFIG_NF_NAT_PPTP), nf_nat_proto_gre (CONFIG_NF_NAT_PROTO_GRE)

Links
Wikipedia

PRIVOXY - PRIVACY PROXY - SIMPLE SERVICE

Example
Configuration sample:

```
server privoxy accept
```

Server Ports
tcp/8118

Client Ports
default

Links
Homepage

RADIUS - REMOTE AUTHENTICATION DIAL IN USER SERVICE (RADIUS) - SIMPLE SERVICE

Example
Configuration sample:

```
server radius accept
```

Server Ports
udp/1812 udp/1813

Client Ports
default

Links
Wikipedia

RADIUSOLD - REMOTE AUTHENTICATION DIAL IN USER SERVICE (RADIUS) - SIMPLE SERVICE

Example
Configuration sample:

```
server radiusold accept
```

Server Ports
udp/1645 udp/1646

Client Ports
default

Links
Wikipedia

RADIUSOLDPROXY - REMOTE AUTHENTICATION DIAL IN USER SERVICE (RADIUS) - SIMPLE SERVICE

Example
Configuration sample:

```
server radiusoldproxy accept
```

Server Ports
udp/1647

Client Ports
default

Links
Wikipedia

RADIUSPROXY - REMOTE AUTHENTICATION DIAL IN USER SERVICE (RADIUS) - SIMPLE SERVICE

Example
Configuration sample:

```
server radiusproxy accept
```

Server Ports
udp/1814

Client Ports
default

Links
Wikipedia

RDP - REMOTE DESKTOP PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server rdp accept
```

Server Ports
tcp/3389

Client Ports
default

Links
Wikipedia

Notes
Remote Desktop Protocol is also known also as Terminal Services.

RNDC - REMOTE NAME DAEMON CONTROL - SIMPLE SERVICE

Example
Configuration sample:

```
server rndc accept
```

Server Ports
tcp/953

Client Ports
default

Links
Wikipedia

RSYNC - RSYNC PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server rsync accept
```

Server Ports
tcp/873 udp/873

Client Ports
default

Links
Homepage, Wikipedia

RTP - REAL-TIME TRANSPORT PROTOCOL - SIMPLE SERVICE

Example

Configuration sample:

```
server rtp accept
```

Server Ports

udp/10000:20000

Client Ports

any

Links

Wikipedia

Notes

RTP ports are generally all the UDP ports. This definition narrows down RTP ports to UDP 10000 to 20000.

SAMBA - SAMBA - COMPLEX SERVICE

Example

Configuration sample:

```
server samba accept
```

Server Ports

many

Client Ports

default

Links

Homepage, Wikipedia

Notes

The samba service automatically sets all the rules for netbios_ns - NETBIOS Name Service - simple service , netbios_dgm - NETBIOS Datagram Distribution Service - simple service , netbios_ssn - NETBIOS Session Service - simple service and microsoft_ds - Direct Hosted (NETBIOS-less) SMB - simple service .

Please refer to the notes of the above services for more information.

NETBIOS initiates based on the broadcast address of an interface (request goes to broadcast address) but the server responds from its own IP address. This makes the "server samba accept" statement drop the server reply, because of the way the iptables connection tracker works.

This service definition includes a hack, that allows a Linux samba server to respond correctly in such situations, by allowing new outgoing connections from the well known netbios_ns - NETBIOS Name Service - simple service port to the clients high ports.

However, for clients and routers this hack is not applied because it would open all unprivileged ports to the samba server. The only solution to overcome the problem in such cases (routers or clients) is to build a trust relationship between the samba servers and clients.

SANE - SANE SCANNER SERVICE - SIMPLE SERVICE

Server Ports
tcp/6566

Client Ports
default

Netfilter Modules
nf_conntrack_sane (CONFIG_NF_CONNTRACK_SANE)

Netfilter NAT Modules
N/A

Links
Homepage

SIP - SESSION INITIATION PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server sip accept
```

Server Ports
udp/5060

Client Ports
5060 default

Netfilter Modules
nf_conntrack_sip (CONFIG_NF_CONNTRACK_SIP)

Netfilter NAT Modules
nf_nat_sip (CONFIG_NF_NAT_SIP)

Links
Wikipedia

Notes
SIP is an IETF standard protocol (RFC 2543) for initiating interactive user sessions involving multimedia elements such as video, voice, chat, gaming, etc. SIP works in the application layer of the OSI communications model.

SMTP - SIMPLE MAIL TRANSPORT PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server smtp accept
```

Server Ports
tcp/25

Client Ports
default

Links
Wikipedia

SMTPS - SECURE SIMPLE MAIL TRANSPORT PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server smtps accept
```

Server Ports
tcp/465

Client Ports
default

Links
Wikipedia

SNMP - SIMPLE NETWORK MANAGEMENT PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server snmp accept
```

Server Ports
udp/161

Client Ports
default

Links
Wikipedia

SNMPTRAP - SNMP TRAP - SIMPLE SERVICE

Example
Configuration sample:

```
server snmptrap accept
```

Server Ports
udp/162

Client Ports
any

Links
Wikipedia

Notes
An SNMP trap is a notification from an agent to a manager.

SOCKS - SOCKET SECURE - SIMPLE SERVICE

Example
Configuration sample:

```
server socks accept
```

Server Ports
tcp/1080 udp/1080

Client Ports
default

Links
Wikipedia

Notes
See also RFC 1928.

