

# Firewall Rules

## Advanced Firewall Rule Writing

# FIREWALL SCRIPTS

---

## 1. Introduction

A “firewall” is a protection system designed to prevent access to your local area network by unauthorised “external” parties, i.e. other users of the internet or another wide area network. It may also limit the degree of access local users have to external network resources. A firewall does not provide a complete security solution; it provides only one element of a fully secure system. Consideration should also be given to the use of user authentication and data encryption. Refer to the IPSec section for further information.

In simple terms, a firewall is a packet filtering system that allows or prevents the transmission of data (in either direction) based on a set of rules. These rules can allow filtering based on the following criteria:

- ◆ source and destination IP addresses
- ◆ source and destination IP port or port ranges
- ◆ type of protocol in use
- ◆ direction of the data (in or out)
- ◆ interface type
- ◆ the route the packet is on
- ◆ if an interface is OOS (out of service)
- ◆ ICMP message type
- ◆ TCP flags (SYN, ACK, URG, RESET, PUSH, FIN)
- ◆ TOS field
- ◆ status of a link and/or data packets on UDP/TCP and ICMP protocols

In addition to providing comprehensive filtering facilities, the routers also allow you to specify rules relating to the logging of information for audit/debugging purposes. This information can be logged to a pseudo-file on the unit called FWLOG.TXT, the EVENTLOG.TXT pseudo-file or to a syslog server. It can also be used to generate SNMP traps.

---

## 2. Firewall Script Syntax

A firewall must be individually configured to match the needs of authorised users and their applications. The rules governing firewall behaviour are defined in a script file called FW.TXT. Each line in this file consists of a label definition, a comment or a filter rule.

### 2.1 Labels

A label definition is a string of up to 12 characters followed by a colon. Labels can only include letters, digits and the underscore character and are used in conjunction with the break option to cause the processing of the script to jump to a new location.

### 2.2 Comments

Any line starting with the hash character (“#”) is deemed to be a comment and ignored.

### 2.3 Filter Rules

The syntax for a filter rule is:

```
[action] [in-out] [options] [tos] [proto] [dnslist] [ip-range] [inspect-state]
```

When the firewall is active, the script is processed one line at a time as each packet is received or transmitted. Even when a packet matches a filter-rule, processing still continues and all the other filter rules are checked until the end of the script is reached. The action taken with respect to a particular packet is that specified by the last matching rule. With the **break** option however the script processing

can be redirected to a new location or to the end of the script if required. The default action that the firewall assigns to a packet is to block. This means that if the packet does not match any of the rules it will be blocked.

The various fields of a script rule are described below:

**[action]**

The **[action]** field may be specified as **block**, **pass**, **pass-ifup**, **dscp**, **vdscp** or **debug**. These operate as follows:

**block:**

The **block** action prevents a packet from being allowed through the firewall. When block is specified an optional field can be included that will cause an ICMP packet to be returned to the interface from which that packet was received. This technique is sometimes used to confuse hackers by having different responses to different packets or for fooling an attacker into thinking a service is not present on a network.

The syntax for specifying the return of an ICMP packet is:

```
"return-icmp" [icmp-type [icmp-code]]
```

where **[icmp\_type]** is a decimal number representing the ICMP type or can be one of the pre-defined text codes listed in the following table:

ICMP type value	ICMP type
1	Unreach
2	Echo
3	Echorep
4	squench
5	redir
6	timex
7	paraprob
8	timest
9	timestrap
10	inforeq
11	inforep
12	maskreg
13	maskrep
14	routerad
15	routersol

The optional **[icmp-code]** field can also be a decimal number representing the ICMP code of the return ICMP packet but if the **[icmp-type]** is **[unreach]** then the code can also be one of the following pre-defined text codes:

ICMP code	Meaning
net-unr	Network unreachable
host-unr	Host unreachable
proto-unr	Protocol unrecognised
port-unr	Port unreachable
needfrag	Needs fragmentation
srcfail	Source route fail

For example:

```
block return-icmp unreach in on ppp 0
```

This rule would cause the unit to return an ICMP Unreachable packet in response to all packets received on PPP 0.

Instead of using the **return-icmp** option to return an ICMP packet, **return-rst** can be used to return a TCP reset packet instead. This would only be applicable for a TCP packet. For example:

```
block return-rst in on eth 0 proto tcp from any to 10.1.2.0/24
```

This would return a TCP reset packet when the firewall receives a TCP packet on the Ethernet interface 0 with destination address 10.1.2.\*.

#### **pass:**

The **pass** action allows packets that match the rule to pass through the firewall.

#### **pass-ifup:**

The **pass-ifup** action allows outbound packets that match the rule to pass through the firewall but only if the link is already active.

