FireHOL Reference

Version 3.0.1 (Built 28 Feb 2016)

Contents

The latest version of this manual is available online as a PDF, as single page HTML and also as multiple pages within the website.

1 FireHOL Reference

1.1 Who should read this manual

This is a reference guide with specific detailed information on commands and configuration syntax for the FireHOL tool. The reference is unlikely to be suitable for newcomers to the tools, except as a means to look up more information on a particular command.

For tutorials and guides to using FireHOL and FireQOS, please visit the website.

1.2 Where to get help

The FireHOL website.

The mailing lists and archives.

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

1.3 Installation

You can download tar-file releases by visiting the FireHOL website download area.

Unpack and change directory with:

```
tar xfz firehol-version.tar.gz
cd firehol-version
```

Options for the configure program can be seen in the INSTALL file and by running:

```
./configure --help
```

To build and install taking the default options:

```
./configure && make && sudo make install
```

Alternatively, just copy the `sbin/firehol.in` file to where you want it. All of the common SysVInit command line arguments are recognised which makes it easy to deploy the script as a startup service.

Packages are available for most distributions and you can use your distribution's standard commands (e.g. aptitude, yum, etc.) to install these.

> **Note**
>
> Distributions do not always offer the latest version. You can see what the latest release is on the FireHOL website.

1.4 Licence

2 Setting up and running FireHOL

FireHOL is started and stopped using the firehol script. The default firewall configuration is to be found in /etc/firehol/firehol.conf, with some behaviours governed by variables in /etc/firehol/firehol-defaults.conf.

3 Primary commands

These are the primary packet filtering building blocks. Below each of these, sub-commands can be added.

command	4/6/46	forbidden params	description
interface	Y	inface outface physout	Define packet filtering blocks, protecting the firewall host itself.
router	Y	-	Define packet filtering blocks, protecting other hosts from routed traffic.

4 Sub-commands

A rule in an `interface` or `router` definition typically consists of a subcommand to apply to a service using one of the standard actions provided it matches certain optional rule parameters. e.g.

```
server ssh accept src 10.0.0.0/8
```

The following sub-commands can be used below **primary commands** to form rules.

command	4/6/46	forbidden params	description
client	Y	sport dport	Allow access to a client running on the `interface` or the protected `router` hosts.
group	Y	-	Define groups of commands that share optional rule parameters. Groups can be nested.
iptables ip6tables	N	*all forbidden*	A wrapper for the system iptables command, to add custom iptables statements to a FireHOL firewall.

command	4/6/46	forbidden params	description
masquerade	Y	inface outface	Change the source IP of packets leaving `outface`, with the IP of the interface they are using to leave.
policy	N	*all* *forbidden*	Define the action to be applied on packets not matched by any `server` or `client` statements in the `interface` or `router`.
protection	N	*all* *forbidden*	Examine incoming packets per `interface` or `router` and filter out bad packets or limit request frequency.
server	Y	sport dport	Allow access to a server running on the `interface` or the protected `router` hosts.
tcpmss	Y	*all* *forbidden*	Set the MSS (Maximum Segment Size) of TCP SYN packets routed through the firewall.

5 Helper commands

The following commands are generally used to set things up before the first **primary command**. Some can be used below an `interface` or `router` and also appear in the subcommands table.

command	4/6/46	forbidden params	description
action	Y	-	Define new actions that can differentiate the final action based on rules. `action` can be used to define traps.
blacklist	Y	-	Drop matching packets globally.
classify	Y	-	Put matching traffic into the specified traffic shaping class.
connmark	Y	-	Set a stateful mark from the `connmark` group.
dscp	Y	-	Set the DSCP field of packets.
ipset	4/6	*all* *forbidden*	Define ipsets. A wrapper for the system ipset command to add ipsets to a FireHOL firewall.

command	4/6/46	forbidden params	description
iptables ip6tables	N	*all* *forbidden*	A wrapper for the system iptables command, to add custom iptables statements to a FireHOL firewall.
iptrap	4/6	-	Dynamically put IP addresses in an ipset.
mac	Y	*all* *forbidden*	Restricts an IP to a particular MAC address.
mark	Y	-	Set a stateful mark from the `usermark` group.
masquerade	Y	-	Change the source IP of packets leaving `outface`, with the IP of the interface they are using to leave.
dnat	Y	-	Change the destination IP or port of packets received, to fixed values or fixed ranges. `dnat` can be used to implement load balancers.
snat	Y	-	Change the source IP or port of packets leaving, to fixed values or fixed ranges.
redirect	Y	-	Redirect packets to the firewall host, possibly changing the destination port. Can support load balancers if multiple daemons run on localhost.
transparent_proxy	Y	*see notes*	Set up a transparent TCP, HTTP or squid proxy.
synproxy	Y	-	Configure synproxy.
tcpmss	Y	*all* *forbidden*	Set the MSS (Maximum Segment Size) of TCP SYN packets routed through the firewall.
tos	Y	-	Set the Type of Service (TOS) of packets.
tosfix	Y	*all* *forbidden*	Apply suggested TOS values to packets.
version	N	*all* *forbidden*	Specify a version number for the configuration file.

6 Manual Pages in Alphabetical Order

6.1 firehol(1)

6.1.1 NAME

firehol - an easy to use but powerful iptables stateful firewall

6.1.2 SYNOPSIS

firehol

sudo -E firehol panic [*IP*]

firehol *command* [– *conf-arg*...]

firehol *CONFIGFILE* [start|debug|try] [– *conf-arg*...]

6.1.3 DESCRIPTION

Running `firehol` invokes iptables(8) to manipulate your firewall.

Run without any arguments, `firehol` will present some help on usage.

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

6.1.3.1 PANIC

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

FireHOL removes all rules from the running firewall and then DROPs all traffic on all iptables(8) 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, FireHOL checks for the existence of the SSH_CLIENT shell environment variable, which is set by ssh(1). If it finds this, then panic mode will allow the established SSH connection specified in this variable to operate.

> **Note**
>
> In order for FireHOL to see the environment variable you must ensure that it is preserved. For sudo(8) use the `-E` and for su(1) 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.

6.1.4 COMMANDS

start; restart Activates the firewall using `/etc/firehol/firehol.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(8) 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 Restarts the FireHOL firewall only if it is already active. This is the generally expected behaviour (but opposite to FireHOL prior to 2.0.0-pre4).

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

save Start the firewall and then save it using iptables-save(8) to the location given by FIREHOL_AUTOSAVE. See firehol-variables(5) for more information.

The required kernel modules are saved to an executable shell script `/var/spool/firehol/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 FireHOL will work.

This is because as part of starting a firewall, FireHOL 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, FireHOL shows the generated iptables(8) statements.

explain Enters an interactive mode where FireHOL accepts normal configuration commands and presents the generated iptables(8) commands for each of them, together with some reasoning for its purpose.

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

Some extra commands are available in `explain` mode.

help Present some help
show Present the generated configuration
quit Exit interactive mode and quit

helpme; wizard Tries to guess the FireHOL configuration needed for the current machine.

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

The generated FireHOL 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.

6.1.5 FILES

`/etc/firehol/firehol.conf`

6.1.6 SEE ALSO

- firehol.conf(5) - FireHOL configuration
- firehol-variables(5) - control variables
- FireHOL Website
- FireHOL Online PDF Manual

- FireHOL Online Documentation

6.2 firehol.conf(5)

6.2.1 NAME

firehol.conf - FireHOL configuration

6.2.2 DESCRIPTION

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

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

```
version 6
```

See firehol-version(1) 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 firehol-interface(5) for full details.

By default FireHOL will try to create both IPv4 and IPv6 rules for each interface. To make this explicit or restrict which rules are created write `both interface`, `ipv4 interface` or `ipv6 interface`.

Note that IPv6 will be disabled silently if your system is not configured to use it. You can test this by looking for the file `/proc/net/if_inet6`. The IPv6 HOWTO has more information.

A configuration file contains zero 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 firehol-router(5) for full details.

By default FireHOL will try to create both IPv4 and IPv6 rules for each router. To make this explicit or restrict which rules are created write `both router`, `ipv4 router` or `ipv6 router`.

It is simple to add extra service definitions which can then be used in the same way as those provided as standard. See 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 may be set up and used outside of any definition, see firehol-variables(5) as can the functions in CONFIGURATION HELPER COMMANDS and HELPER COMMANDS.

6.2.3 VARIABLES AVAILABLE

The following variables are made available in the FireHOL 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/firehol/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/firehol/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/firehol/MULTICAST_IPS`.

6.2.4 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(8) 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 firehol-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 FIREHOL_NAT is non-zero (see firehol-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:

```
# Setup service
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), FireHOL simply determines all of the combinations of client and server ports and generates multiple iptables(8) 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 firehol(1) script.

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/firehol/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. The 213 originally referred to a FireHOL 1.x minor version but is no longer checked.
>
> FireHOL will refuse to run if the API version does not match the expected one.

6.2.5 DEFINITIONS

- firehol-interface(5) - interface definition
- firehol-router(5) - router definition

6.2.6 SUBCOMMANDS

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

6.2.7 HELPER COMMANDS

These helpers can be used in `interface` and `router` definitions as well as before them:

- firehol-iptables(5) - iptables helper
- firehol-masquerade(5) - masquerade helper

This helper can be used in `router` definitions as well as before any `router` or `interface`:

- firehol-tcpmss(5) - tcpmss helper

6.2.8 CONFIGURATION HELPER COMMANDS

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

- firehol-version(5) - version config helper
- firehol-action(5) - action config helper
- firehol-blacklist(5) - blacklist config helper
- firehol-classify(5) - classify config helper
- firehol-connmark(5) - connmark config helper
- firehol-dscp(5) - dscp config helper
- firehol-mac(5) - mac config helper
- firehol-mark(5) - mark config helper
- firehol-nat(5) - nat, snat, dnat, redirect helpers
- firehol-proxy(5) - transparent proxy/squid helpers
- firehol-tos(5) - tos config helper
- firehol-tosfix(5) - tosfix config helper

6.2.9 SEE ALSO

- firehol(1) - FireHOL program
- firehol-variables(5) - control variables
- firehol-services(5) - services list
- firehol-actions(5) - actions for rules
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.3 firehol-action(5)

6.3.1 NAME

firehol-action - set up custom filtering actions

6.3.2 SYNOPSIS

action *name* [table *table_name*] *type type_params* [next [type *type_params* [next ...]]]

6.3.3 DESCRIPTION

The `action` helper creates custom actions that can be used everywhere in FireHOL, like this:

```
action ACT1 chain accept

interface any world
    server smtp ACT1

router myrouter
    policy ACT1
```

The `action` helper allows linking multiple actions together and having some logic to select which action to execute, like this:

```
action ACT1 \
         rule src 192.168.0.0/16 action reject \
    next rule dst 192.168.0.0/16 action reject \
    next rule inface eth2 action drop \
    next rule outface eth2 action drop \
    next action accept

interface any world
    server smtp ACT1

router myrouter
    policy ACT1
```

There is no limit on the number of actions that can be linked together.

`type` can be `chain` or `action` (`chain` and `action` are aliases), `rule`, `iptrap`, `ipuntrap` or `sockets_suspects_trap`.

6.3.3.1 Chain type actions

This is the simpler action. It creates an iptables(8) 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 FireHOL (see firehol-actions(5)). Only ACCEPT, REJECT, DROP, RETURN have any meaning in this instance.

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

```
action ACT1 chain accept

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

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.

6.3.3.2 Rule type actions

`rule` type actions define a few conditions that will lead to an action.

All optional rule parameters FireHOL supports can be used here (see firehol-params(5)).

```
action ACT1 \
    rule inface eth0 action accept
    next rule outface eth0 action accept
    next action reject

interface any world
    server smtp ACT1
```

In the above example the smtp server can only be accessed from eth0.

It is important to remember that actions will be applied for all the traffic, both requests and replies. The type of traffic can be filtered with the `state` optional rule parameter, like this:

```
action ACT1 \
    rule inface eth0 state NEW action reject
    next action accept

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

In the above example, the smtp server will not accept NEW connections from eth0, but the smtp client will be able to connect to servers on eth0 (and everywhere else).

6.3.3.3 iptrap type actions

`iptrap` (see firehol-iptrap(5)) is a helper than copies (traps) an IP to an ipset (see firehol-ipset(5)). It does not perform any action on the traffic.

Using the `iptrap` action, the `iptrap` helper can be linked to filtering actions, like this:

```
 # a simple version of TRAP_AND_REJECT
 # this uses just 2 ipsets, one for counting packets (policytrap)
 # and one to store the banned IPs (trap).
 # it also needs a ipset called whitelist, for excluded source IPs.
 # it will ban IPs when they have 50+ reject packets
 action4 TRAP_AND_REJECT \
    rule iptrap src policytrap 30 inface "${wan}" \
        src not "${UNROUTABLE_IPS} ipset:whitelist" \
        state NEW log "POLICY TRAP" \
```

```
   next iptrap trap src 86400 \
       state NEW log "POLICY TRAP - BANNED" \
       ipset policytrap src no-counters packets-above 50 \
   next action reject

# a complete TRAP_AND_REJECT
# this uses 3 ipset, one for keeping track of the rejected sockets
# per source IP (called 'sockets'), one for counting the sockets
# per source IP (called 'suspects') and one to store the banned IPs
# (called 'trap').
# it also needs a ipset called whitelist, for excluded source IPs.
# it will ban IPs when they have 3 or more rejected sockets
action4 TRAP_AND_REJECT \
   iptrap sockets src,dst,dst 3600 method hash:ip,port,ip counters \
       state NEW log "TRAP AND REJECT - NEW SOCKET" \
       inface "${wan}" \
       src not "${UNROUTABLE_IPS} ipset:whitelist" \
   next iptrap suspects src 3600 counters \
       state NEW log "TRAP AND REJECT - NEW SUSPECT" \
       ipset sockets src,dst,dst no-counters packets 1 \
   next iptrap trap src 86400 \
       state NEW log "TRAP AND REJECT - BANNED" \
       ipset suspects src no-counters packets-above 2 \
   next action REJECT

interface any world
   policy TRAP_AND_REJECT
   protection bad-packets
   ...

router wan2lan inface "${wan}" outface "${lan}"
   policy TRAP_AND_REJECT
   protection bad-packets
   ...
```

Since we used the action TRAP_AND_REJECT as an interface policy, it will get all the traffic not accepted, rejected, or dropped by the server and client statements.

For all these packets, the action TRAP_AND_REJECT will first check that they are coming in from wan0, that their src IP is not in `UNROUTABLE_IPS` list and in the `whitelist` ipset, that they are NEW connections, and if all these conditions are met, it will log with the tag `POLICY TRAP` and add the src IP of the packets in the `policytrap` ipset for 30 seconds.

All traffic not matched by the above, will be just rejected.

6.3.3.4 sockets_suspects_trap type actions

The type `sockets_suspects_trap` will automatically a custom trap using the following template:

```
action4 *name* sockets_suspects_trap *SUSPECTS_TIMEOUT* *TRAP_TIMEOUT* *VALID_CONNECTIONS* |
```

This will:

1. Create the ipset `${name}_sockets` where the matched sockets will be stored for `SUSPECTS_TIMEOUT` seconds.
2. Create the ipset `${name}_suspects` where the source IPs of the matched sockets will be stored for `SUSPECTS_TIMEOUT` seconds.
3. Create the ipset `${name}_trap` where the trapped IPs will be stored for `TRAP_TIMEOUT` seconds. IPs will be added to this ipset only if more than `VALID_CONNECTIONS` have been matched by this IP.

`optional params` are FireHOL optional rule parameters (firehol-params(5)) that can be used to limit the match for the first ipset (sockets).

So, to design the same TRAP_AND_REJECT as above, this statement is needed:

```
action4 TRAP_AND_REJECT \
    sockets_suspects_trap 3600 86400 2 \
        inface "${wan}" \
        src not "${UNROUTABLE_IPS} ipset:whitelist" \
    next action REJECT
```

The ipsets that will be created will be named: `TRAP_AND_REJECT_sockets`, `TRAP_AND_REJECT_suspects` and `TRAP_AND_REJECT_trap`.

> **Note** Always terminate `sockets_suspects_trap` with a `next action DROP` or `next action REJECT`, or the traffic will continue to flow.

6.3.4 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-actions(5) - optional rule parameters
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.4 firehol-actions(5)

6.4.1 NAME

firehol-actions - actions for rules

6.4.2 SYNOPSIS

accept

accept with hashlimit *name* upto|above *amount/period* [burst *amount*] [mode *{srcip|srcport|dstip|dstport},...*] [srcmask *prefix*] [dstmask *prefix*] [htable-size *buckets*] [htable-max *entries*] [htable-expire *msec*] [htable-gcinterval *msec*]

accept with connlimit upto|above *limit* [mask *mask*] [saddr|daddr]

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

6.4.3 DESCRIPTION

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

FireHOL 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 (FireHOL will generate multiple firewall rules to make this happen). For more information see log and loglimit.

The following actions are defined:

6.4.3.1 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
```

6.4.3.2 accept with hashlimit *name* upto|above *amount/period* [burst *amount*] [mode *{srcip|srcport|dstip|dstport},...*] [srcmask *prefix*] [dstmask *prefix*] [htable-size *buckets*] [htable-max *entries*] [htable-expire *msec*] [htable-gcinterval *msec*]

`hashlimit` hashlimit uses hash buckets to express a rate limiting match (like the limit match) for a group of connections using a single iptables rule. Grouping can be done per-hostgroup (source and/or destination address) and/or per-port.

name The name for the /proc/net/ipt_hashlimit/*name* entry.