SQUID - SQUID WEB CACHE - SIMPLE SERVICE

Example
Configuration sample:

```
server squid accept
```

Server Ports
tcp/3128

Client Ports
default

Links
Homepage, Wikipedia

SSH - SECURE SHELL PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server ssh accept
```

Server Ports
tcp/22

Client Ports
default

Links
Wikipedia

STUN - SESSION TRAVERSAL UTILITIES FOR NAT - SIMPLE SERVICE

Example
Configuration sample:

```
server stun accept
```

Server Ports
udp/3478 udp/3479

Client Ports
any

Links
Wikipedia

Notes
STUN is a protocol for assisting devices behind a NAT firewall or router with their packet routing.

SUBMISSION - SMTP OVER SSL/TLS SUBMISSION - SIMPLE SERVICE

Example
Configuration sample:

```
server submission accept
```

Server Ports
tcp/587

Client Ports
default

Links
Wikipedia

Notes
Submission is essentially normal SMTP with an SSL/TLS negotiation.

SUNRPC - OPEN NETWORK COMPUTING REMOTE PROCEDURE CALL - PORT MAPPER - SIMPLE SERVICE

Alias
See portmap - Open Network Computing Remote Procedure Call - Port Mapper - simple service

SWAT - SAMBA WEB ADMINISTRATION TOOL - SIMPLE SERVICE

Example
Configuration sample:

```
server swat accept
```

Server Ports
tcp/901

Client Ports
default

Links
Homepage

SYSLOG - SYSLOG REMOTE LOGGING PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server syslog accept
```

Server Ports
udp/514

Client Ports
syslog default

Links
Wikipedia

TELNET - TELNET - SIMPLE SERVICE

Example
Configuration sample:

```
server telnet accept
```

Server Ports
tcp/23

Client Ports
default

Links
Wikipedia

TFTP - TRIVIAL FILE TRANSFER PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server tftp accept
```

Server Ports
udp/69

Client Ports
default

Netfilter Modules
nf_conntrack_tftp (CONFIG_NF_CONNTRACK_TFTP)

Netfilter NAT Modules
nf_nat_tftp (CONFIG_NF_NAT_TFTP)

Links
Wikipedia

TIME - TIME PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server time accept
```

Server Ports
tcp/37 udp/37

Client Ports
default

Links
Wikipedia

TIMESTAMP - ICMP TIMESTAMP - COMPLEX SERVICE

Example
Configuration sample:

```
server timestamp accept
```

Server Ports
N/A

Client Ports
N/A

Links
Wikipedia

Notes
This services matches requests of protocol ICMP and type timestamp-request (TYPE=13) and their replies of type timestamp-reply (TYPE=14).

The timestamp service is stateful.

TOMCAT - HTTP ALTERNATE PORT - SIMPLE SERVICE

Alias
See httpalt - HTTP alternate port - simple service

UPNP - UNIVERSAL PLUG AND PLAY - SIMPLE SERVICE

Example
Configuration sample:

```
server upnp accept
```

Server Ports
udp/1900 tcp/2869

Client Ports
default

Links
Homepage, Wikipedia

Notes
For a Linux implementation see: Linux IGD.

UUCP - UNIX-TO-UNIX COPY - SIMPLE SERVICE

Example
Configuration sample:

```
server uucp accept
```

Server Ports
tcp/540

Client Ports
default

Links
Wikipedia

VMWARE - VMWARE - SIMPLE SERVICE

Example
Configuration sample:

```
server vmware accept
```

Server Ports
tcp/902

Client Ports
default

Notes
Used from VMWare 1 and up. See the VMWare KnowledgeBase.

VMWAREAUTH - VMWAREAUTH - SIMPLE SERVICE

Example
Configuration sample:

```
server vmwareauth accept
```

Server Ports
tcp/903

Client Ports
default

Notes
Used from VMWare 1 and up. See the VMWare KnowledgeBase.

VMWAREWEB - VMWAREWEB - SIMPLE SERVICE

Example
Configuration sample:

```
server vmwareweb accept
```

Server Ports
tcp/8222 tcp/8333

Client Ports
default

Notes
Used from VMWare 2 and up. See VMWare Server 2.0 release notes and the VMWare KnowledgeBase.

VNC - VIRTUAL NETWORK COMPUTING - SIMPLE SERVICE

Example
Configuration sample:

```
server vnc accept
```

Server Ports
tcp/5900:5903

Client Ports
default

Links
Wikipedia

Notes
VNC is a graphical desktop sharing protocol.

WEBCACHE - HTTP ALTERNATE PORT - SIMPLE SERVICE

Alias
See httpalt - HTTP alternate port - simple service

WEBMIN - WEBMIN ADMINISTRATION SYSTEM - SIMPLE SERVICE

Example
Configuration sample:

```
server webmin accept
```

Server Ports
tcp/10000

Client Ports
default

Links
Homepage

WHOIS - WHOIS PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server whois accept
```

Server Ports
tcp/43

Client Ports
default

Links
Wikipedia

XBOX - XBOX LIVE - COMPLEX SERVICE

Example
Configuration sample:

```
client xbox accept
```

Server Ports
many

Client Ports
default

Notes
Definition for the Xbox live service.

See program source for contributor details.

XDMCP - X DISPLAY MANAGER CONTROL PROTOCOL - SIMPLE SERVICE

Example
Configuration sample:

```
server xdmcp accept
```

Server Ports
udp/177

Client Ports
default

Links
Wikipedia

Notes
See Gnome Display Manager for a discussion about XDMCP and firewalls (Gnome Display Manager is a replacement for XDM).

See Also

Sanewall program: sanewall(1)
Sanewall configuration: sanewall.conf(5)

Chapter 4

Index