#### **debug:**

The **debug** action causes the unit to tag any packets matching the rule for debug. This means that for every matching rule that is encountered from this point in the script onwards, an entry will be placed in the pseudo-file FWLOG.TXT.

#### **dscp:**

The **dscp** action causes any packets matching this rule to have its DSCP value adjusted according to this rule. The DSCP value of a packet indicates the type of service required and is used in conjunction with QOS (Quality of Service) functions. A decimal or hex number must follow the **dscp** keyword to indicate the value that should be set.

#### **vdscp:**

The **vdscp** action is very similar to the **dscp** action as described above in that it adjusts the DSCP value in a packet. The difference however is that this is a virtual change only which means that the actual packet is not changed, and that the packet is processed as if it had the DSCP value as indicated. Like the **dscp** action, a decimal or hex number must follow.

#### **[in-out]**

The **[in-out]** field can be **in** or **out** and is used to specify whether the action applies to inbound or outbound packets. When the field is left blank the rule is applied to any packet irrespective of its direction.

#### **[options]**

The **[options]** field is used to define a number of options that may be applied to packets matching the rule. These are:

#### **log:**

When the **log** option is specified, the unit will place an entry in the FWLOG.TXT file each time it processes a packet that matches the rule. This log will normally detail the rule that was matched along with a summary of the packet contents. If the **log** option is followed by the **body** sub-option, the complete IP packet is entered into the log file so that when the log file is displayed, a more detailed decode of the IP packet is shown.

The **log** field may also be followed by a further sub-option that specifies a different type of log output. This may either be **snmp**, **syslog** or **event**.

If **snmp** is specified an SNMP trap (containing similar information to the normal log entry), is generated when a packet matches the rule.

If **syslog** is specified, a syslog message is sent to the configured syslog manager IP address. This message will contain the same information as that entered into the log file, but in a different format. If the **body** option has been specified, some of the IP packet information is also included. Note that the size of the syslog message is limited to the maximum of 1024 bytes. The syslog message is sent with default priority value of 14, which expands out to facility of USER, and priority INFO.

If **event** is specified the log output will be copied to the EVENTLOG.TXT pseudo-file as well as the FWLOG.TXT file. The event log entry will contain the line number and hit count for the rule that caused the packet to be logged.

**Example:**

Say your local network is on subnet 192.168.\*.\* and you want to block any packets received on PPP 0 that were “pretending” to be on the local network and log the receipt of any such packets to the FWLOG.TXT file and to a syslog server. The filter rule would be constructed as follows:

```
block in log syslog break end on ppp 0 from 192.168.0.0/16 to any
```

**break:**

When the **break** option is specified it must be followed by a user-defined label name or the pre-defined **end** keyword. When followed by a label, the rule processor will “jump” to that label to continue processing. When followed by the **end** keyword rule processing will be terminated and the packet will be treated according to the last matching rule.

**Example:**

```
break ppp_label on ppp 0
# insert rule processing here for packets that are not on ppp 0
break end
ppp_label:
# insert rule processing here for packets that are on ppp 0
```

**on:**

The **on** option is used to specify the interface to which the rule applies and must be followed by a valid interface name. For example, if you were only interested in applying a particular rule to packets being transmitted or received by PPP 0, you would include **on ppp 0** in the rule. Valid interface-names are either **eth n**, **tun n** or **ppp n**, where n is the instance number.

**oneroute:**

The **oneroute** option is used to specify that a rule will only match packets associated with the specified eroute. For example, including the option **oneroute 2** would cause the rule to only match on packets transmitted or received over Eroute 2. The **oneroute** option can be followed with the keyword **any**, which will match if the packet is on any eroute.

**routeto:**

When the **routeto** option is specified and the firewall is processing a received packet, if the rule is the last matching rule, then the packet is tagged as being required to be routed to the specified interface.

For example:

```
pass in break end routeto eth 1 from 10.1.0.0/16 to 1.2.3.4 port=telnet
```

would ensure that all packets from 10.1.\*.\* to 1.2.3.4 on the telnet port are all routed to ETH 1.

**oosed:**

The **oosed** option is used to check the out of service status of an interface. For example, including the option **oosed ppp 1** would cause the rule to match only if interface PPP 1 is out of service.