`upto` *amount[/second|/minute|/hour|/day]* Match if the rate is below or equal to amount/quantum. It is specified either as a number, with an optional time quantum suffix (the default is 3/hour).

`above` *amount[/second|/minute|/hour|/day]* Match if the rate is above amount/quantum.

`burst` *amount* Maximum initial number of packets to match: this number gets recharged by one every time the limit specified above is not reached, up to this number; the default is 5. This option should be used with caution - if the entry expires, the burst value is reset too.

`mode` *{srcip|srcport|dstip|dstport},...* A comma-separated list of objects to take into consideration. If no `mode` option is given, *srcip,dstport* is assumed.

`srcmask` *prefix* When –hashlimit-mode srcip is used, all source addresses encountered will be grouped according to the given prefix length and the so-created subnet will be subject to hashlimit. prefix must be between (inclusive) 0 and 32. Note that `srcmask` *0* is basically doing the same thing as not specifying srcip for `mode`, but is technically more expensive.

`dstmask` *prefix* Like `srcmask`, but for destination addresses.

`htable-size` *buckets* The number of buckets of the hash table

`htable-max` *entries* Maximum entries in the hash.

`htable-expire` *msec* After how many milliseconds do hash entries expire.

`htable-gcinterval` *msec* How many milliseconds between garbage collection intervals.

Examples:

Allow up to 5 connections per second per client to SMTP server:

```
server smtp accept with hashlimit smtplimit upto 5/s
```

You can monitor it using the file /proc/net/ipt_hashlimit/smtplimit

6.4.3.3 accept with connlimit upto|above *limit* [mask *mask*] [saddr|daddr]

`accept with connlimit` matches on the number of connections per IP.

saddr matches on source IP. *daddr* matches on destination IP. *mask* groups IPs with the *mask* given *upto* matches when the number of connections is up to the given *limit above* matches when the number of connections above to the given *limit*

The number of connections counted are system wide, not service specific. For example for *saddr*, you cannot connlimit 2 connections for SSH and 4 for SMTP. If you connlimit 2 connections for SSH, then the first 2 connections of a client can be SSH. If a client has already 2 connections to another service, the client will not be able to connect to SSH.

So, `connlimit` can safely be used:

- with *daddr* to limit the connections a server can accept
- with *saddr* to limit the total connections per client to all services.

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

`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 timeouts on otherwise valid service clients).

Examples:

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

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

6.4.3.5 accept with recent *name seconds hits*

`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.

6.4.3.6 accept with knock *name*

`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 FireHOL 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. FireHOL already matches everything else needed for its rules to work.

6.4.3.7 reject

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

6.4.3.8 reject with *message*

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 REJECT WITH MESSAGES 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
```

6.4.3.9 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
```

6.4.3.10 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
```

6.4.3.11 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 FireHOL, 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:

6.4.3.12 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.

6.4.3.13 redirect; redirect to-port port

`redirect` is used internally by FireHOL helper commands.

Only FireHOL developers should need to use this action directly.

6.4.4 REJECT WITH MESSAGES

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.

6.4.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration

- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.5 firehol-blacklist(5)

6.5.1 NAME

firehol-blacklist - set up a unidirectional or bidirectional blacklist

6.5.2 SYNOPSIS

{ blacklist | blacklist4 | blacklist6 } [*type*] [inface *device*] [log *"text"*] [loglimit *"text"*] [accounting *accounting_name*] *ip...* [except *rule-params*]

6.5.3 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 `all` is supplied (or no type at all), 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 `him`, `her`, `it`, `this`, `these` is supplied, a unidirectional stateful blacklist will be generated. Connections can be established to such IP addresses, but the IP addresses will not be able to connect to the firewall or hosts protected by it.

Using `log` or `loglimit`, the `text` will be logged when matching packets are found.

Using `inface`, the blacklist will be created on the interface `device` only (this includes forwarded traffic).

`accounting` will update the NFACCT accounting with the name given.

If the keyword `except` is found, then all the parameters following it are rules to match packets that should excluded from the blacklist (i.e. they are a whitelist for this blacklist). See firehol-params(5) for more details.

Blacklists must be declared before the first router or interface.

IP Lists for abuse, malware, attacks, proxies, anonymizers, etc can be downloaded with the contrib/update-ipsets.sh script. Information about the supported IP Lists can be found at FireHOL IP Lists

6.5.4 EXAMPLES

```
blacklist full 192.0.2.1 192.0.2.2
blacklist input "192.0.2.3 192.0.2.4"
blacklist full inface eth0 log "BADGUY" 192.0.1.1 192.0.1.2
```

6.5.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- FireHOL IP Lists

6.6 firehol-classify(5)

6.6.1 NAME

firehol-classify - classify traffic for traffic shaping tools

6.6.2 SYNOPSIS

{ classify | classify46 } *class* [*rule-params*]

classify4 *class* [*rule-params*]

classify6 *class* [*rule-params*]

6.6.3 DESCRIPTION

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

The *class* is a class as used by iptables(8) and tc(8) (e.g. MAJOR:MINOR).

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

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

6.6.4 EXAMPLES

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

6.6.5 SEE ALSO

- firehol-params(5) - optional rule parameters
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- tc(8) - show / manipulate traffic control settings

- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- Linux Advanced Routing & Traffic Control HOWTO

6.7 firehol-client(5)

6.7.1 NAME

firehol-client - client command

6.7.2 SYNOPSIS

{ client | client46 } *service action* [*rule-params*]

client4 *service action* [*rule-params*]

client6 *service action* [*rule-params*]

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

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

The *action* can be any of the actions listed in firehol-actions(5).

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

> **Note**
>
> Writing `client4` is equivalent to writing `ipv4 client` and ensures this subcommand is applied only in the IPv4 firewall rules.
>
> Writing `client6` is equivalent to writing `ipv6 client` and ensures this subcommand is applied only in the IPv6 firewall rules.
>
> Writing `client46` is equivalent to writing `both client` and ensures this subcommand is applied in both the IPv4 and IPv6 firewall rules; it cannot be used as part an interface or router that is IPv4 or IPv6 only.

The default `client` inherits its behaviour from the enclosing interface or router.

6.7.4 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
```

6.7.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-modifiers(5) - ipv4/ipv6 selection
- firehol-services(5) - services list
- firehol-actions(5) - actions for rules
- firehol-params(5) - optional rule parameters
- firehol-server(5) - server subcommand
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.8 firehol-connmark(5)

6.8.1 NAME

firehol-connmark - set a stateful mark on a connection

6.8.2 SYNOPSIS

Warning - this manual page is out of date for nightly build/v3 behaviour

{ connmark | connmark46 } { value | save | restore } *chain rule-params*

connmark4 { value | save | restore } *chain rule-params*

connmark6 { value | save | restore } *chain rule-params*

6.8.3 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 firehol-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(8) 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 firehol-params(5) for more details.

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

6.8.4 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(8), to pick the correct routing table based on the mark on the packets.

6.8.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- firehol-mark(5) - mark traffic for traffic shaping tools
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- ip(8) - show / manipulate routing, devices, policy routing and tunnels
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- Linux Advanced Routing & Traffic Control HOWTO

6.9 firehol-dscp(5)

6.9.1 NAME

firehol-dscp - set the DSCP field in the packet header

6.9.2 SYNOPSIS

dscp { *value* | class *classid* } *chain rule-params*

6.9.3 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 firehol-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(8) 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(8) 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 firehol-params(5) for more details.

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

6.9.4 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
```

6.9.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- ip(8) - show / manipulate routing, devices, policy routing and tunnels
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- Linux Advanced Routing & Traffic Control HOWTO

6.10 firehol-group(5)

6.10.1 NAME

firehol-group - group commands with common options

6.10.2 SYNOPSIS

group with *rule-params*

group end

6.10.3 DESCRIPTION

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

Grouping commands with common options (see firehol-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.

6.10.4 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 telnet accept
  server ssh accept
group end
```

6.10.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.11 firehol-interface(5)

6.11.1 NAME

firehol-interface - interface definition

6.11.2 SYNOPSIS

{ interface | interface46 } *real-interface name rule-params*

interface4 *real-interface name rule-params*

interface6 *real-interface name rule-params*

6.11.3 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 from those listed in INTERFACE SUBCOMMANDS.

> **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 firehol-router(5)).

> **Note**
>
> Writing `interface4` is equivalent to writing `ipv4 interface` and ensures the defined interface is created only in the IPv4 firewall along with any rules within it.
>
> Writing `interface6` is equivalent to writing `ipv6 interface` and ensures the defined interface is created only in the IPv6 firewall along with any rules within it.
>
> Writing `interface46` is equivalent to writing `both interface` and ensures the defined interface is created in both the IPv4 and IPv6

firewalls. Any rules within it will also be applied to both, unless they specify otherwise.

6.11.4 PARAMETERS

real-interface This is the interface name as shown by `ip link show`. Generally anything iptables(8) 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 FireHOL interface and router definitions.

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

See firehol-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 firehol.conf(5) for an explanation of ${UNROUTABLE_IPS}.

6.11.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- firehol-modifiers(5) - ipv4/ipv6 selection
- firehol-router(5) - router definition
- firehol-iptables(5) - iptables helper
- firehol-masquerade(5) - masquerade helper
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.11.5.1 Interface Subcommands

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

6.12 firehol-ipset(5)

6.12.1 NAME

firehol-ipset - configure ipsets

6.12.2 SYNOPSIS

ipset *command name options*

6.12.3 DESCRIPTION

FireHOL has an `ipset` helper. It is a wrapper around the real `ipset` command and is handled internally within FireHOL in such a way so that the ipset collections defined in the configuration will be activated before activating the firewall.

FireHOL is also smart enough to restore the ipsets after a reboot, before it restores the firewall, so that everything will work as seamlessly as possible.

The `ipset` helper has the same syntax with the real `ipset` command. So in FireHOL you just add the `ipset` statements you need, and FireHOL will do the rest.

Keep in mind that each `ipset` collection is either IPv4 or IPv6. In FireHOL prefix `ipset` with either `ipv4` or `ipv6` and FireHOL will choose the right IP version (there is also `ipset4` and `ipset6`).

Also, do not add `-!` to ipset statements given in `firehol.conf`. FireHOL will batch import all ipsets and this option is not needed.

6.12.4 FireHOL ipset extensions

The features below are extensions of `ipset` that can only be used from within `firehol.conf`. They will not work on a terminal.

The FireHOL helper allows mass import of ipset collections from files. This is done with `ipset addfile` command.

The `ipset addfile` command will get a filename, remove all comments (anything after a `#` on the same line), trim any empty lines and spaces, and add all the remaining lines to `ipset`, as if each line of the file was run with `ipset add COLLECTION_NAME IP_FROM_FILE [other options]`.

The syntax of the `ipset addfile` command is:

```
ipset addfile *name* [ip|net] *filename* [*other ipset add options*]
```

`name` is the collection to add the IPs.

`ip` is optional and will select all the lines of the file that do not contain a `/`.

`net` is optional and will select all the lines of the file that contain a `/`.

`filename` is the filename to read. You can give absolute filenames and relative filenames (to `/etc/firehol`).

`other ipset add options` is whatever else `ipset add` supports, that you are willing to give for each line.

The `ipset add` command implemented in FireHOL also allows you to give multiple IPs separated by comma or enclosed in quotes and separated by space.

6.12.5 EXAMPLES

```
ipv4 ipset create badguys hash:ip
ipv4 ipset add badguys 1.2.3.4
ipv4 ipset addfile badguys file-with-the-bad-guys-ips.txt
...
ipv4 blacklist full ipset:badguys

# example with multiple IPs
ipv4 ipset create badguys hash:ip
ipv4 ipset add badguys 1.2.3.4,5.6.7.8,9.10.11.12 # << comma separated
ipv4 ipset add badguys "11.22.33.44 55.66.77.88"  # << space separated in quotes
```

ipsets with IP Lists for abuse, malware, attacks, proxies, anonymizers, etc can be downloaded with the contrib/update-ipsets.sh script. Information about the supported ipsets can be found at FireHOL IP Lists

6.12.6 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition

- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- firehol-masquerade(5) - masquerade helper
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- FireHOL IP Lists
- NAT HOWTO
- netfilter flow diagram

6.13 firehol-iptables(5)

6.13.1 NAME

firehol-iptables - include custom iptables commands

6.13.2 SYNOPSIS

iptables *argument...*

ip6tables *argument...*

6.13.3 DESCRIPTION

The `iptables` and `ip6tables` helper commands pass all of their *argument*s to the real iptables(8) or ip6tables(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` or `/sbin/ip6tables` directly in a FireHOL configuration as they will run before FireHOL activates its firewall. This means that the commands are applied to the previously running firewall, not the new firewall, and will be lost when the new firewall is activated.

The `iptables` and `ip6tables` helpers are provided to allow you to hook in commands safely.

6.13.4 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.14 firehol-iptrap(5)

6.14.1 NAME

firehol-iptrap - dynamically put IP addresses in an ipset

6.14.2 SYNOPSIS

{ iptrap | iptrap4 | iptrap6 } *ipset type seconds* [timeout | counters] [method **method**] [*rule-params*] [except [*rule-params*]]...

{ ipuntrap | ipuntrap4 | ipuntrap6 } *ipset type* [timeout | counters] [method **method**] [*rule-params*] [except [*rule-params*]]...

6.14.3 DESCRIPTION

`iptrap` adds the IP addresses of the matching packets to `ipset`.

`ipuntrap` deletes the IP addresses of the matching packets from `ipset`.

Both helpers do not affect the flow of traffic. They do not `ACCEPT`, `REJECT`, `DROP` packets or affect the firewall in any way.

`ipset` is the name of the ipset to use.