**[tos]**

The **[tos]** field may be used to specify the Type of Service (TOS) to match. If included, the **[tos]** field consists of the keyword **tos** followed by a decimal or hexadecimal code identifying the TOS to match. For example, to block any inbound packet on PPP 0 with a TOS of 0 you would use a rule such as:

```
block in on ppp 0 tos 0
```

## [proto]

The **[proto]** field is used to specify a protocol to match and consists of the **proto** keyword followed by one of the following protocol identifiers:

Identifier	Meaning
tcp, udp	TCP or UDP packet
udp	UDP packet
tcp	TCP packet
ftp	FTP packets regardless of port number
icmp	ICMP packet
decimal number	decimal number matched to protocol type in IP header

The **[proto]** field is also important when “stateful” inspection is enabled for a rule (using the **[inspect-state]** field), as it describes the protocol to inspect (see **[inspect-state]** below).

## [dnslist]

The **[dnslist]** field is used to match packets that contain DNS names that are in a given dnslist. Following **dnslist** there needs to be a name of a dnslist as specified by the **#dns** command. For example, say we have the following dnslist.

```
#dns gglist www.westermo.co.*,www*.co.nz
```

Then the following firewall rule will block all dns lookups to DNS names matching the above list.

```
block out break end on ppp 1 proto udp dnslist gglist from any to any port=dns
```

## [ip-range]

The **[ip-range]** field is used to describe the range of IP addresses and ports to match upon and may be specified in one of several ways. The basic syntax is:

```
ip-range = "all" | "from" ip-object "to" ip-object [flags] [icmp]
```

where **ip-object** is an IP address specification. Full details of the syntax with examples are given under the heading “Specifying IP Addresses and Address Ranges” below.

## [inspect-state]

The **[inspect-state]** field is used in create rules for “stateful inspection”. This is a powerful option in which the firewall script includes rules that allow the unit to keep track of a TCP/UDP or ICMP session and therefore to only pass packets that match the state of a connection.

Additionally, the **[inspect state]** field can specify an optional OOS (Out Of Service) parameter. This parameter allows the unit to mark any route as being out-of-service for a given period of time in the event that the stateful inspect engine has detected an error.

A full description of how the **[inspect state]** field works is given below under the heading “Stateful Inspection”.

---

### 3. Specifying IP Addresses and Ranges

The **ip-range** field of a firewall script rule identifies the IP address or range of addresses to which the rule applies. The syntax for specifying an IP address range is:

```
ip-range = "all" | "from" ip-object "to" ip-object [ flags ] [ icmp ]
```

where:

```
ip-object = addr [port-comp | port-range]
flags = "flags" { flags } [ !{ flags } ]
icmp = "icmp-type" icmp-type [ "code" decnum ]
addr = "any" | ip-addr[ "/"decnum ] [ "mask" ip-addr | "mask" hexnum ]
port-comp = "port" compare port-num
port-range = "port" port-num "<>" | "><" port-num
ip-addr = IP address in format nnn.nnn.nnn.nnn
decnum = a decimal number
hexnum = a hexadecimal number
compare = "=" | "!=" | "<" | "<=" | ">" | ">="
port-num = service-name | decnum
service-name = "http" | "telnet" | "ftpd" | "ftpcnt" | "pop3" | "ike" | "xot"
| "snmp" | "smtp"
```

In the above syntax definition:

- ◆ items in quotes are keywords
- ◆ items in square brackets are optional
- ◆ items in curly braces are optional and can be repeated
- ◆ the vertical bar symbol ("|") means "or"

An **ip-object** therefore consists of an IP address and an IP port specification, preceded by the keyword **from** or **to** to define whether it is the source or destination address. The most basic form for an **ip-object** is simply an IP address preceded by **from** or **to**. For example, to block all packets destined for address 10.1.2.98 the script rule would be:

```
block out from any to 10.1.2.98
```

An **ip-object** can also be specified using an address mask. This is a way of describing which bits of the IP address are relevant when matching. The script processor supports two formats for specifying masks.

Method 1: The IP address is followed by a forward slash and a decimal number. The decimal number specifies the number of significant bits in the IP address. For example, if you wanted to block all packets in the range 10.1.2.\* the rule would be:

```
block from any to 10.1.2.0/24
```

i.e. only the first 24 bits of the address are significant.

Method 2: This same rule could be described another way using the mask keyword:

```
block from any to 10.1.2.0 mask 255.255.255.0
```

The IP address can also contain either "addr-ppp n" or "addr-eth n" where "n" is the **eth** or **ppp** instance number. In this case the rule is specifying that the IP address is that allocated to the PPP interface or to the Ethernet interface. This is useful in the situation where IP addresses are obtained automatically and therefore are not known by the author of the filtering rules. For example:

```
block in break end on ppp 0 from addr-eth 0 to any
```

---

## 4. Address/Port Translation

One further option that may be used when specifying addresses is to use address translation. The syntax for this is:

```
srcdst = "all | fromto [-> [ip-object] "to" object]
```

I.e. directly after the IP addresses and port are specified an optional "->" can follow indicating that the addresses/ports should be translated. The first source object is optional and is unlikely to be used as it is more normal to translate the destination address. The following example will reroute packets originally destined for 10.10.10.12 to 10.1.2.3:

```
pass out break end from any to 10.10.10.12 -> to 10.1.2.3
```

Additionally to this complete subnets can have NAT applied, the address bits not covered by the subnet mask are taken from the original IP address, so for example to NAT the destination subnet of 192.168.0.0/24 to be 192.168.1.0/24 the firewall rule is:

```
pass out break end from any to 192.168.0.0/24 -> to 192.168.1.0/24
```

---

## 5. Filtering on Port Numbers

Now let us say there is a Telnet server running on a machine on IP address 10.1.2.63 and you wish to make this accessible. Using the filter from the previous example would block all packets to 10.1.2.\*. To make the Telnet server available on 10.1.2.63 we need to add the following line in front of the blocking rule:

```
pass break end from any to 10.1.2.63 port=23
```

So, a packet being sent to the Telnet server (port 23) on IP address 10.1.2.63 will match this rule and further checking is prevented by the break end option.

The above example illustrates the "=" comparison. Other comparison methods supported are:

Symbol	Meaning
!=	not equal
>	greater than
<	less than
<=	less than or equal to
>=	greater than or equal to

It is also possible to specify a port in range or a port out of range with the "><" or "<>" symbols. For example, to pass all packets to addresses in the range 23 to 28, the rule would be specified as:

```
pass break end from any to 10.1.2.63 port 23><28
```

To simplify references to ports, some commonly used port numbers are associated with the pre-defined strings listed in the table below. For instance, in the example above we could substitute the number 23 with the string telnet. This would make the rule:

```
pass break end from any to 10.1.2.63 port=telnet
```

The other port keywords that are defined are:

Keyword	Std. Port	Service
ftpd	20	File Transfer Protocol data port
ftpcnt	21	File Transfer Protocol control port
telnet	23	Telnet server port
smtp	25	SMTP server port
http	80	Web server port

Keyword	Std. Port	Service
pop3	110	Mail server port
sntp	123	NTP server port
ike	500	Source/destination port for IKE key
xot	1998	Destination port for XOT packets

**Note:**

The above service keywords are pre-defined based on “standard” port numbers. It is possible that these may have been defined differently on your system in which case you should use the port numbers explicitly (not the defined names).

## 6. Filtering on TCP Flags

An **ip-object** can be followed by an optional **[flags]** field. This field allows the script to filter based on any combination of TCP flags. The **[flags]** field is used to specify the flags to check and consists of the flags keyword followed by a string specifying the flags themselves. Each letter in this string represents a particular flag type as listed below:

Code	Flag
f	FIN Flag
r	RESET Flag
s	SYN Flag
p	PUSH Flag
u	URG Flag
a	ACK Flag

These flag codes allow the filter to check any combination of flags.

Following on from the previous example, to block packets that have all the flags set you would need to precede the pass rule with the following block rule:

```
block break end from any to 10.1.2.0/24 port=telnet flags frspua
```

Here, the list of flags causes the unit to check that those flags are set. This list may be optionally followed by an exclamation mark (“!”) and a second list of flags that the unit should check for being clear. For example:

```
flags s !a
```

would test for the **s** flag being on and the **a** flag being off with all other flags ignored.

As a further example, let us say we want to allow outward connections from a machine on 10.1.2.33 to a Telnet server. We have to define a filter rule to pass outbound connections and the inbound response packets. Because this is an outbound Telnet service we can make use of the fact that all incoming packets will have their ACK bits set. Only the first packet establishing the connection will have the ACK bit off. The filter rules to do this would look like this:

```
pass out break end from 10.1.2.33 port>1023 to any port=telnet
pass in break end from any port=telnet to 10.1.2.33 port>1023 flags !a
```

The first rule allows the outward connections, and the second rule above allows the response packets back in which the ACK flag must always be on. This second rule will filter out any packets that do not have the ACK flag on. This will bar any attackers from trying to open connections onto the private network by simply specifying the source port as the Telnet port (note that there is a simpler way to achieve the same effect using the inspect state option described below).

---

## 7. Filtering on ICMP Codes

An `ip-object` can be followed by an optional `[icmp]` field. This allows the script to filter packets based on ICMP codes. ICMP packets are normally used to debug and diagnose a network and can be extremely useful. However they form part of a low-level protocol and are frequently exploited by hackers for attacking networks. For this reason most network administrators will want to restrict the use of ICMP packets.