`type` selects which of the IP addresses of the matching packets will be used (added or removed from the ipset). `type` can be `src`, `dst`, `src,dst`, `dst,src`, etc. If type is a pair, then the ipset must be an ipset of pairs too.

`seconds` is required by `iptrap` and gives the duration in seconds of the lifetime of each IP address that is added to `ipset`. Every matching packet will refresh this duration for the IP address in the ipset. The Linux kernel will automatically remove the IP from the ipset when this time expires. The user may monitor the remaining time for each IP, by running `ipset list NAME` (where `NAME` is the `ipset` parameter given in the `iptrap` command).

The seconds value `default` will not set any seconds. The ipset default will be used.

A seconds of `0` (zero), writes to the ipset permanently (this is a feature of the ipset command, not the ipset FireHOL helper).

The keywords `timeout` and `counters` are mutually exclusive. `timeout` is the default and means that each IP address every time is matched its timeout will

be refreshed, while `counters` means that its packets and bytes counters will be refreshed. Unfortunately the kernel either re-add the IP in the ipset with the new timeout - but its counters will be lost, or just the counters will be updated, but the timeout will not be refreshed.

`method` is defines the storage method of the underlying ipset. It accepts all the types the ipset commands accepts.

`method` and `type` should match. For example if method is `hash:ip` then method should be either `src` or `dst`. If method is `hash:ip,ip` then method should be either `src,dst` or `dst,src`. If method is `hash:ip,port,ip` method should be `src,src,dst` or `src,dst,dst` or `dst,src,src` or `dst,dst,src`. For more information check the manual page of the ipset command.

The *rule-params* define a set of rule parameters to restrict the traffic that is matched to this helper. See firehol-params(5) for more details.

`except` *rule-params* are used to exclude traffic, i.e. traffic that normally is matched by the first set of *rule-params*, will be excluded if matched by the second.

`iptrap` and `ipuntrap` are hooked on PREROUTING so it is only useful for incoming traffic.

`iptrap` and `ipuntrap` cannot setup both IPv4 and IPv6 traps with one call. The reason is that the `ipset` can either be IPv4 or IPv6.

Both helpers will create the `ipset` specified, if that ipset is not already created by other statements. When the ipset is created by the `iptrap` helper, the ipset will not be reset (emptied) when the firewall is restarted.

The ipset options used when these helpers create ipsets can be controlled with the variable IPTRAP_DEFAULT_IPSET_OPTIONS.

6.14.4 EXAMPLES

```
# Example: mini-IDS
# add to the ipset ‘trap‘ for an hour (3600 seconds) all IPs from all packets
# coming from eth0 and going to tcp/3306 (mysql).
iptrap4 src trap 3600 inface eth0 proto tcp dport 3306 log "TRAPPED HTTP"
# block them
blacklist4 full inface eth0 log "BLOCKED" src ipset:trap except src ipset:whitelist

# Example: ipuntrap
ipuntrap4 src trap inface eth0 src ipset:trap proto tcp dport 80 log "UNTRAPPED HTTP"

# Example: a knock
```

```
# The user will be able to knock at tcp/12345
iptrap4 src knock1 30 inface eth0 proto tcp dport 12345 log "KNOCK STEP 1"
# in 30 seconds knock at tcp/23456
iptrap4 src knock2 60 inface eth0 proto tcp dport 23456 src ipset:knock1 log "KNOCK STEP 2"
# in 60 seconds knock at tcp/34566
iptrap4 src knock3 90 inface eth0 proto tcp dport 34567 src ipset:knock2 log "KNOCK STEP 3"
#
# and in 90 seconds ssh
interface ...
    server ssh accept src ipset:knock3
```

6.14.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.15 firehol-mac(5)

6.15.1 NAME

firehol-mac - ensure source IP and source MAC address match

6.15.2 SYNOPSIS

mac *IP macaddr*

6.15.3 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 firehol-params(5)).

The `mac` helper command DROPs traffic from the *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 LOGGING in firehol-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.

6.15.4 EXAMPLES

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

6.15.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.16 firehol-mark(5)

6.16.1 NAME

firehol-mark - mark traffic for traffic shaping tools

6.16.2 SYNOPSIS

Warning - this manual page is out of date for nightly build/v3 behaviour

mark *value chain rule-params*

6.16.3 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 firehol-connmark(5). There is also a `mark` parameter which allows matching marks within individual rules (see firehol-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(8) 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 firehol-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(8) marks, you will need to disable the Root Path Filtering on the interfaces involved (rp_filter in sysctl).

6.16.4 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
```

6.16.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- firehol-connmark(5) - set a stateful mark on a connection
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- ip(8) - show / manipulate routing, devices, policy routing and tunnels
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- Linux Advanced Routing & Traffic Control HOWTO

6.17 firehol-masquerade(5)

6.17.1 NAME

firehol-masquerade - set up masquerading (NAT) on an interface

6.17.2 SYNOPSIS

masquerade *real-interface rule-params*

masquerade [reverse] *rule-params*

6.17.3 DESCRIPTION

The `masquerade` helper command sets up masquerading on the output of a real network interface (as opposed to a FireHOL `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 given, then the definition's `inface`(s) will be used, if specified.

Unlike most commands, `masquerade` does not inherit its parent definition's *rule-params*, it only honours its own. The `inface` and `outface` parameters should not be used (iptables(8) does not support inface in the POSTROUTING chain and outface will be overwritten by FireHOL using the rules above).

> **Note**
>
> The masquerade always applies to the output of the chosen network interfaces.
>
> FIREHOL_NAT will be turned on automatically (see firehol-variables(5)) and FireHOL will enable packet-forwarding in the kernel.

6.17.4 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.

6.17.5 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
```

6.17.6 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- firehol-nat(5) - nat, snat, dnat, redirect config helpers
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.18 firehol-modifiers(5)

6.18.1 NAME

firehol-modifiers - select IPv4 or IPv6 mode

6.18.2 SYNOPSIS

ipv4 *definition-or-command argument...*

ipv6 *definition-or-command argument...*

6.18.3 DESCRIPTION

Without a modifier, interface and router definitions and commands that come before either will be applied to both IPv4 and IPV6. Commands within an `interface` or `router` assume the same behaviour as the enclosing definition.

When preceded by a modifier, the command or definition can be made to apply to IPv4 or IPv6 only. Note that you cannot create an IPv4 only command within and IPv6 interface or vice-versa.

Examples:

```
interface eth0 myboth src4 192.0.2.0/24 src6 2001:DB8::/24
  ipv4 server http accept
  ipv6 server http accept

ipv4 interface eth0 my4only src 192.0.2.0/24
  server http accept

ipv6 interface eth0 my6only src 2001:DB8::/24
  server http accept
```

Many definitions and commands have explicitly named variants (such as router4, router6, router46) which can be used as shorthand.

6.18.4 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-policy(5) - policy command
- firehol-protection(5) - protection command
- firehol-server(5) - server, route commands
- firehol-client(5) - client command
- firehol-group(5) - group command
- firehol-iptables(5) - iptables helper
- firehol-masquerade(5) - masquerade helper
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.19 firehol-nat(5)

6.19.1 NAME

firehol-nat - set up NAT and port redirections

6.19.2 SYNOPSIS

{ nat to-destination | dnat [to] } *ipaddr*[:*port*] [random] [persistent] [id *id*] [at *chain*] [*rule-params*]

{ nat to-source | snat [to] } *ipaddr*[:*port*] [random] [persistent] [id *id*] [at *chain*] [*rule-params*]

{ nat redirect-to | redirect [to] } *port*[-*range*] [random] [id *id*] [at *chain*] [*rule-params*]

6.19.3 DESCRIPTION

Destination NAT is provided by `nat to-destination` and its synonym `dnat`.

Source NAT is provided by `nat to-source` and its synonym `snat`.

Redirection to a port on the local host is provided by `nat redirect-to` and its synonym `redirect`.

The *port* part of the new address is optional with SNAT and DNAT; if not specified it will not be changed.

When you apply NAT to a packet, the Linux kernel will track the changes it makes, so that when it sees replies the transformation will be applied in the opposite direction. For instance if you changed the destination port of a packet from 80 to 8080, when a reply comes back, its source is set as 80. This means the original sender is not aware a transformation is happening.

This means that NAT is only applied on the first packet of each connection (the nat FireHOL helper always appends `state NEW` to NAT statements).

The NAT helper can be used to setup load balancing. Check the section BALANCING below.

> **Note**
>
> The *rule-params* are used only to determine the traffic that will be matched for NAT in these commands, not to permit traffic to flow.

Applying NAT does not automatically create rules to allow the traffic to pass. You will still need to include client or server entries in an interface or router to allow the traffic.

When using `dnat` or `redirect`, the transformation is in the PREROUTING chain of the NAT table and happens before normal rules are matched, so your client or server rule should match the "modified" traffic.

When using `snat`, the transformation is in the POSTROUTING chain of the NAT table and happens after normal rules are matched, so your client or server rule should match the "unmodified" traffic.

See the netfilter flow diagram if you would like to see how network packets are processed by the kernel in detail.

The `at` keyword allows setting a different chain to attach the rules. For `dnat` and `redirect` the default is PREROUTING, but OUTPUT is also supported. For `snat` the default is POSTROUTING, but INPUT is also supported.

`random` will randomise the port mapping involved, to ensure the ports used are not predictable.

`persistent` is used when the statement is given alternatives (i.e. many destination servers for `dnat`, many source IPs for `snat`, many ports for `redirect`). It will attempt to keep each client on the same nat map. See below for more information about persistence.

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

to-destination *ipaddr*[:*port*] Defines a Destination NAT (DNAT). Commonly thought of as port-forwarding (where packets destined for the firewall with a given port and protocol are sent to a different IP address and possibly port), DNAT is much more flexible in that any number of parameters can be matched before the destination information is rewritten.

ipaddr[:*port*] 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.

ipaddr[:*port*] accepts any `--to-destination` values that iptables(8) accepts. Run `iptables -j DNAT --help` for more information. Multiple *ipaddr*[:*port*] may be specified by separating with spaces and enclosing with quotes.

to-source *ipaddr*[:*port*] Defines a Source NAT (SNAT). SNAT is similar to masquerading but is more efficient for static IP addresses. You can use it to give a public IP address to a host which does not have one behind the firewall. See also firehol-masquerade(5).

ipaddr[:*port*] 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.

ipaddr[:*port*] accepts any `--to-source` values that iptables(8) accepts. Run `iptables -j SNAT --help` for more information. Multiple *ipaddr*[:*port*] may be specified by separating with spaces and enclosing with quotes.

redirect-to *port*[-*range*] Redirect matching traffic to the local machine. This is typically useful if you want to intercept some traffic and process it on the local machine.

port[-*range*] 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.

6.19.4 BALANCING

NAT can balance multiple servers (or IPs in case of `snat`) when a range is specified. This is handled by the kernel.

Example:

```
dnat4 to 10.0.0.1-10.0.0.10 persistent proto tcp dst 1.1.1.1 dport 80
```

In the above example, the Linux kernel will give a `persistent` server to all the sockets of any single client.

FireHOL can also setup balancing using a round-robin or weighted average distribution of requests. However `persistent` cannot be used (the Linux kernel applies persistence on a single NAT statement).

6.19.4.1 Round Robbin distribution

To enable round robin distribution, give multiple `to` values, space separated and enclosed in quotes, or comma separated.

Example:

```
dnat4 to 10.0.0.1,10.0.0.2,10.0.0.3 proto tcp dst 1.1.1.1 port 80
# or
dnat4 to "10.0.0.1 10.0.0.2 10.0.0.3" proto tcp dst 1.1.1.1 port 80
```

Ports can also be given per IP:

```
dnat4 to 10.0.0.1:70,10.0.0.2:80,10.0.0.3:90 proto tcp dst 1.1.1.1 port 80
# or
dnat4 to "10.0.0.1:70 10.0.0.2:80 10.0.0.3:90" proto tcp dst 1.1.1.1 port 80
```

6.19.4.2 Weighted distribution

To enable weighted distribution, append a slash with the weight requested for each entry.

FireHOL adds all the weights given and calculates the percentage of traffic each entry should receive.

Example:

```
dnat4 to 10.0.0.1/30,10.0.0.2/30,10.0.0.3/40 proto tcp dst 1.1.1.1 port 80
# or
dnat4 to "10.0.0.1/30 10.0.0.2/30 10.0.0.3/40" proto tcp dst 1.1.1.1 port 80
# or
dnat4 to 10.0.0.1:70/30,10.0.0.2:80/30,10.0.0.3:90/40 proto tcp dst 1.1.1.1 port 80
# or
dnat4 to "10.0.0.1:70/30 10.0.0.2:80/30 10.0.0.3:90/40" proto tcp dst 1.1.1.1 port 80
```

6.19.4.3 PERSISTENCE

The kernel supports persistence only if the NAT alternatives are contiguous (i.e. dnat to A-B, snat to A-B, redirect to 1000:1010, etc). If they are contiguous, persistence is left at the kernel. FireHOL does nothing.

If the alternatives are not contiguous, FireHOL will use the *recent* iptables module to apply persistence itself.

FireHOL supports mixed mode persistence. For example, you can have something like this:

```
dnat to A-B/70,C-D/20,F/10 persistence id mybalancer
```

The above is a weighted distribution of persistence. Group A-B will get 70%, C-D 20% and server F 10%.

Using the above, FireHOL will apply its persistence to pick one of the groups A-B, or C-D, or F. Once the group has been picked by FireHOL, the kernel will apply persistence within the group, to pick the server that will handle the request.

The FireHOL persistence works like this:

1. A packet is received that should be NATed
2. A lookup is made using the *recent* module to find if it has been seen before. The source IP of packet is looked up.

3. If it has been seen before, the connection is mapped the same way the last time was mapped. The *recent* module is updated too.
4. If it has not been seen before, the connection is mapped using the distribution method specified. The *recent* module is updated too, to be ready for the next connection.

The *recent* module has a few limitations:

1. It has lookup tables. We need one lookup table for each member of of the NAT. FireHOL uses the `id` parameter and the definition of each alternative in the NAT statement to form a name for the lookup table. These lookup tables are persistent to firewall restarts, this is why FireHOL requires from you to set an `id`.

2. It can keep entries in its lookup tables for a given time. FireHOL sets this to 3600 seconds. You can control it by setting `FIREHOL_NAT_PERSISTENCE_SECONDS`.

3. It has a limit on the number of entries in the lookup tables. FireHOL cannot set this. This is kernel module option. The default is 200 entries.

Check this:

~~ # modinfo xt_recent filename: /lib/modules/4.1.12-gentoo/kernel/net/netfilter/xt_recent.ko alias: ip6t_recent alias: ipt_recent license: GPL description: Xtables: "recently-seen" host matching author: Jan Engelhardt jengelh@medozas.de author: Patrick McHardy kaber@trash.net depends: x_tables intree: Y vermagic: 4.1.12-gentoo SMP preempt mod_unload modversions parm: ip_list_tot:number of IPs to remember per list (uint) parm: ip_list_hash_size:size of hash table used to look up IPs (uint) parm: ip_list_perms:permissions on /proc/net/xt_recent/* files (uint) parm: ip_list_uid:default owner of /proc/net/xt_recent/* files (uint) parm: ip_list_gid:default owning group of /proc/net/xt_recent/* files (uint) parm: ip_pkt_list_tot:number of packets per IP address to remember (max. 255) (uint) ~~

```
You have to consult your distribution documentation to set these.
You can find their current values by examining files found in
‘/sys/module/xt_recent/parameters/‘ Unfortunately, these files
are not writable, so to change parameters you have unload and
reload the module (i.e. apply a firewall that does not use the
*recent* module, ‘rmmod xt_recent‘, change the parameter,
re-apply a firewall that uses the *recent* module).

Normaly, you will need a line in ‘/etc/modprobe.d/netfitler.conf‘
like this:

~~~~
options xt_recent ip_list_tot=16384
~~~~
```

```
The number 16384 I used is the max number of unique client IPs
I expect to have per hour (‘FIREHOL_NAT_PERSISTENCE_SECONDS‘)
for this service.

‘ip_list_hash_size‘ is calculated by kernel when the module
is loaded to be bigger and up to twice ‘ip_list_tot‘.
```

Once you have the balancer running, you can find its lookup tables in `/proc/net/xt_recent/`. There you will find files starting with the *id* parameter, one file for every alternative of the NAT rule.