The syntax for including ICMP filtering is:

```
icmp = "icmp-type" icmp-type ["code" decnum]
```

The `icmp-type` can be one of the pre-defined strings listed in the following table or the equivalent decimal numeric value:

ICMP Type	ICMP Value
Unreach	3
Echo	8
Echorep	0
Squench	4
Redir	5
Timex	11
Paramprob	12
Timest	13
Timestrep	14
Inforeq	15
Inforep	16
Maskreq	17
Maskrep	18
Routerad	9
Routersol	10

The following two rules are therefore equivalent:

```
pass in break end on ppp 0 proto icmp from any to 10.1.2.0/24 icmp-type 0
pass in break end on ppp 0 proto icmp from any to 10.1.2.0/24 icmp-type echorep
```

Both of these rules allow echo replies to come in from interface `ppp 0` if they are addressed to our example local network address (10.1.2.\*).

In addition to having a type, ICMP packets also include an ICMP code field. The filter syntax allows for the specification of an optional code field after the ICMP type. When specified the code field must also match. The ICMP code field is specified with a decimal number.

For example, suppose we wish to allow only echo replies and ICMP unreachable type ICMP packets from interface PPP 0. Then the rules would look something like this:

```
pass in break end on ppp 0 proto icmp from any to 10.1.2.0/24 icmp-type echorep
code 0
pass in break end on ppp 0 proto icmp from any to 10.1.2.0/24 icmp-type unreachable
code 0
block in break end on ppp 0 proto icmp
```

The first two rules in this set allow in the ICMP packets that we are willing to permit and the third rule denies all other ICMP packets in from this interface. Now if we ever expect to see echo replies in on `ppp 0` we should allow echo requests out on that interface too. To do that we would have the rule:

```
pass out break end on ppp 0 proto icmp icmp-type echo
```

---

## 8. Stateful Inspection

The routing code stack contains a sophisticated scripted “Stateful Firewall” and “Route Inspection” engine. Stateful inspection is a powerful tool that allows the unit to keep track of a TCP/UDP or ICMP session and match packets based on the state of the connection on which they are being carried. In addition to providing sophisticated Firewall functionality the SF/RI engine also provides a number of facilities for tracking the “health” of routes, marking “dead” routes as being Out Of Service (OOS) and creating rules for the automatic status checking of routes previously marked as OOS (for use in multi-level backup/restore scenarios).

The firewall may be used to place interface into an OOS state and also control how the interfaces return to service. When an interface goes OOS, all routes configured to use that interface will have their route metric set to 16 (the maximum value), meaning that some other route with a lower metric will be selected.

When a firewall stateful inspection rule expires, a decision is made as to whether the traffic being allowed to pass by this rule completed successfully or not. For example, if the stateful rule monitors SYN and FIN packets in both directions for a TCP socket then that rule will expire successfully. However, if SYNs are seen to pass in one direction but no SYNs pass in the other direction, the stateful rule will expire and the unit will tag this as a failure.

The following conditions tag a stateful rule as a failure:

- ◆ packets have only passed in one direction
- ◆ 10 packets have passed in one direction with no return packets (for TCP the packets must also be re-transmits)

All of these features depend upon the stateful inspection capabilities of the Firewall engine which are explained below.

The `[inspect]` field takes the following format:

```
inspect = ["inspect-state" {"oos" {interface-name|logical-name} secs {t=secs}
{c=count} {d=count}} {r="ping"|"tcp",{secs{secs}}} {rd=x} {dt=secs}{stat}]
```

The field can be used on its own or with an optional `oos` (Out Of Service) parameter.

To understand this better let us look at a simple example in which we want to set up a filter to allow all machines on a local network with addresses in the range 10.1.2.\*, to access the Internet on port 80. We will need one rule to filter the outgoing packets and another to filter the responses:

```
pass out break end on ppp 0 from 10.1.2.0/24 to any port=80
pass in break end on ppp 0 from any port=80 to 10.1.2.0/24
```

In this example, the first rule allows outgoing http requests on PPP 0 from any address matching the mask 10.1.2.\* providing that the requests are on port 80 (the normal port address for HTTP requests).

The second rule allows http response packets to be received on PPP 0 providing they are on port 80 and they are addressed to an IP address matching the mask 10.1.2.\*.

However, rule 2 creates a potential security “hole”. The problem with filtering based on the source port is that you can trust the source port only as much as you trust the source machine. For instance an attacker could perform a port scan and provided the source port was set to 80 in each packet, it would get through this filter. Alternatively, on an already compromised system, a “Trojan horse” might be set up listening on port 80.

A more secure firewall can be defined using the “inspect-state” option. The stateful inspection system intelligently creates and manages dynamic filter rules based on the type of connection and the source/destination IP addresses. Applying this to the above example, we can redesign the script to make it both simpler and more effective as described below.

As a consequence of the fact that only the first packet in a TCP handshake will have the SYN flag set, we can use a rule that checks the SYN flag:

```
pass out break end on ppp 0 from 10.1.2.0/24 to any port=80 flags s inspect-state
block in break end on ppp 0
```

The first rule matches only the first outgoing packet because it checks the status of the **s** (SYN) flag and will only pass the packet if the SYN flag is set. At first glance however, it appears that the second rule blocks all inbound packets on PPP 0. Whilst this may be inherently more secure, it would also mean that users on the network would not be able to receive responses to their HTTP requests and would therefore be of little use!

The reason that this is not a problem is that the stateful inspection system creates temporary filter rules based on the outbound traffic. The first of these temporary rules allows the first response packet to pass because it also will have the SYN flag set. However, once the connection is established, a second temporary rule is created that passes inbound or outbound packets if the IP address and port number match those of the initial rule but does not check the SYN flag. It does however monitor the FIN flag so that the system can tell when the connection has been terminated. Once an outbound packet with the FIN flag has been detected along with a FIN/ACK response, the temporary rule ceases to exist and further packets on that IP address/port are blocked.

In the above example, if a local user on address 10.1.2.34 issues an http request to a host on 100.12.2.9, the outward packet would match and be passed. At the same time a temporary filter rule is automatically created by the firewall that will pass inbound packets from IP address 100.12.2.9 that are addressed to 10.1.2.34 port x (where x is the source port used in the original request from 10.1.2.34).

This use of dynamic filters is more secure because both the source and destination IP addresses/ports are checked. In addition, the firewall will automatically check that the correct flags are being used for each stage of the communication.

The potential for a security breach has now been virtually eliminated because even if a hacker could time his attack perfectly he would still have to forge a response packet using the correct source address and port (which was randomly created by the sender of the HTTP request) and also has to target the specific IP address that opened the connection.

Another advantage of “inspect-state” rules is that they are scalable, i.e. many machines can use the rule simultaneously. In our above example for instance many machines on the local network could all browse the Internet and the inspection engine would be dynamically creating precise inward filters as they are required and closing them when they are finished with.

The **inspect-state** option can be used on TCP, UDP protocols and some ICMP packets. The ICMP types that can be used with the “inspect-state” option are “echo”, “timest”, “inforeq” and “maskreq”.

## 8.1 Using [inspect-state] with Flags

As can be seen above, the **inspect-state** option can be used with flags. To illustrate this we will refer back to the earlier example of filtering using flags. It is possible to simplify the script by using the **inspect-state** option. The original script was:

```
pass out break end from 10.1.2.33 port>1023 to any port=telnet
pass in break end from any port=telnet to 10.1.2.33 port>1023 flags a/a
```

Using the inspect state option this can be replaced with a single filter rule:

```
pass out break end from 10.1.2.33 port>1023 to any port=telnet flags s/sa
inspect-state
```

No rule is needed for the return packets because a temporary filter will be created that will only allow inbound packets to pass if they match sessions set up by this stateful inspection rule.

A further point to note about the new rule is that the “flags s/sa” specification ensures that it only matches the first packet in a connection. This is because the first packet in a TCP connection has the

SYN flag on and the ACK flag off and so we only match on that combination. The stateful inspection engine will take care of matching the rest of the packets for this connection.

## 8.2 Using [inspect-state] with ICMP

The [inspect-state] option can be also used with ICMP codes. To allow the use of echo request and to allow echo replies you would have just the one rule:

```
pass out break end on ppp 0 proto icmp icmp-type echo inspect-state
```

The advantage of using **inspect-state**, other than just needing one rule, is that it leads to a more secure firewall. For instance with the **inspect-state** option the echo replies are not allowed in all the time; they will only be allowed in once an echo request has been sent out on that interface. The moment that a valid echo reply comes back (or there is a timeout), echo replies will again be blocked. Furthermore, the full IP address is checked; the IP source and destination must exactly match the IP destination and source of the echo request. If you compare this to the rule to allow echo replies in without using **inspect-state** it would not be possible to check the source address at all and the destination address would match any IP address on our network.