6.19.5 EXAMPLES

```
# Port forwarding HTTP
dnat4 to 192.0.2.2 proto tcp dport 80

# Port forwarding HTTPS on to a different port internally
dnat4 to 192.0.2.2:4443 proto tcp dport 443

# Fix source for traffic leaving the firewall via eth0 with private address
snat4 to 198.51.100.1 outface eth0 src 192.168.0.0/24

# Transparent squid (running on the firewall) for some hosts
redirect4 to 8080 inface eth0 src 198.51.100.0/24 proto tcp dport 80

# Send to 192.0.2.1
#  - all traffic arriving at or passing through the firewall
nat4 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
nat4 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
nat4 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
```

```
nat4 to-destination 192.0.2.1 proto tcp dport 25 dst 203.0.113.1
```

6.19.6 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- firehol-masquerade(5) - masquerade helper
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- NAT HOWTO
- netfilter flow diagram

6.20 firehol-params(5)

6.20.1 NAME

firehol-params - optional rule parameters

6.20.2 SYNOPSIS

Common

{ src | src4 | src6 } [not] *host*

{ dst | dst4 | dst6 } [not] *host*

srctype [not] *type*

dsttype [not] *type*

proto [not] *protocol*

mac [not] *macaddr*

dscp [not] *value class classid*

mark [not] *id*

connmark [not] *id*

custommark [not] *name id*

rawmark [not] *id*

tos [not] *id*

custom "*iptables-options...*"

custom-in "*iptables-options...*"

custom-out "*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

connlog “log text”

log “log text” [level *loglevel*]

loglimit “log text” [level *loglevel*]

Helpers Only

sport *port*

dport *port*

state *state*

ipset [not] name flags [no-counters] [bytes-lt|bytes-eq|bytes-gt|bytes-not-eq *number*] [packets-lt|packets-eq|packets-gt|packets-not-eq *number*] [options *custom-ipset-options*]

limit *limit burst*

connlimit upto|above *limit* [mask *mask*] [saddr|daddr]

hashlimit *name* upto|above *amount/period* [burst *amount*] [mode *{srcip|srcport|dstip|dstport},...*] [srcmask *prefix*] [dstmask *prefix*] [htable-size *buckets*] [htable-max *entries*] [htable-expire *msec*] [htable-gcinterval *msec*]

6.20.3 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. FireHOL automatically sets up the necessary stateful rules to deal with replies in the reverse direction.

All matches should be true for a statement to be executed. However, many matches support multiple values. In this case, at least one of the values must match.

Example:

```
server smtp accept src 1.1.1.1 dst 2.2.2.2
```

In the above example all smtp requests coming in from 1.1.1.1 and going out to smtp server 2.2.2.2 will be matched.

```
server smtp accept src 1.1.1.1 dst 2.2.2.2,3.3.3.3
```

In the above example all smtp requests coming in from 1.1.1.1 and going out to either smtp server 2.2.2.2 or 3.3.3.3 will be matched.

Use the keyword `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.

FireHOL 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).

6.20.4 COMMON

6.20.4.1 src, dst

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.

host can also refer to an ipset, using this syntax: `ipset:NAME`, where NAME is the name of the ipset. The ipset has to be of type `hash:ip` for this match to work. The source IP or the destination IP will be used for the match, depending if the ipset is given as `src` or `dst`.

IPs and ipsets can be mixed together, like this: `src 1.1.1.1,ipset:NAME1,2.2.2.2,ipset:NAME2`

Examples:

```
server4 smtp accept src not 192.0.2.1
server4 smtp accept dst 198.51.100.1
server4 smtp accept src not 192.0.2.1 dst 198.51.100.1
server6 smtp accept src not 2001:DB8:1::/64
server6 smtp accept dst 2001:DB8:2::/64
server6 smtp accept src not 2001:DB8:1::/64 dst 2001:DB8:2::/64
```

When attempting to create rules for both IPv4 and IPv6 it is generally easier to use the `src4`, `src6`, `dst4` and `dst6` pairs:

```
server46 smtp accept src4 192.0.2.1 src6 2001:DB8:1::/64
server46 smtp accept dst4 198.51.100.1 dst6 2001:DB8:2::/64
server46 smtp accept dst4 $d4 dst6 $d6 src4 not $d4 src6 not $s6
```

To keep the rules sane, if one of the 4/6 pair specifies `not`, then so must the other. If you do not want to use both IPv4 and IPv6 addresses, you must specify

the rule as IPv4 or IPv6 only. It is always possible to write a second IPv4 or IPv6 only rule.

6.20.4.2 srctype, dsttype

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"
```

6.20.4.3 proto

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

6.20.4.4 mac

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 `server`s. It refers to the destination of requests for `client`s. 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"
```

6.20.4.5 dscp

Use `dscp` to match the DSCP field on packets. For details on DSCP values and classids, see firehol-dscp(5).

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

6.20.4.6 mark

Use `mark` to match marks set on packets. For details on mark ids, see firehol-mark(5).

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

6.20.4.7 tos

Use `tos` to match the TOS field on packets. For details on TOS ids, see firehol-tos(5).

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

6.20.4.8 custom

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'"
```

6.20.5 ROUTER ONLY

6.20.5.1 inface, outface

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

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

6.20.5.2 physin, physout

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 firehol-interface(5). Examples:

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

6.20.6 INTERFACE ONLY

These parameters match information related to information gathered from the local host. They apply only to outgoing packets and are silently ignored for incoming requests and requests that will be forwarded.

> **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.

6.20.6.1 uid

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 by preventing replies for certain users from being sent:

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

Similarly, 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
```

6.20.6.2 gid

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.

6.20.7 LOGGING

6.20.7.1 connlog

Use `connlog` to log only the first packet of a connection.

6.20.7.2 log, loglimit

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

Logging is controlled using the FIREHOL_LOG_OPTIONS and FIREHOL_LOG_LEVEL environment variables - see firehol-variables(5). `loglimit` additionally honours the FIREHOL_LOG_FREQUENCY and FIREHOL_LOG_BURST variables.

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

6.20.8 HELPERS ONLY PARAMETERS

6.20.8.1 dport, sport

FireHOL also provides `dport`, `sport` and `limit` which are used internally and rarely needed within configuration files.

`dport` and `sport` require an argument *port* which can be a name, number, range (FROM:TO) or a quoted list of ports.

For `dport` *port* specifies the destination port of a request and can be useful when matching traffic to helper commands (such as nat) where there is no implicit port.

For `sport` *port* specifies the source port of a request and can be useful when matching traffic to helper commands (such as nat) where there is no implicit port.

6.20.8.2 limit

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

6.20.8.3 connlimit

`connlimit` matches on the number of connections per IP. It has been added to FireHOL since v3.

saddr matches on source IP. *daddr* matches on destination IP. *mask* groups IPs with the *mask* given *upto* matches when the number of connections is up to the given *limit above* matches when the number of connections above to the given *limit*

The number of connections counted are system wide, not service specific. For example for *saddr*, you cannot connlimit 2 connections for SSH and 4 for SMTP. If you connlimit 2 connections for SSH, then the first 2 connections of a client can be SSH. If a client has already 2 connections to another service, the client will not be able to connect to SSH.

So, `connlimit` can safely be used:

- with *daddr* to limit the connections a server can accept
- with *saddr* to limit the total connections per client to all services.

6.20.8.4 hashlimit

`hashlimit` has been added to FireHOL since v3.

`hashlimit` hashlimit uses hash buckets to express a rate limiting match (like the limit match) for a group of connections using a single iptables rule. Grouping can be done per-hostgroup (source and/or destination address) and/or per-port. It gives you the ability to express "N packets per time quantum per group" or "N bytes per seconds" (see below for some examples).

A hash limit type (`upto`, `above`) and *name* are required.

name The name for the /proc/net/ipt_hashlimit/*name* entry.

`upto` *amount[/second|/minute|/hour|/day]* Match if the rate is below or equal to amount/quantum. It is specified either as a number, with an optional time quantum suffix (the default is 3/hour), or as amountb/second (number of bytes per second).

`above` *amount[/second|/minute|/hour|/day]* Match if the rate is above amount/quantum.

`burst` *amount* Maximum initial number of packets to match: this number gets recharged by one every time the limit specified above is not reached, up to this number; the default is 5. When byte-based rate matching is requested, this option specifies the amount of bytes that can exceed the given rate. This option should be used with caution - if the entry expires, the burst value is reset too.

`mode` *{srcip|srcport|dstip|dstport},...* A comma-separated list of objects to take into consideration. If no `mode` option is given, *srcip,dstport* is assumed.

`srcmask` *prefix* When –hashlimit-mode srcip is used, all source addresses encountered will be grouped according to the given prefix length and the so-created subnet will be subject to hashlimit. prefix must be between (inclusive) 0 and 32. Note that `srcmask` *0* is basically doing the same thing as not specifying srcip for `mode`, but is technically more expensive.

`dstmask` *prefix* Like `srcmask`, but for destination addresses.

`htable-size` *buckets* The number of buckets of the hash table

`htable-max` *entries* Maximum entries in the hash.

`htable-expire` *msec* After how many milliseconds do hash entries expire.

`htable-gcinterval` *msec* How many milliseconds between garbage collection intervals.

Examples:

matching on source host: "1000 packets per second for every host in 192.168.0.0/16"

```
src 192.168.0.0/16 hashlimit mylimit mode srcip upto 1000/sec
```

matching on source port: “100 packets per second for every service of 192.168.1.1”

```
src 192.168.1.1 hashlimit mylimit mode srcport upto 100/sec
```

matching on subnet: “10000 packets per minute for every /28 subnet (groups of 8 addresses) in 10.0.0.0/8”

```
src 10.0.0.8 hashlimit mylimit mask 28 upto 10000/min
```

matching bytes per second: “flows exceeding 512kbyte/s”

```
hashlimit mylimit mode srcip,dstip,srcport,dstport above 512kb/s
```

matching bytes per second: “hosts that exceed 512kbyte/s, but permit up to 1Megabytes without matching”

```
hashlimit mylimit mode dstip above 512kb/s burst 1mb
```

6.20.9 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-server(5) - server, route commands
- firehol-client(5) - client command
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-mark(5) - mark config helper
- firehol-tos(5) - tos config helper
- firehol-dscp(5) - dscp config helper
- firehol-variables(5) - control variables
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.21 firehol-policy(5)

6.21.1 NAME

firehol-policy - set default action for an interface or router

6.21.2 SYNOPSIS

policy *action*

6.21.3 DESCRIPTION

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

The *action* can be any of the actions listed in firehol-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. FireHOL CANNOT be used to create valid "accept by default" firewalls.

6.21.4 EXAMPLE

```
interface eth0 intranet src 192.0.2.0/24
  # I trust this interface absolutely
  policy accept
```

6.21.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.22 firehol-protection(5)

6.22.1 NAME

firehol-protection - add extra protections to a definition

6.22.2 SYNOPSIS

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

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

protection [reverse] { bad-packets | *packet-protection-type* }

protection [reverse] connlimit *connections* [mask *prefix*]

protection [reverse] connrate *rate* [burst *amount*] [srcmask *prefix*] [htable-size *buckets*] [htable-max *entries*] [htable-expire *msec*] [htable-gcinterval *msec*]

6.22.3 DESCRIPTION

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

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

The default rate FireHOL 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 by using a single command and 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.

6.22.4 PACKET PROTECTION TYPES

bad-packets: Drops all the bad packets detected by these rules.

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

See also FIREHOL_DROP_INVALID in firehol-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.

6.22.4.1 EXAMPLES

```
protection bad-packets
```

6.22.5 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).

6.22.5.1 EXAMPLES

```
protection all-floods 90/sec 40
```

6.22.6 CLIENT LIMITING TYPES

These protections were added in v3.

These protections are used to limit the connections client make, per `interface` or `router`.

They support appending `optional rule parameters` to limit their scope to certain clients only.

protection [**reverse**] **connlimit** ***connections*** [**mask** ***prefix***] Allow only a number of connections per client (implemented with `connlimit` with fixed type=*saddr*).

protection [**reverse**] **connrate** ***rate*** [**burst** ***amount***] [**srcmask** ***prefix***] [**htable-size** ***buckets***] [**htable-r**
Allow up to a rate of new connections per client (implemented with `hashlimit` with fixed type=*upto* and mode=*srcip*).

6.22.6.1 EXAMPLES

Limit the number of concurrent connections to 10 per client

```
protection connlimit 10 mask 32
```

Limit the number of concurrent connections to 100 per client class-C and also limit it to 5 for 1.2.3.4

```
protection connlimit 100 mask 24
protection connlimit 5 src 1.2.3.4
```

In the last example above, if you want to give client 1.2.3.4 more connections than all others, you should exclude it from the first connlimit statement, like this:

```
protection connlimit 100 mask 24 src not 1.2.3.4
protection connlimit 200 src 1.2.3.4
```

Limit all clients to 10 concurrect connections and 60 connections/minute

```
protection connlimit 10
protection connrate 60/minute
```

6.22.7 KNOWN ISSUES

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, FireHOL 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.

6.22.8 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.23 firehol-proxy(5)

6.23.1 NAME

firehol-proxy - set up a transparent TCP, HTTP or squid proxy

6.23.2 SYNOPSIS

transparent_proxy *service port user rule-params*

transparent_squid *port user rule-params*

6.23.3 DESCRIPTION

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

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

> **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 firehol-params(5) for more details.

For traffic destined for the firewall host or passing through the firewall, do not use the `outface` parameter 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 that traffic generated by the local user(s) *user*. The `inface`, `outface` and `src` *rule-params* are all ignored for locally-generated traffic.