The **inspect-state** option can be used with the following ICMP packet types:

ICMP Type	Matching ICMP Type
Echo	Echo reply
Timest	Timestrep
Inforeq	Inforep
Maskreq	Maskrep

## 8.3 Using [inspect-state] with the Out Of Service Option

The **inspect-state** field can be used with an optional **oos** parameter. This parameter allows the stateful inspect engine to mark as “out of service” any routes that are associated with the specified interface and also to control how and the interfaces are returned to service. Such routes will only be marked as out of service if the specified **oos** option parameters are met. The oos parameter takes the format:

```
oos {interface-name|logical-name} secs {t=secs} {c=count} {d=count}
{r="ping"|"tcp" { ,secs}}
```

where:

**interface-name** or **logical-name** specifies the interface with which the firewall rule is associated, e.g. PPP 1. This can also be a logical interface name which is simply a name that can be created (e.g. “waffle”). When a logical interface name is specified then this name can become oos (out of service) and can be tested in other firewall rules with the **oosed** keyword.

**secs** specifies the length of time in seconds for which the routes that are using the specified interface are marked as out of service.

**{t=secs}** is an optional parameter that specifies the length of time in seconds the unit will wait for a response the packet that matched the rule.

**{c=count}** is an optional parameter that specifies the number of times that the stateful inspection engine must trigger on the rule before the route is marked as out of service.

**{d=count}** is an optional parameter that specifies the number of times that the stateful inspection engine must trigger on the rule before the interface is deactivated (only applies to PPP interfaces).

**{r="ping"|"tcp" { ,secs { ,secs}}}** is an optional parameter that specifies a recovery procedure. When a recovery procedure is specified then after the oos timeout has expired instead of bringing the interface back into service immediately the link is tested first. It is tested by either sending a TCP SYN packet or a ping packet to the address/port that caused the oos condition. The “secs” field specifies the retry time when checking for recovery. Only when the recovery succeeds will interface become in service again.

## UDP Example

```
pass in
pass out
pass out on ppp 1 proto udp from any to 156.15.0.0/16 port=1234 inspect-state
oos ppp 1 300 t=10 c=2 d=2
```

The first two rules simply configure the unit to allow any type of packets to be transmitted or received (the default action of the firewall is to block all traffic).

The third rule is more complex. What it does is to configure the stateful inspection engine to watch for UDP packets (with any source address) being routed via the PPP 1 interface to any address that begins with 156.15 on port 1234. If a hit occurs on this rule but the unit does not detect a reply within 10 seconds (as specified by the **t=** parameter), it will increment an internal counter. When this counter reaches the value set by the **c=** parameter, the stateful inspection engine will mark the PPP 1 interface (and therefore any routes using it), as being out of service for 300 seconds. Similarly, if this counter matches the **d=** parameter the stateful inspection engine will deactivate PPP 1. So in the above example, the stateful inspection engine will mark any routes that use PPP 1 as out of service AND deactivate PPP 1 if no reply is detected within 10 seconds for two packets in a row.

Routes will come back into service when either the specified timeout expires or if there are no other routes with a higher metric in service.

PPP interfaces will be re-activated when either the routes using them are back in service and there is a packet to route and the AODI mode parameter is set to "On".

## TCP Example

```
pass out log break end on ppp 3 proto tcp from any to 192.168.0.1 flags S!A
inspect-state oos 30 t=10 c=2 d=2
pass in
pass out
```

This rule will specifically trace attempts to open a TCP connection on PPP 3 to the 192.168.0.1 IP address and if it fails within 10 seconds twice in a row, will cause the PPP 3 interface to be flagged as out of service (i.e. its metric will be set to 16), for 30 seconds. The optional **d=2** entry will also cause the PPP link to be deactivated. Deactivating the link can be useful in scenarios where renegotiating the PPP connection is likely to resolve the problem. Again, if a matching route with a higher metric has been defined it will be used whilst PPP 3 routes are out of service thus providing a powerful route-backup mechanism.

## 8.4 Using [inspect-state] with the Stat Option

The **inspect-state** option can be used with the **stat** option. The **stat** option will cause this firewall rule to record statistics associated with this firewall rule. Transaction times, counts and errors are recorded under the PPP statistics with this option.

## 8.5 Assigning DSCP Values

When using QOS, packet priorities will be determined by the DSCP values in their TOS fields. These priorities may have already been assigned but if necessary, the router can be configured to assign them by inserting the appropriate rules in the firewall. This is done by using the **dscp** command.

For example:

```
dscp 46 in on eth 0 from 100.100.100.25 to 1.2.3.4 port=4000
```

would set the DSCP value to 46 for almost any type of packet received on ETH 0 from IP address 100.100.100.25 addressed to 1.2.3.4 on port 4000. This allows you to set the DSCP value for almost any type of packet.

As a further example:

```
dscp 46 in on eth 0 proto smtp from any to any
```

would cause outgoing mail traffic to the same top priority queue (46 is by default a very high priority code in the DSCP mappings).