6.23.4 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"
```

6.23.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- firehol-nat(5) - nat, snat, dnat, redirect config helpers
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.24 firehol-router(5)

6.24.1 NAME

firehol-router - create a router definition

6.24.2 SYNOPSIS

{ router | router46 } *name rule-params*

router4 *name rule-params*

router6 *name rule-params*

6.24.3 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 from those listed in ROUTER SUBCOMMANDS.

> **Note**
>
> Writing `router4` is equivalent to writing `ipv4 router` and ensures the defined router is created only in the IPv4 firewall along with any rules within it.
>
> Writing `router6` is equivalent to writing `ipv6 router` and ensures the defined router is created only in the IPv6 firewall along with any rules within it.
>
> Writing `router46` is equivalent to writing `both router` and ensures the defined router is created in both the IPv4 and IPv6 firewalls. Any rules within it will also be applied to both, unless they specify otherwise.

6.24.4 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 FireHOL interface and router definitions.

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

See firehol-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 firehol.conf(5) for an explanation of ${UNROUTABLE_IPS}.

6.24.5 WORKING WITH ROUTERS

Routers create stateful iptables(8) 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` firehol-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.

6.24.6 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- firehol-modifiers(5) - ipv4/ipv6 selection
- firehol-interface(5) - interface definition
- firehol-iptables(5) - iptables helper
- firehol-masquerade(5) - masquerade helper
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.24.6.1 Router Subcommands

- firehol-policy(5) - policy command
- firehol-protection(5) - protection command
- firehol-server(5) - server, route commands
- firehol-client(5) - client command
- firehol-group(5) - group command
- firehol-tcpmss(5) - tcpmss helper

6.25 firehol-server(5)

6.25.1 NAME

firehol-server - server, route commands: accept requests to a service

6.25.2 SYNOPSIS

{ server | server46 } *service action rule-params*

server4 *service action rule-params*

server6 *service action rule-params*

{ route | route46 } *service action rule-params*

route4 *service action rule-params*

route6 *service action rule-params*

6.25.3 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 FireHOL a server is the destination of a request. Even though this is more complex for some multi-socket services, to FireHOL 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 firehol-services(5). Multiple services may be specified, space delimited in quotes.

The *action* can be any of the actions listed in firehol-actions(5).

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

> **Note**
>
> Writing `server4` is equivalent to writing `ipv4 server` and ensures this subcommand is applied only in the IPv4 firewall rules.

Writing `server6` is equivalent to writing `ipv6 server` and ensures this subcommand is applied only in the IPv6 firewall rules.

Writing `server46` is equivalent to writing `both server` and ensures this subcommand is applied in both the IPv4 and IPv6 firewall rules; it cannot be used as part an interface or router that is IPv4 or IPv6 only.

The default `server` inherits its behaviour from the enclosing interface or router.

The same rules apply to the variations of `route`.

6.25.4 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
```

6.25.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-modifiers(5) - ipv4/ipv6 selection
- firehol-services(5) - services list
- firehol-actions(5) - actions for rules
- firehol-params(5) - optional rule parameters
- firehol-client(5) - client subcommand
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.26 firehol-services(5)

6.26.1 NAME

firehol-services - FireHOL services list

6.26.2 SYNOPSIS

AH all amanda any anystateless apcupsd apcupsdnis aptproxy asterisk

cups custom cvspserver

darkstat daytime dcc dcpp dhcp dhcprelay dhcpv6 dict distcc dns

echo emule eserver ESP

finger ftp

gift giftui gkrellmd GRE

h323 heartbeat http httpalt https hylafax

iax iax2 ICMP icmp ICMPV6 icmpv6 icp ident imap imaps ipsecnatt ipv6error ipv6mld ipv6neigh ipv6router irc isakmp

jabber jabberd

l2tp ldap ldaps lpd

microsoft_ds mms msn msnp ms_ds multicast mysql

netbackup netbios_dgm netbios_ns netbios_ssn nfs nis nntp nntps nrpe ntp nut nxserver

openvpn oracle OSPF

ping pop3 pop3s portmap postgres pptp privoxy

radius radiusold radiusoldproxy radiusproxy rdp rndc rsync rtp

samba sane sip smtp smtps snmp snmptrap socks squid ssh stun submission sunrpc swat syslog

telnet tftp time timestamp tomcat

upnp uucp

vmware vmwareauth vmwareweb vnc

webcache webmin whois

xbox xdmcp

6.26.3 DESCRIPTION

6.26.3.1 service: AH

IPSec Authentication Header (AH) Example:

```
server AH accept
```

Service Type:

- simple

Server Ports:

- 51/any

Client Ports:

- any

Links

- [Wikipedia][WIKI-AH]

Notes

For more information see this Archive of the FreeS/WAN documentation and RFC 2402. [WIKI-AH]: http://en.wikipedia.org/wiki/IPsec#Authentication_Header

6.26.3.2 service: all

Match all traffic Example:

```
server all accept
```

Service Type:

- simple

Server Ports:

- all

Client Ports:

- all

Netfilter Modules

- nf_conntrack_ftp CONFIG_NF_CONNTRACK_FTP

- nf_conntrack_irc CONFIG_NF_CONNTRACK_IRC
- nf_conntrack_sip CONFIG_NF_CONNTRACK_SIP
- nf_conntrack_pptp CONFIG_NF_CONNTRACK_PPTP
- nf_conntrack_proto_gre CONFIG_NF_CT_PROTO_GRE

Netfilter NAT Modules

- nf_nat_ftp CONFIG_NF_NAT_FTP
- nf_nat_irc CONFIG_NF_NAT_IRC
- nf_nat_sip CONFIG_NF_NAT_SIP
- nf_nat_pptp CONFIG_NF_NAT_PPTP
- nf_nat_proto_gre CONFIG_NF_NAT_PROTO_GRE

Notes

Matches all traffic (all protocols, ports, etc.). Note that to provide "connections in one direction with replies" semantics, the kernel connection tracker is still used: this will therefore still not match packets if they are not understood as part of a connection (e.g. some ICMPv6 packets, requests and replies taking different routes, complex protocols with no helper 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:

6.26.3.3 service: amanda

Advanced Maryland Automatic Network Disk Archiver Service Type:

- simple

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

6.26.3.4 service: any

Match all traffic (without modules or indirect) Example:

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

Service Type:

- simple

Server Ports:

- all

Client Ports:

- all

Netfilter Modules

-

Netfilter NAT Modules

-

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 firehol-params(5) this service can match unusual traffic (e.g. GRE - protocol 47).
>
> Note that you have to supply your own name in addition to "any".

6.26.3.5 service: anystateless

Match all traffic statelessly Example:

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

Service Type:

- complex

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 firehol-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.

Note that you have to supply your own name in addition to "anystateless".

6.26.3.6 service: apcupsd

APC UPS Daemon Example:

```
server apcupsd accept
```

Service Type:

- simple

Server Ports:

- tcp/6544

Client Ports:

- default

Links

- [Homepage][HOME-apcupsd]
- [Wikipedia][WIKI-apcupsd]

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 FireHOL service definition using the procedures described in Adding Services in firehol.conf(5). [HOME-apcupsd]: http://www.apcupsd.com [WIKI-apcupsd]: http://en.wikipedia.org/wiki/Apcupsd

6.26.3.7 service: apcupsdnis

APC UPS Daemon Network Information Server Example:

```
server apcupsdnis accept
```

Service Type:

- simple

Server Ports:

- tcp/3551

Client Ports:

- default

Links

- [Homepage][HOME-apcupsdnis]
- [Wikipedia][WIKI-apcupsdnis]

Notes

> This service allows the remote WEB interfaces of APCUPSD, to connect and get information from the server directly connected to the UPS device. [HOME-apcupsdnis]: http://www.apcupsd.com [WIKI-apcupsdnis]: http://en.wikipedia.org/wiki/Apcupsd

6.26.3.8 service: aptproxy

Advanced Packaging Tool Proxy Example:

```
server aptproxy accept
```

Service Type:

- simple

Server Ports:

- tcp/9999

Client Ports:

- default

Links

- Wikipedia

6.26.3.9 service: asterisk

Asterisk PABX Example:

```
server asterisk accept
```

Service Type:

- simple

Server Ports:

- tcp/5038

Client Ports:

- default

Links

- [Homepage][HOME-asterisk]
- [Wikipedia][WIKI-asterisk]

Notes

> This service refers only to the manager interface of asterisk. You should normally enable sip, h323, rtp, etc. at the firewall level, if you enable the relative channel drivers of asterisk. [HOME-asterisk]: http://www.asterisk.org [WIKI-asterisk]: http://en.wikipedia.org/wiki/Asterisk_PBX

6.26.3.10 service: cups

Common UNIX Printing System Example:

```
server cups accept
```

Service Type:

- simple

Server Ports:

- tcp/631 udp/631

Client Ports:

- any

Links

- Homepage
- Wikipedia

6.26.3.11 service: custom

Custom definitions Example:

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

Service Type:

- custom

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 FireHOL 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 Adding Services in firehol.conf(5).

6.26.3.12 service: cvspserver

Concurrent Versions System Example:

```
server cvspserver accept
```

Service Type:

- simple

Server Ports:

- tcp/2401

Client Ports:

- default

Links

- Homepage
- Wikipedia

6.26.3.13 service: darkstat

Darkstat network traffic analyser Example:

```
server darkstat accept
```

Service Type:

- simple

Server Ports:

- tcp/666

Client Ports:

- default

Links

- Homepage

6.26.3.14 service: daytime

Daytime Protocol Example:

```
server daytime accept
```

Service Type:

- simple

Server Ports:

- tcp/13

Client Ports:

- default

Links

- Wikipedia

6.26.3.15 service: dcc

Distributed Checksum Clearinghouse Example:

```
server dcc accept
```

Service Type:

- simple

Server Ports:

- udp/6277

Client Ports:

- default

Links

- [Wikipedia][WIKI-dcc]

Notes

See also this DCC FAQ. [WIKI-dcc]: http://en.wikipedia.org/wiki/Distributed_Checksum_Clearingh

6.26.3.16 service: dcpp

Direct Connect++ P2P Example:

```
server dcpp accept
```

Service Type:

- simple

Server Ports:

- tcp/1412 udp/1412

Client Ports:

- default

Links

- Homepage

6.26.3.17 service: dhcp

Dynamic Host Configuration Protocol Example:

```
server dhcp accept
```

Service Type:

- complex

Server Ports:

- udp/67

Client Ports:

- 68

Links

- [Wikipedia][WIKI-dhcp]

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
```

For example: interface eth0 dhcp policy return server dhcp accept interface eth0 lan src "*mylan*"*dst*"myip" client all accept

This service implicitly sets its client or server to ipv4 mode. [WIKI-dhcp]: http://en.wikipedia.org/wiki/Dhcp

6.26.3.18 service: dhcprelay

DHCP Relay Example:

```
server dhcprelay accept
```

Service Type:

- simple

Server Ports:

- udp/67

Client Ports:

- 67

Links

- [Wikipedia][WIKI-dhcprelay]

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 [WIKI-dhcprelay]: http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol#DHCP_relaying

6.26.3.19 service: dhcpv6

Dynamic Host Configuration Protocol for IPv6 Example:

```
server dhcp accept
client dhcp accept
```

Service Type:

- complex

Server Ports:

- udp/547

Client Ports:

- udp/546

Links

- [Wikipedia][WIKI-dhcpv6]

Notes

> The dhcp service is implemented as stateless rules. It cannot be stateful as the connection tracker will not match a unicast reply to a broadcast request. Further, if you wish to add src/dst rule parameters, you must account for both the broadcast and link-local network prefixes.
>
> Clients broadcast from a link-local address to the multicast address ff02::1:2 on UDP port 547 to find a server. The server sends a unicast reply back to the client which listens on UDP port 546.
>
> For a FireHOL interface, creating a client will allow sending to port 547 and receiving on port 546. Creating a server allows sending to port 546 and receiving on port 547.
>
> Unlike DHCP for IPv4, the source ports to be used are not defined in DHCPv6 - see section 5.2 of RFC3315. Some servers are known to make use of this to send from arbitrary ports, so FireHOL does not assume a source port.
>
> This service implicitly sets its client or server to ipv6 mode. [WIKI-dhcpv6]: https://en.wikipedia.org/wiki/DHCPv6

6.26.3.20 service: dict

Dictionary Server Protocol Example:

```
server dict accept
```

Service Type:

- simple

Server Ports:

- tcp/2628

Client Ports:

- default

Links

- [Wikipedia][WIKI-dict]

Notes

See RFC2229. [WIKI-dict]: http://en.wikipedia.org/wiki/DICT

6.26.3.21 service: distcc

Distributed CC Example:

```
server distcc accept
```

Service Type:

- simple

Server Ports:

- tcp/3632

Client Ports:

- default

Links

- [Homepage][HOME-distcc]
- [Wikipedia][WIKI-distcc]

Notes

For distcc security, please check the distcc security design. [HOME-distcc]: https://code.google.com/p/distcc/ [WIKI-distcc]: http://en.wikipedia.org/wiki/Distcc

6.26.3.22 service: dns

Domain Name System Example:

```
server dns accept
```

Service Type:

- simple

Server Ports:

- udp/53 tcp/53

Client Ports:

- any

Links

- [Wikipedia][WIKI-dns]

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. [WIKI-dns]: http://en.wikipedia.org/wiki/Domain_Name_System

6.26.3.23 service: echo

Echo Protocol Example:

```
server echo accept
```

Service Type:

- simple

Server Ports:

- tcp/7

Client Ports:

- default

Links

- Wikipedia

6.26.3.24 service: emule

eMule (Donkey network client) Example:

```
client emule accept src 192.0.2.1
```

Service Type:

- complex

Server Ports:

- many

Client Ports:

- many

Links

- [Homepage][HOME-emule]

Notes

According to eMule Port Definitions, FireHOL 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 FireHOL firehol-client(5) command to match the eMule client.

Please note that the eMule client is an HTTP client also. [HOME-emule]: http://www.emule-project.com

6.26.3.25 service: eserver

eDonkey network server Example:

```
server eserver accept
```

Service Type:

- simple

Server Ports:

- tcp/4661 udp/4661 udp/4665

Client Ports:

- any

Links

- Wikipedia

6.26.3.26 service: ESP

IPSec Encapsulated Security Payload (ESP) Example:

```
server ESP accept
```

Service Type:

- simple

Server Ports:

- 50/any

Client Ports:

- any

Links

- [Wikipedia][WIKI-ESP]

Notes

For more information see this Archive of the FreeS/WAN documentation RFC 2406. [WIKI-ESP]: http://en.wikipedia.org/wiki/IPsec#Encapsulating_Security_Pay

6.26.3.27 service: finger

Finger Protocol Example:

```
server finger accept
```

Service Type:

- simple

Server Ports:

- tcp/79

Client Ports:

- default

Links

- Wikipedia

6.26.3.28 service: ftp

File Transfer Protocol Example:

```
server ftp accept
```

Service Type:

- simple

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][WIKI-ftp]

Notes

The FTP service matches both active and passive FTP connections. [WIKI-ftp]: http://en.wikipedia.org/wiki/Ftp

6.26.3.29 service: gift

giFT Internet File Transfer Example:

```
server gift accept
```

Service Type:

- simple

Server Ports:

- tcp/4302 tcp/1214 tcp/2182 tcp/2472

Client Ports:

- any

Links

- [Homepage][HOME-gift]
- [Wikipedia][WIKI-gift]

Notes

The gift FireHOL 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. [HOME-gift]: http://gift.sourceforge.net [WIKI-gift]: http://en.wikipedia.org/wiki/GiFT

6.26.3.30 service: giftui

giFT Internet File Transfer User Interface Example:

```
server giftui accept
```

Service Type:

- simple

Server Ports:

- tcp/1213

Client Ports:

- default

Links

- [Homepage][HOME-giftui]
- [Wikipedia][WIKI-giftui]

Notes

This service refers only to the user interface ports offered by giFT. To allow gift accept P2P requests, use the gift. [HOME-giftui]: http://gift.sourceforge.net [WIKI-giftui]: http://en.wikipedia.org/wiki/GiFT

6.26.3.31 service: gkrellmd

GKrellM Daemon Example:

```
server gkrellmd accept
```

Service Type:

- simple

Server Ports:

- tcp/19150

Client Ports:

- default

Links

- Homepage
- Wikipedia

6.26.3.32 service: GRE

Generic Routing Encapsulation Example:

```
server GRE accept
```

Service Type:

- simple

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][WIKI-GRE]

Notes

Protocol No 47.

For more information see RFC RFC 2784. [WIKI-GRE]: http://en.wikipedia.org/wiki/Generic_Routing_Encapsulation

6.26.3.33 service: h323

H.323 VoIP Example:

```
server h323 accept
```

Service Type:

- simple

Server Ports:

- udp/1720 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

6.26.3.34 service: heartbeat

HeartBeat Example:

```
server heartbeat accept
```

Service Type:

- simple

Server Ports:

- udp/690:699

Client Ports:

- default

Links

- [Homepage][HOME-heartbeat]

Notes

> This FireHOL service has been designed such a way that it will allow multiple heartbeat clusters on the same LAN. [HOME-heartbeat]: http://www.linux-ha.org/

6.26.3.35 service: http

Hypertext Transfer Protocol Example:

```
server http accept
```

Service Type:

- simple

Server Ports:

- tcp/80

Client Ports:

- default

Links

- Wikipedia

6.26.3.36 service: httpalt

HTTP alternate port Example:

```
server httpalt accept
```

Service Type:

- simple

Server Ports:

- tcp/8080

Client Ports:

- default

Links

- [Wikipedia][WIKI-httpalt]

Notes

> This port is commonly used by web servers, web proxies and caches where the standard http port is not available or can or should not be used. [WIKI-httpalt]: http://en.wikipedia.org/wiki/Http

6.26.3.37 service: https

Secure Hypertext Transfer Protocol Example:

```
server https accept
```

Service Type:

- simple

Server Ports:

- tcp/443

Client Ports:

- default

Links

- Wikipedia

6.26.3.38 service: hylafax

HylaFAX Example:

```
server hylafax accept
```

Service Type:

- complex

Server Ports:

- many

Client Ports:

- many

Links

- [Homepage][HOME-hylafax]
- [Wikipedia][WIKI-hylafax]

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). [HOME-hylafax]: http://www.hylafax.org/ [WIKI-hylafax]: http://en.wikipedia.org/wiki/Hylafax

6.26.3.39 service: iax

Inter-Asterisk eXchange Example:

```
server iax accept
```

Service Type:

- simple

Server Ports:

- udp/5036

Client Ports:

- default

Links

- [Homepage][HOME-iax]
- [Wikipedia][WIKI-iax]

Notes

This service refers to IAX version 1. There is also iax2. [HOME-iax]: http://www.asterisk.org [WIKI-iax]: http://en.wikipedia.org/wiki/Iax

6.26.3.40 service: iax2

Inter-Asterisk eXchange v2 Example:

```
server iax2 accept
```

Service Type:

- simple

Server Ports:

- udp/5469 udp/4569

Client Ports:

- default

Links

- [Homepage][HOME-iax2]
- [Wikipedia][WIKI-iax2]

Notes

This service refers to IAX version 2. There is also iax. [HOME-iax2]: http://www.asterisk.org [WIKI-iax2]: http://en.wikipedia.org/wiki/Iax

6.26.3.41 service: ICMP

Internet Control Message Protocol Example:

```
server ICMP accept
```

Service Type:

- simple

Server Ports:

- icmp/any

Client Ports:

- any

Links

- Wikipedia

6.26.3.42 service: icmp

Internet Control Message Protocol Alias for ICMP

6.26.3.43 service: ICMPV6

Internet Control Message Protocol v6 Example:

```
server ICMPV6 accept
```

Service Type:

- simple

Server Ports:

- icmpv6/any

Client Ports:

- any

Links

- Wikipedia

6.26.3.44 service: icmpv6

Internet Control Message Protocol v6 Alias for ICMPV6

6.26.3.45 service: icp

Internet Cache Protocol Example:

```
server icp accept
```

Service Type:

- simple

Server Ports:

- udp/3130

Client Ports:

- 3130

Links

- Wikipedia

6.26.3.46 service: ident

Identification Protocol Example:

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

Service Type:

- simple

Server Ports:

- tcp/113

Client Ports:

- default

Links

- Wikipedia

6.26.3.47 service: imap

Internet Message Access Protocol Example:

```
server imap accept
```

Service Type:

- simple

Server Ports:

- tcp/143

Client Ports:

- default

Links

- Wikipedia

6.26.3.48 service: imaps

Secure Internet Message Access Protocol Example:

```
server imaps accept
```

Service Type:

- simple

Server Ports:

- tcp/993

Client Ports:

- default

Links

- Wikipedia

6.26.3.49 service: ipsecnatt

NAT traversal and IPsec Service Type:

- simple

Server Ports:

- udp/4500

Client Ports:

- any

Links

- Wikipedia

6.26.3.50 service: ipv6error

ICMPv6 Error Handling Example:

```
server ipv6error accept
```

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Notes

Not all icmpv6 error types should be 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

Interfaces should always have this set:

```
server ipv6error accept
```

In a router with inface being internal and outface being external the following will meet the recommendations of RFC 4890:

```
server ipv6error accept
```

Do not use: `client ipv6error accept` unless you are controlling traffic on a router interface where outface is the internal destination.

This service implicitly sets its client or server to ipv6 mode.

6.26.3.51 service: ipv6mld

IPv6 Multicast Listener Discovery for IPv6 Example:

```
client ipv6mld accept
```

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-ipv6mld]

Notes

IPv6 uses Multicast Listener Discovery to discover multicast listeners and what they are listening for.

In practice all IPv6 nodes are multicast listeners since multicast is used in the neighbour discovery protocol which replaces ARP in IPv4.

These rules are stateless since reports can happen automatically as well as on query.

Unless muticast snooping is disabled across the network, MLD should be enabled for any clients:

```
client ipv6mld accept
```

MLD should also be enabled as a server on any hosts acting as a router:

```
server ipv6mld accept
```

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

This service implicitly sets its client or server to ipv6 mode. [WIKI-ipv6mld]: https://en.wikipedia.org/wiki/Multicast_Listener_Discovery

6.26.3.52 service: ipv6neigh

IPv6 Neighbour discovery Example:

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

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-ipv6neigh]

Notes

IPv6 uses the Neighbour Discovery Protocol to do automatic configuration of routes and to replace ARP. To allow 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 definition) unless the firewall is for a bridge.

This service implicitly sets its client or server to ipv6 mode. [WIKI-ipv6neigh]: https://en.wikipedia.org/wiki/Neighbor_Discovery_Protocol

6.26.3.53 service: ipv6router

IPv6 Router discovery Example:

```
client ipv6router accept
```

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-ipv6router]

Notes

IPv6 uses the Neighbour Discovery Protocol to do automatic configuration of routes and to replace ARP. To allow 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 definition) unless the firewall is for a bridge.

This service implicitly sets its client or server to ipv6 mode. [WIKI-ipv6router]: https://en.wikipedia.org/wiki/Neighbor_Discovery_Protocol

6.26.3.54 service: irc

Internet Relay Chat Example:

```
server irc accept
```

Service Type:

- simple

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

6.26.3.55 service: isakmp

Internet Security Association and Key Management Protocol (IKE)
Example:

```
server isakmp accept
```

Service Type:

- simple

Server Ports:

- udp/500

Client Ports:

- any

Links

- [Wikipedia][WIKI-isakmp]

Notes

For more information see the Archive of the FreeS/WAN documentation [WIKI-isakmp]: http://en.wikipedia.org/wiki/ISAKMP

6.26.3.56 service: jabber

Extensible Messaging and Presence Protocol Example:

```
server jabber accept
```

Service Type:

- simple

Server Ports:

- tcp/5222 tcp/5223

Client Ports:

- default

Links

- [Wikipedia][WIKI-jabber]

Notes

Allows clear and SSL client-to-server connections. [WIKI-jabber]: http://en.wikipedia.org/wiki/Jabber

6.26.3.57 service: jabberd

Extensible Messaging and Presence Protocol (Server) Example:

```
server jabberd accept
```

Service Type:

- simple

Server Ports:

- tcp/5222 tcp/5223 tcp/5269

Client Ports:

- default

Links

- [Wikipedia][WIKI-jabberd]

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. [WIKI-jabberd]: http://en.wikipedia.org/wiki/Jabber

6.26.3.58 service: l2tp

Layer 2 Tunneling Protocol Service Type:

- simple

Server Ports:

- udp/1701

Client Ports:

- any

Links

- Wikipedia

6.26.3.59 service: ldap

Lightweight Directory Access Protocol Example:

```
server ldap accept
```

Service Type:

- simple

Server Ports:

- tcp/389

Client Ports:

- default

Links

- Wikipedia

6.26.3.60 service: ldaps

Secure Lightweight Directory Access Protocol Example:

```
server ldaps accept
```

Service Type:

- simple

Server Ports:

- tcp/636

Client Ports:

- default

Links

- Wikipedia

6.26.3.61 service: lpd

Line Printer Daemon Protocol Example:

```
server lpd accept
```

Service Type:

- simple

Server Ports:

- tcp/515

Client Ports:

- any

Links

- [Wikipedia][WIKI-lpd]

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). [WIKI-lpd]: http://en.wikipedia.org/wiki/Line_Printer_Daemon_protocol

6.26.3.62 service: microsoft-ds

Direct Hosted (NETBIOS-less) SMB Example:

```
server microsoft_ds accept
```

Service Type:

- simple

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. 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 ports, so that either direct hosted or traditional SMB can be served simultaneously.

Please refer to the netbios_ssn for more information.

6.26.3.63 service: mms

Microsoft Media Server Example:

```
server mms accept
```

Service Type:

- simple

Server Ports:

- tcp/1755 udp/1755

Client Ports:

- default

Netfilter Modules

- See here.

Netfilter NAT Modules

- See here.

Links

- [Wikipedia][WIKI-mms]

Notes

> Microsoft's proprietary network streaming protocol used to transfer unicast data in Windows Media Services (previously called NetShow Services). [WIKI-mms]: http://en.wikipedia.org/wiki/Microsoft_Media_Server

6.26.3.64 service: msn

Microsoft MSN Messenger Service Example:

```
server msn accept
```

Service Type:

- simple

Server Ports:

- tcp/1863 udp/1863

Client Ports:

- default

6.26.3.65 service: msnp

msnp Example:

```
server msnp accept
```

Service Type:

- simple

Server Ports:

- tcp/6891

Client Ports:

- default

6.26.3.66 service: ms-ds

Direct Hosted (NETBIOS-less) SMB Alias for microsoft_ds

6.26.3.67 service: multicast

Multicast Example:

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

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-multicast]

Notes

> The multicast service matches all packets sent to the $MULTICAST_IPS addresses using IGMP or UDP. For IPv4 that means 224.0.0.0/4 and for IPv6 FF00::/16. [WIKI-multicast]: http://en.wikipedia.org/wiki/Multicast

6.26.3.68 service: mysql

MySQL Example:

```
server mysql accept
```

Service Type:

- simple

Server Ports:

- tcp/3306

Client Ports:

- default

Links

- Homepage
- Wikipedia

6.26.3.69 service: netbackup

Veritas NetBackup service Example:

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

Service Type:

- simple

Server Ports:

- tcp/13701 tcp/13711 tcp/13720 tcp/13721 tcp/13724 tcp/13782 tcp/13783

Client Ports:

- any

Links

- [Wikipedia][WIKI-netbackup]

Notes

> To use this service you must define it as both client and server in NetBackup clients and NetBackup servers. [WIKI-netbackup]: http://en.wikipedia.org/wiki/Netbackup

6.26.3.70 service: netbios-dgm

NETBIOS Datagram Distribution Service Example:

```
server netbios_dgm accept
```

Service Type:

- simple

Server Ports:

- udp/138

Client Ports:

- any

Links

- [Wikipedia][WIKI-netbios_dgm]

Notes

> See also the samba.
>
> 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. [WIKI-netbios_dgm]: http://en.wikipedia.org/wiki/Netbios#Datagram_distribution_service

6.26.3.71 service: netbios-ns

NETBIOS Name Service Example:

```
server netbios_ns accept
```

Service Type:

- simple

Server Ports:

- udp/137

Client Ports:

- any

Links

- [Wikipedia][WIKI-netbios_ns]

Notes

> See also the samba. [WIKI-netbios_ns]: http://en.wikipedia.org/wiki/Netbios#Name_service

6.26.3.72 service: netbios-ssn

NETBIOS Session Service Example:

```
server netbios_ssn accept
```

Service Type:

- simple

Server Ports:

- tcp/139

Client Ports:

- default

Links

- [Wikipedia][WIKI-netbios_ssn]

Notes

> See also the samba.
>
> Please keep in mind that newer NETBIOS clients prefer to use port 445 (microsoft_ds) 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 with a tcp-reset message:
>
> server microsoft_ds reject with tcp-reset [WIKI-netbios_ssn]: http://en.wikipedia.org/wiki/Netbios#Session_service

6.26.3.73 service: nfs

Network File System Example:

```
client nfs accept dst 192.0.2.1
```

Service Type:

- complex

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 too. Take care that this is allowed by the running firewall when FireHOL 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 predefined ports, as documented in NFS Howto. If you do this then you will have to define the the ports using the procedure described in Adding Services in firehol.conf(5).

6.26.3.74 service: nis

Network Information Service Example:

```
client nis accept dst 192.0.2.1
```

Service Type:

- complex

Server Ports:

- many

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-nis]

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 too. Take care that this is allowed by the running firewall when FireHOL 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. [WIKI-nis]: http://en.wikipedia.org/wiki/Network_Information_Service

6.26.3.75 service: nntp

Network News Transfer Protocol Example:

```
server nntp accept
```

Service Type:

- simple

Server Ports:

- tcp/119

Client Ports:

- default

Links

- Wikipedia

6.26.3.76 service: nntps

Secure Network News Transfer Protocol Example:

```
server nntps accept
```

Service Type:

- simple

Server Ports:

- tcp/563

Client Ports:

- default

Links

- Wikipedia

6.26.3.77 service: nrpe

Nagios NRPE Service Type:

- simple

Server Ports:

- tcp/5666

Client Ports:

- default

Links

- Wikipedia

6.26.3.78 service: ntp

Network Time Protocol Example:

```
server ntp accept
```

Service Type:

- simple

Server Ports:

- udp/123 tcp/123

Client Ports:

- any

Links

- Wikipedia

6.26.3.79 service: nut

Network UPS Tools Example:

```
server nut accept
```

Service Type:

- simple

Server Ports:

- tcp/3493 udp/3493

Client Ports:

- default

Links

- Homepage

6.26.3.80 service: nxserver

NoMachine NX Server Example:

```
server nxserver accept
```

Service Type:

- simple

Server Ports:

- tcp/5000:5200

Client Ports:

- default

Links

- [Wikipedia][WIKI-nxserver]

Notes

> Default ports used by NX server for connections without encryption.
>
> Note that nxserver also needs the ssh 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 is needed. [WIKI-nxserver]: http://en.wikipedia.org/wiki/NX_Server

6.26.3.81 service: openvpn

OpenVPN Service Type:

- simple

Server Ports:

- tcp/1194 udp/1194

Client Ports:

- default

Links

- Homepage
- Wikipedia

6.26.3.82 service: oracle

Oracle Database Example:

```
server oracle accept
```

Service Type:

- simple

Server Ports:

- tcp/1521

Client Ports:

- default

Links

- Wikipedia

6.26.3.83 service: OSPF

Open Shortest Path First Example:

```
server OSPF accept
```

Service Type:

- simple

Server Ports:

- 89/any

Client Ports:

- any

Links

- Wikipedia

6.26.3.84 service: ping

Ping (ICMP echo) Example:

```
server ping accept
```

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-ping]

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. [WIKI-ping]: http://en.wikipedia.org/wiki/Ping

6.26.3.85 service: pop3

Post Office Protocol Example:

```
server pop3 accept
```

Service Type:

- simple

Server Ports:

- tcp/110

Client Ports:

- default

Links

- Wikipedia

6.26.3.86 service: pop3s

Secure Post Office Protocol Example:

```
server pop3s accept
```

Service Type:

- simple

Server Ports:

- tcp/995

Client Ports:

- default

Links

- Wikipedia

6.26.3.87 service: portmap

Open Network Computing Remote Procedure Call - Port Mapper

Example:

```
server portmap accept
```

Service Type:

- simple

Server Ports:

- udp/111 tcp/111

Client Ports:

- any

Links

- Wikipedia

6.26.3.88 service: postgres

PostgreSQL Example:

```
server postgres accept
```

Service Type:

- simple

Server Ports:

- tcp/5432

Client Ports:

- default

Links

- Wikipedia

6.26.3.89 service: pptp

Point-to-Point Tunneling Protocol Example:

```
server pptp accept
```

Service Type:

- simple

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

6.26.3.90 service: privoxy

Privacy Proxy Example:

```
server privoxy accept
```

Service Type:

- simple

Server Ports:

- tcp/8118

Client Ports:

- default

Links

- Homepage

6.26.3.91 service: radius

Remote Authentication Dial In User Service (RADIUS) Example:

```
server radius accept
```

Service Type:

- simple

Server Ports:

- udp/1812 udp/1813

Client Ports:

- default

Links

- Wikipedia

6.26.3.92 service: radiusold

Remote Authentication Dial In User Service (RADIUS) Example:

```
server radiusold accept
```

Service Type:

- simple

Server Ports:

- udp/1645 udp/1646

Client Ports:

- default

Links

- Wikipedia

6.26.3.93 service: radiusoldproxy

Remote Authentication Dial In User Service (RADIUS) Example:

```
server radiusoldproxy accept
```

Service Type:

- simple

Server Ports:

- udp/1647

Client Ports:

- default

Links

- Wikipedia

6.26.3.94 service: radiusproxy

Remote Authentication Dial In User Service (RADIUS) Example:

```
server radiusproxy accept
```

Service Type:

- simple

Server Ports:

- udp/1814

Client Ports:

- default

Links

- Wikipedia

6.26.3.95 service: rdp

Remote Desktop Protocol Example:

```
server rdp accept
```

Service Type:

- simple

Server Ports:

- tcp/3389

Client Ports:

- default

Links

- [Wikipedia][WIKI-rdp]

Notes

Remote Desktop Protocol is also known also as Terminal Services.
[WIKI-rdp]: http://en.wikipedia.org/wiki/Remote_Desktop_Protocol

6.26.3.96 service: rndc

Remote Name Daemon Control Example:

```
server rndc accept
```

Service Type:

- simple

Server Ports:

- tcp/953

Client Ports:

- default

Links

- Wikipedia

6.26.3.97 service: rsync

rsync protocol Example:

```
server rsync accept
```

Service Type:

- simple

Server Ports:

- tcp/873 udp/873

Client Ports:

- default

Links

- Homepage
- Wikipedia

6.26.3.98 service: rtp

Real-time Transport Protocol Example:

```
server rtp accept
```

Service Type:

- simple

Server Ports:

- udp/10000:20000

Client Ports:

- any

Links

- [Wikipedia][WIKI-rtp]

Notes

> RTP ports are generally all the UDP ports. This definition narrows down RTP ports to UDP 10000 to 20000. [WIKI-rtp]: http://en.wikipedia.org/wiki/Real-time_Transport_Protocol

6.26.3.99 service: samba

Samba Example:

```
server samba accept
```

Service Type:

- complex

Server Ports:

- many

Client Ports:

- default

Links

- [Homepage][HOME-samba]
- [Wikipedia][WIKI-samba]

Notes

The samba service automatically sets all the rules for netbios_ns, netbios_dgm, netbios_ssn and microsoft_ds.

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 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. [HOME-samba]: http://www.samba.org/ [WIKI-samba]: http://en.wikipedia.org/wiki/Samba_(software)

6.26.3.100 service: sane

SANE Scanner service Service Type:

- simple

Server Ports:

- tcp/6566

Client Ports:

- default

Netfilter Modules

- nf_conntrack_sane CONFIG_NF_CONNTRACK_SANE

Netfilter NAT Modules

- N/A

Links

- Homepage

6.26.3.101 service: sip

Session Initiation Protocol Example:

```
server sip accept
```

Service Type:

- simple

Server Ports:

- tcp/5060 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][WIKI-sip]

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. [WIKI-sip]: http://en.wikipedia.org/wiki/Session_Initiation_Protocol

6.26.3.102 service: smtp

Simple Mail Transport Protocol Example:

```
server smtp accept
```

Service Type:

- simple

Server Ports:

- tcp/25

Client Ports:

- default

Links

- Wikipedia

6.26.3.103 service: smtps

Secure Simple Mail Transport Protocol Example:

```
server smtps accept
```

Service Type:

- simple

Server Ports:

- tcp/465

Client Ports:

- default

Links

- Wikipedia

6.26.3.104 service: snmp

Simple Network Management Protocol Example:

```
server snmp accept
```

Service Type:

- simple

Server Ports:

- udp/161

Client Ports:

- default

Links

- Wikipedia

6.26.3.105 service: snmptrap

SNMP Trap Example:

```
server snmptrap accept
```

Service Type:

- simple

Server Ports:

- udp/162

Client Ports:

- any

Links

- [Wikipedia][WIKI-snmptrap]

Notes

An SNMP trap is a notification from an agent to a manager.
[WIKI-snmptrap]: http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol#Trap

6.26.3.106 service: socks

SOCKet Secure Example:

```
server socks accept
```

Service Type:

- simple

Server Ports:

- tcp/1080 udp/1080

Client Ports:

- default

Links

- [Wikipedia][WIKI-socks]

Notes

See also RFC 1928. [WIKI-socks]: http://en.wikipedia.org/wiki/SOCKS

6.26.3.107 service: squid

Squid Web Cache Example:

```
server squid accept
```

Service Type:

- simple

Server Ports:

- tcp/3128

Client Ports:

- default

Links

- Homepage
- Wikipedia

6.26.3.108 service: ssh

Secure Shell Protocol Example:

```
server ssh accept
```

Service Type:

- simple

Server Ports:

- tcp/22

Client Ports:

- default

Links

- Wikipedia

6.26.3.109 service: stun

Session Traversal Utilities for NAT Example:

```
server stun accept
```

Service Type:

- simple

Server Ports:

- udp/3478 udp/3479

Client Ports:

- any

Links

- [Wikipedia][WIKI-stun]

Notes

> STUN is a protocol for assisting devices behind a NAT firewall or router with their packet routing. [WIKI-stun]: http://en.wikipedia.org/wiki/STUN

6.26.3.110 service: submission

SMTP over SSL/TLS submission Example:

```
server submission accept
```

Service Type:

- simple

Server Ports:

- tcp/587

Client Ports:

- default

Links

- [Wikipedia][WIKI-submission]

Notes

> Submission is essentially normal SMTP with an SSL/TLS negotiation. [WIKI-submission]: http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol

6.26.3.111 service: sunrpc

Open Network Computing Remote Procedure Call - Port Mapper
Alias for portmap

6.26.3.112 service: swat

Samba Web Administration Tool Example:

```
server swat accept
```

Service Type:

- simple

Server Ports:

- tcp/901

Client Ports:

- default

Links

- Homepage

6.26.3.113 service: syslog

Syslog Remote Logging Protocol Example:

```
server syslog accept
```

Service Type:

- simple

Server Ports:

- udp/514

Client Ports:

- 514 default

Links

- Wikipedia

6.26.3.114 service: telnet

Telnet Example:

```
server telnet accept
```

Service Type:

- simple

Server Ports:

- tcp/23

Client Ports:

- default

Links

- Wikipedia

6.26.3.115 service: tftp

Trivial File Transfer Protocol Example:

```
server tftp accept
```

Service Type:

- simple

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

6.26.3.116 service: time

Time Protocol Example:

```
server time accept
```

Service Type:

- simple

Server Ports:

- tcp/37 udp/37

Client Ports:

- default

Links

- Wikipedia

6.26.3.117 service: timestamp

ICMP Timestamp Example:

```
server timestamp accept
```

Service Type:

- complex

Server Ports:

- N/A

Client Ports:

- N/A

Links

- [Wikipedia][WIKI-timestamp]

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. [WIKI-timestamp]: http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#Timestamp

6.26.3.118 service: tomcat

HTTP alternate port Alias for httpalt

6.26.3.119 service: upnp

Universal Plug and Play Example:

```
server upnp accept
```

Service Type:

- simple

Server Ports:

- udp/1900 tcp/2869

Client Ports:

- default

Links

- [Homepage][HOME-upnp]
- [Wikipedia][WIKI-upnp]

Notes

For a Linux implementation see: Linux IGD. [HOME-upnp]: http://upnp.sourceforge.net/ [WIKI-upnp]: http://en.wikipedia.org/wiki/Universal_Plug_and_Play

6.26.3.120 service: uucp

Unix-to-Unix Copy Example:

```
server uucp accept
```

Service Type:

- simple

Server Ports:

- tcp/540

Client Ports:

- default

Links

- Wikipedia

6.26.3.121 service: vmware

vmware Example:

```
server vmware accept
```

Service Type:

- simple

Server Ports:

- tcp/902

Client Ports:

- default

Notes

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

6.26.3.122 service: vmwareauth

vmwareauth Example:

```
server vmwareauth accept
```

Service Type:

- simple

Server Ports:

- tcp/903

Client Ports:

- default

Notes

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

6.26.3.123 service: vmwareweb

vmwareweb Example:

```
server vmwareweb accept
```

Service Type:

- simple

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.

6.26.3.124 service: vnc

Virtual Network Computing Example:

```
server vnc accept
```

Service Type:

- simple

Server Ports:

- tcp/5900:5903

Client Ports:

- default

Links

- [Wikipedia][WIKI-vnc]

Notes

VNC is a graphical desktop sharing protocol. [WIKI-vnc]: http://en.wikipedia.org/wiki/Virtual_Network_Computing

6.26.3.125 service: webcache

HTTP alternate port Alias for httpalt

6.26.3.126 service: webmin

Webmin Administration System Example:

```
server webmin accept
```

Service Type:

- simple

Server Ports:

- tcp/10000

Client Ports:

- default

Links

- Homepage

6.26.3.127 service: whois

WHOIS Protocol Example:

```
server whois accept
```

Service Type:

- simple

Server Ports:

- tcp/43

Client Ports:

- default

Links

- Wikipedia

6.26.3.128 service: xbox

Xbox Live Example:

```
client xbox accept
```

Service Type:

- complex

Server Ports:

- many

Client Ports:

- default

Notes

Definition for the Xbox live service.

See program source for contributor details.

6.26.3.129 service: xdmcp

X Display Manager Control Protocol Example:

```
server xdmcp accept
```

Service Type:

- simple

Server Ports:

- udp/177

Client Ports:

- default

Links

- [Wikipedia][WIKI-xdmcp]

Notes

See Gnome Display Manager for a discussion about XDMCP and
firewalls (Gnome Display Manager is a replacement for XDM).
[WIKI-xdmcp]: http://en.wikipedia.org/wiki/X_display_manager_(program_type)#X_Display_M

6.27 firehol-synproxy(5)

6.27.1 NAME

firehol-synproxy - configure synproxy

6.27.2 SYNOPSIS

synproxy *type rules-to-match-request action* [*action options*]

6.27.3 DESCRIPTION

- **type** defines where the SYNPROXY will be attached. It can be `input` (or `in`), `forward` (or `pass`):
  - use `input` (or `in`) when the IP of the real server is an IP assigned to a physical interface of the machine (i.e. the IP is at the firewall itself)
  - use `forward` (or `pass`) when the IP of the real server is routed by the machine (i.e. SYNPROXY should look at the FORWARD chain for this traffic).
- `rules to match request` are FireHOL optional rule parameters and should match the original client REQUEST, before any destination NAT. `inface` and `dst` are required:
  - `inface` is one or more interfaces the REQUEST should be received from
  - `dst` is the IP of the real server, as seen by the client (before any destination NAT)
- **action** defines how SYNPROXY will reach the real server and can be:
  - `accept` to just allow the REQUEST reach the real server without any destination NAT
  - `dnat to IP:PORT` or `dnat to IP1-IP2:PORT1-PORT2` or `dnat to IP` or `dnat to :PORT` to have SYNPROXY reach a server on another machine in a DMZ (different IP and/or PORT compared to the original request). The synproxy statement supports everything supported by the dnat helper (see firehol-nat(5)).

- `redirect to PORT` to divert the request to a port on the firewall itself. The synproxy statement supports everything supported by the redirect helper (see firehol-nat(5)).
- `action CUSTOM_ACTION` to have any other FireHOL action performed on the NEW socket. Use the `action` helper to define custom actions (see firehol-action(5)).
- `action options` are everything supported by FireHOL optional rule parameters that should be applied only on the final action of SYN packet from SYNPROXY to the real server. For example this can be used to append `loglimit "TEXT"` to have something logged by iptables, or limit the concurrent sockets with `connlimit`. Generally, everything you can write on the same line after `server http accept` is also accepted here.

6.27.4 BACKGROUND

SYNPROXY is a TCP SYN packets proxy. It can be used to protect any TCP server (like a web server) from SYN floods and similar DDos attacks.

SYNPROXY is a netfilter module, in the Linux kernel. It is optimized to handle millions of packets per second utilizing all CPUs available without any concurrency locking between the connections.

The net effect of this, is that the real servers will not notice any change during the attack. The valid TCP connections will pass through and served, while the attack will be stopped at the firewall.

For more information on why you should use a SYNPROXY, check these articles:

- http://rhelblog.redhat.com/2014/04/11/mitigate-tcp-syn-flood-attacks-with-red-hat-enterprise-linux-7-beta/
- https://r00t-services.net/knowledgebase/14/Homemade-DDoS-Protection-Using-IPTables-SYNPROXY.html

SYNPROXY is included in the Linux kernels since version 3.12.

6.27.5 HOW IT WORKS

- When a SYNPROXY is used, clients transparently get connected to the SYNPROXY. So the 3-way TCP handshake happens first between the client and the SYNPROXY:
  - Clients send TCP SYN to server A

  - At the firewall, when this packet arrives it is marked as UNTRACKED
  - The UNTRACKED TCP SYN packet is then given to SYNPROXY
  - SYNPROXY gets this and responds (as server A) with TCP SYN+ACK (UNTRACKED)
  - Client responds with TCP ACK (marked as INVALID or UNTRACKED in iptables) which is also given to SYNPROXY

- Once a client has been connected to the SYNPROXY, SYNPROXY automatically initiates a 3-way TCP handshake with the real server, spoofing the SYN packet so that the real server will see that the original client is attempting to connect:
  - SYNPROXY sends TCP SYN to real server A. This is a NEW connection in iptables and happens on the OUTPUT chain. The source IP of the packet is the IP of the client
  - The real server A responds with SYN+ACK to the client
  - SYNPROXY receives this and responds back to the server with ACK. The connection is now marked as ESTABLISHED

- Once the connection has been established, SYNPROXY leaves the traffic flow between the client and the server

So, SYNPROXY can be used for any kind of TCP traffic. It can be used for both unencrypted and encrypted traffic, since it does not interfere with the content itself.

6.27.6 USE CASES

In FireHOL SYNPROXY support is implemented as a helper. The `synproxy` command can be used to set up any number of SYNPROXYs.

FireHOL can set up SYNPROXY for any of these cases:

1. **real server on the firewall itself, on the same port** (e.g. SYNPROXY on port 80, real server on port 80 too).
2. **real server on the firewall itself, on a different port** (e.g. SYNPROXY on port 2200, real server on port 22).
3. **real server on a different machine, without NAT** (e.g. SYNPROXY on a router catching traffic towards IP A, port 80 and real server is at IP A port 80 too).
4. **real server on a different machine, with NAT** (e.g. SYNPROXY on a router catching traffic towards IP A, port 80 and real server is at IP 10.1.1.1 port 90).

5. **screening incoming traffic that should never be sent to a real server** so that traps and dynamic blacklists can be created using traffic that has been screened by SYNPROXY (eliminate "internet noise" and spoofed packets).

So, generally, all cases are covered.

6.27.7 DESIGN

The general guidelines for using `synproxy` in FireHOL, are:

1. **Design your firewall as you would normally do without SYNPROXY**
2. Test that it works without SYNPROXY. Test especially the servers you want to protect. They should be working too
3. Add `synproxy` statements for the servers you want to protect.

To achieve these requirements:

1. The helper will automatically do everything needed for SYNPROXY to:

   - receive the initial SYN packet from the client
   - respond back to the client with SYN+ACK
   - receive the first ACK packet from the client
   - send the initial SYN packet to the server

There are cases where the above are very tricky to achieve. You don't need to match these in your `firehol.conf`. The `synproxy` helper will automatically take care of them. However:

   You do need the allow the flow of traffic between the real server and the real client (as you normally do without a `synproxy`, with a `client`, `server`, or `route` statement in an `interface` or `router` section).

2. The helper will prevent the 3-way TCP handshake between SYNPROXY and the real server interact with other **destination NAT** rules you may have. However for this to happen, make sure you place the `synproxy` statements above any destination NAT rules (`redirect`, `dnat`, `transparent_squid`, `transparent_proxy`, `tproxy`, etc). So:

   SYNPROXY will interact with destination NAT you have in `firehol.conf` **only** if the `synproxy` statements are place below the destination NAT ones.

   You normally do not need to have `synproxy` interact with other destination NAT rules. The `synproxy` helper will handle the destination NAT (`dnat` or `redirect`) it needs by itself.

So **place `synproxy` statements above all destination NAT statements, unless you know what you are doing**.

3. The helper will allow the 3-way TCP handshake between SYNPROXY and the real server interact with **source NAT** rules you may have (`snat`, `masquerade`), since these may be needed to reach the real server.

6.27.8 LIMITATIONS

1. Internally there are matches that are made without taking into account the original `inface`. So, in case different actions have to be taken depending on the interface the request is received, `src` should be added to differentiate the traffic between the two flows.
2. SYNPROXY does not inherit MARKs from the original request packets. It should and it would make matching a lot easier, but it does not. This means that for all packets generated by SYNPROXY, `inface` is lost.
3. FireHOL internally uses a MARK to tag packets send from SYNPROXY to the target server. This is used for 3 reasons:
   - isolate these packets from other destination NAT rules. If they were not isolated from the destination NAT rules, then packets from the SYNPROXY could be matched by a transparent proxy and enter your web proxy. They could be matched by a transparent proxy because they actually originate from the local machine.
   - isolate the same packets from the rest of the packet filtering rules. Without this isolation, most probably the packets will have been dropped since they come from lo.
   - report if orphan synproxy packets are encountered. So packets the FireHOL engine failed to match properly, should appear with a iptables log saying "ORPHAN SYNPROXY->SERVER". If you don't have such logs, everything works as expected.

6.27.9 OTHER OPTIONS

You can change the TCP options used by `synproxy` by setting the variable `FIREHOL_SYNPROXY_OPTIONS`. The default is this:

```
FIREHOL_SYNPROXY_OPTIONS="--sack-perm --timestamp --wscale 7 --mss 1460"
```

If you want to see it in action in the iptables log, then enable logging:

```
FIREHOL_SYNPROXY_LOG=1
```

The default is disabled (0). If you enable it, every step of the 3-way setup between the client and SYNPROXY and the SYN packet of SYNPROXY towards the real server will be logged by iptables.

Using the variable `FIREHOL_CONNTRACK_LOOSE_MATCHING` you can set `net.netfilter.nf_conntrack_tcp_loose`. FireHOL will automatically set this to 0 when a synproxy is set up.

Using the variable `FIREHOL_TCP_TIMESTAMPS` you can set `net.ipv4.tcp_timestamps`. FireHOL will automatically set this to 1 when a synproxy is set up.

Using the variable `FIREHOL_TCP_SYN_COOKIES` you can set `net.ipv4.tcp_syncookies`. FireHOL will automatically set this to 1 when a synproxy is set up.

On a busy server, you are advised to increase the maximum connection tracker entries and its hash table size.

- Using the variable `FIREHOL_CONNTRACK_HASHSIZE` you can set `/sys/module/nf_conntrack/parameters/hashsize`.
- Using the variable `FIREHOL_CONNTRACK_MAX` you can set `net.netfilter.nf_conntrack_max`.

FireHOL will not alter these variables by itself.

6.27.10 SYNPROXY AND DYNAMIC IP

By default the `synproxy` helper requires from you to define a `dst IP` of the server that is to be protected. This is required because the destination IP will be used to match the SYN packet the synproxy sends to the server.

There is however another way that allows the use of synproxy in environments where the IP of the server is unknown (like a dynamic IP DSL):

1. First you need to set `FIREHOL_SYNPROXY_EXCLUDE_OWNER=1`. This will make synproxy not match packets that are generated by the local machine, even if the process that generates them uses your public IP (the packets in order to be matched they will need not have a UID or GID).
2. Next you will need to exclude you lan IPs by adding `src not "${UNROUTABLE_IPS}"` (or any other network you know you use) to the synproxy statement.

6.27.11 EXAMPLES

Protect a web server running on the firewall with IP 1.2.3.4, from clients on eth0:

```
ipv4 synproxy input inface eth0 dst 1.2.3.4 dport 80 accept

interface eth0 wan
    server http accept
```

Protect a web server running on port 90 on the firewall with IP 1.2.3.4, from clients on eth0 that believe the web server is running on port 80:

```
server_myhttp_ports="tcp/90"
client_myhttp_ports="default"

ipv4 synproxy input inface eth0 dst 1.2.3.4 dport 80 redirect to 90

interface eth0 wan
    server myhttp accept # packet filtering works with the real ports
```

Protect a web server running on another machine (5.6.7.8), while the firewall is the router (without NAT):

```
ipv4 synproxy forward inface eth0 dst 5.6.7.8 dport 80 accept

router wan2lan inface eth0 outface eth1
    server http accept dst 5.6.7.8
```

Protect a web server running on another machine in a DMZ (public IP is 1.2.3.4 on eth0, web server IP is 10.1.1.1 on eth1):

```
ipv4 synproxy input inface eth0 \
    dst 1.2.3.4 dport 80 dnat to 10.1.1.1

router wan2lan inface eth0 outface eth1
    server http accept dst 10.1.1.1
```

Note that we used `input` not `forward`, because the firewall has the IP 1.2.3.4 on its eth0 interface. The client request is expected on input.

Protect an array of 10 web servers running on 10 other machines in a DMZ (public IP is 1.2.3.4 on eth0, web servers IPs are 10.1.1.1 to 10.1.1.10 on eth1):

```
ipv4 synproxy input inface eth0 \
    dst 1.2.3.4 dport 80 dnat to 10.1.1.1-10.1.1.10 persistent

router wan2lan inface eth0 outface eth1
    server http accept dst 10.1.1.1-10.1.1.10
```

The above configuration is a load balancer. Requests towards 1.2.3.4 port 80 will be distributed to the 10 web servers with persistence (each client will always see one of them).

Catch all traffic towards SSH port tcp/22 and send it to `TRAP_AND_DROP` as explained in Working With Traps. At the same time, allow SSH on port

tcp/2200 (without altering the ssh server):

```
# definition of action TRAP_AND_DROP
ipv4 action TRAP_AND_DROP sockets_suspects_trap 3600 86400 1 src not "${UNROUTABLE_IPS}" ne

# send ssh traffic to TRAP_AND_DROP
ipv4 synproxy input inface eth0 dst 1.2.3.4 dport 22 action TRAP_AND_DROP

# accept ssh traffic on tcp/2200
ipv4 synproxy input inface eth0 dst 1.2.3.4 dport 2200 redirect to 22

interface eth0 wan
   server ssh accept
```

6.27.12 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- firehol-params(5) - optional rule parameters
- firehol-masquerade(5) - masquerade helper
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- NAT HOWTO
- netfilter flow diagram

6.28 firehol-tcpmss(5)

6.28.1 NAME

firehol-tcpmss - set the MSS of TCP SYN packets for routers

6.28.2 SYNOPSIS

tcpmss { *mss* | auto } [*if-list*]

6.28.3 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` or `interface` definition the MSS will be applied to outgoing traffic on the `outface`(s) of the router or interface.

If used before any router or interface definitions it will be applied to all traffic passing through the firewall. If *if-list* is given, the MSS will be applied only to those interfaces.

6.28.4 EXAMPLES

```
tcpmss auto

tcpmss 500

tcpmss 500 "eth1 eth2 eth3"
```

6.28.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-interface(5) - interface definition
- firehol-router(5) - router definition
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation
- TCPMSS target in the iptables tutorial

6.29 firehol-tos(5)

6.29.1 NAME

firehol-tos - set the Type of Service (TOS) of packets

6.29.2 SYNOPSIS

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

6.29.3 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 firehol-params(5)).

The *value* can be an integer number (decimal or hexadecimal) or one of the descriptive values accepted by iptables(8) (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(8) built in chains belonging to the `mangle` table. The chain names are: INPUT, FORWARD, OUTPUT, PREROUTING and POSTROUTING. These 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 firehol-params(5) for more details.

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

6.29.4 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
```

6.29.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-params(5) - optional rule parameters
- firehol-tosfix(5) - tosfix config helper
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.30 firehol-tosfix(5)

6.30.1 NAME

firehol-tosfix - apply suggested TOS values to packets

6.30.2 SYNOPSIS

tosfix

6.30.3 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 shaping 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.

6.30.4 EXAMPLE

```
tosfix
```

6.30.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-tos(5) - tosfix config helper
- iptables(8) - administration tool for IPv4 firewalls

- ip6tables(8) - administration tool for IPv6 firewalls
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.31 firehol-variables(5)

6.31.1 NAME

firehol-variables - control variables for FireHOL

6.31.2 SYNOPSIS

Defaults:

- DEFAULT_INTERFACE_POLICY="DROP"
- DEFAULT_ROUTER_POLICY="RETURN"
- UNMATCHED_INPUT_POLICY="DROP"
- UNMATCHED_OUTPUT_POLICY="DROP"
- UNMATCHED_FORWARD_POLICY="DROP"
- FIREHOL_INPUT_ACTIVATION_POLICY="ACCEPT"
- FIREHOL_OUTPUT_ACTIVATION_POLICY="ACCEPT"
- FIREHOL_FORWARD_ACTIVATION_POLICY="ACCEPT"
- FIREHOL_LOG_MODE="LOG"
- FIREHOL_LOG_LEVEL=*see notes*
- FIREHOL_LOG_OPTIONS="–log-level warning"
- FIREHOL_LOG_FREQUENCY="1/second"
- FIREHOL_LOG_BURST="5"
- FIREHOL_LOG_PREFIX=""
- FIREHOL_DROP_INVALID="0"
- DEFAULT_CLIENT_PORTS="1000:65535"
- FIREHOL_NAT="0"
- FIREHOL_ROUTING="0"
- FIREHOL_AUTOSAVE=*see notes*
- FIREHOL_AUTOSAVE6=*see notes*
- FIREHOL_LOAD_KERNEL_MODULES="1"
- FIREHOL_TRUST_LOOPBACK="1"
- FIREHOL_DROP_ORPHAN_TCP_ACK_FIN="0"
- FIREHOL_DEBUGGING=""
- WAIT_FOR_IFACE=""

6.31.3 DESCRIPTION

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

All variables may be set in the main FireHOL configuration file `/etc/firehol/firehol.conf`.

Variables which affect the runtime but not the created firewall may also be set as environment variables before running firehol(1). 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.

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

6.31.4 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 firehol-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 FIREHOL_LOG_FREQUENCY.

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 firehol-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 FIREHOL_LOG_FREQUENCY.

Example:

```
DEFAULT_ROUTER_POLICY="REJECT"
```

UNMATCHED_{INPUT|OUTPUT|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 firehol-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 FIREHOL_LOG_FREQUENCY.

Example:

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

FIREHOL_{INPUT|OUTPUT|FORWARD}_ACTIVATION_POLICY 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.

FireHOL defaults all values to `ACCEPT` so that your communications continue to work uninterrupted.

If you wish to prevent connections whilst the new firewall is activating, set these values to `DROP`. This is important to do 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.

Example:

```
FIREHOL_INPUT_ACTIVATION_POLICY="DROP"
FIREHOL_OUTPUT_ACTIVATION_POLICY="DROP"
FIREHOL_FORWARD_ACTIVATION_POLICY="DROP"
```

FIREHOL_LOG_MODE This variable controls method that FireHOL uses for logging.

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

Example:

```
FIREHOL_LOG_MODE="ULOG"
```

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

FIREHOL_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 FireHOL.

Use the following option to choose an iptables(8) log level (alpha or numeric) which is higher than the `-c` of klogd.

Table 4: iptables/klogd levels

iptables	klogd	description
emerg (0)	0	system is unusable

iptables	klogd	description
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

Note

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

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

Example:

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

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

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

Example:

```
FIREHOL_LOG_FREQUENCY="30/minute"
FIREHOL_LOG_BURST="2"
```

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

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

Example:

```
FIREHOL_LOG_PREFIX="FIREHOL:"
```

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

You may be better off using firehol-protection(5) to control matching of `INVALID` packets and others on a per-interface and per-router basis.

> **Note**
>
> Care must be taken on IPv6 interfaces, since ICMPv6 packets such as Neighbour Discovery are not tracked, meaning they are marked as INVALID.

Example:

```
FIREHOL_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, FireHOL finds the exact client ports by querying the kernel options.

Example:

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

FIREHOL_NAT If set to 1, this variable causes FireHOL 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.

Example:

```
FIREHOL_NAT="1"
```

FIREHOL_ROUTING If set to 1, this variable causes FireHOL 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.

Example:

```
FIREHOL_ROUTING="1"
```

FIREHOL_AUTOSAVE; FIREHOL_AUTOSAVE6 These variables specify the file of IPv4/IPv6 rules that will be created when firehol(1) is called with the `save` argument. It may be set as an environment variable.

If the variable is not set, a system-specific value is used which was defined at configure-time. If no value was chosen then the save fails.

Example:

```
FIREHOL_AUTOSAVE="/tmp/firehol-saved-ipv4.txt"
FIREHOL_AUTOSAVE6="/tmp/firehol-saved-ipv6.txt"
```

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

Example:

```
FIREHOL_LOAD_KERNEL_MODULES="0"
```

FIREHOL_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).

Example:

```
FIREHOL_TRUST_LOOPBACK="0"
```

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

In busy environments the iptables(8) 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 FireHOL.

Example:

```
FIREHOL_DROP_ORPHAN_TCP_ACK_FIN="1"
```

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

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

Example:

```
FIREHOL_DEBUGGING="Y"
```

WAIT_FOR_IFACE If set to the name of a network device (e.g. eth0), FireHOL 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).

Example:

```
WAIT_FOR_IFACE="eth0"
```

6.31.5 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- firehol-nat(5) - nat, snat, dnat, redirect helpers
- firehol-actions(5) - actions for rules
- iptables(8) - administration tool for IPv4 firewalls
- ip6tables(8) - administration tool for IPv6 firewalls
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation

6.32 firehol-version(5)

6.32.1 NAME

firehol-version - set version number of configuration file

6.32.2 SYNOPSIS

version 6

6.32.3 DESCRIPTION

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

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

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

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

> **Note**
>
> If you pass version 5 to FireHOL, it will disable IPv6 support and warn you that you must update your configuration.

6.32.4 SEE ALSO

- firehol(1) - FireHOL program
- firehol.conf(5) - FireHOL configuration
- FireHOL Website
- FireHOL Online PDF Manual
- FireHOL Online Documentation