---

## 9. The FWLOG.TXT File

When the log option is specified within a firewall script rule, an entry is created in the FWLOG.TXT pseudo-file each time an IP packet matches the rule. Each log entry will in turn contain the following information:

Parameter	Description
Timestamp	The time when the log entry is created.
Short Description	Usually "FW LOG" but could be "FW DEBUG" for packets that hit rules with the "debug" action set.
Dir	Either "IN" or "OUT". Indicates the direction the packet is travelling.
Line	The line number of the rule that cause the packet to be logged.
Hits	The number of matches for the rule that caused this packet to be logged.
Iface	The Interface the packet was to be transmitted/received on.
Source IP	The source IP address in the IP packet.
Dest. IP	The destination IP address in the IP packet.
ID	The value of the ID field in the IP packet.
TTL	The value of the TTL field in the IP packet.
PROTO	The value of the protocol field in the IP packet. This will be expanded to text as well for the well-known protocols.
Src Port	The value of the source port field in the TCP/UDP header.
Dst Port	The value of the source port field in the TCP/UDP header.
Rule Text	The rule that caused the packet to be logged is also entered into the log file.

In addition, port numbers will be expanded to text pre-defined port numbers.

### 9.1 Log File Examples

Example: log entry without the **body** option:

```
----- 15-8-2002 16:25:50 -----
FW LOG Dir: IN Line: 11 Hits: 1 IFACE: ETH 0
Source IP: 100.100.100.25 Dest IP: 100.100.100.50 ID: 39311 TTL:
128
PROTO: TCP (6)
Src Port: 4232 Dst Port: WEB (80)
pass in log break end on eth 0 proto tcp from 100.100.100.25 to addr-eth
0
flags S/SA inspect-state
-----
```

Example: Log entry with the **body** option:

```
----- 15-8-2002 16:27:56 -----
FW LOG Dir: IN Line: 7 Hits: 1 IFACE: ETH 0
Source IP: 100.100.100.25 Dest IP: 100.100.100.50 ID: 40140 TTL:
128
PROTO: ICMP (1)
block return-icmp echorep log body break end proto icmp icmp-type echo
From REM TO LOCIFACE: ETH 0
 45 IP Ver: 4
 Hdr Len: 20
 00 TOS: Routine
 Delay: Normal
 Throughput: Normal
 Reliability: Normal
 00 3C Length: 60
 9C CC ID: 40140
 00 00 Frag Offset: 0
 Congestion: Normal
 May Fragment
 Last Fragment
 80 TTL: 128
 01 Proto: ICMP
 0C E1 Checksum: 3297
 64 64 64 19 Src IP: 100.100.100.25
 64 64 64 32 Dst IP: 100.100.100.50
 ICMP:
 08 Type: ECHO REQ
 00 Code: 0
 04 5C Checksum: 1116
-----
```

Example: Text included in the EVENTLOG.TXT pseudo-file when the **event** sub-option is specified:

```
16:26:32, 15 Aug 2002, Firewall Log Event: Line: 10, Hits: 3
```

Example: Syslog message where the **body** option is not specified:

```
2002-09-04 16:30:06 User.Info 100.100.100.50 Aug 15 16:31:59 arm.1140
IP Filter -
Filter Rule: block return-icmp unreachable host-unr in log syslog break
end on eth 0 proto tcp from any to 100.100.100.50 port=telnet
Line: 10
Hits: 4
```

Example: Syslog message with the **body** option is specified:

```
2002-08-30 16:19:59 User.Info100.100.100.50 Aug 10 16:21:56 arm.1140
IP Filter - Filter Rule: block return-icmp unreachable port-unr in log
body syslog break end on eth 0 proto tcp from any to 100.100.100.50
port=telnet
Line: 9
Hits: 3
PKT:
Source IP: 100.100.100.25
Dest IP: 100.100.100.50
ID: 13317
TTL: 128
Protocol: TCP
Source Port: 1441
Dest Port: 23
TCP Flags: S
```

---

## 10. Further *[inspect-state]* Examples

Here is a basic **inspect-state** rule with no OOS options:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state
```

This rule will allow TCP packets from 10.1.1.1 to 10.1.2.1 port 23 with the SYN flag set to pass out on PPP 2. Because the **inspect-state** option is used, a stateful rule will also be set up which allows other packets for that TCP socket to also pass.

Next, we will modify the rule to mark an interface OOS if a stateful rule identifies a failed connection:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60
```

The addition of **oos 60** means that if the stateful rule sees a failure, interface PPP 2 will be set OOS for 60 seconds. If no interface is specified after the **oos** keyword, the interface set to OOS will be the one the packet is currently passing on.

It is possible to OOS a different interface by specifying the interface after the **oos** keyword, e.g. **oos ppp 1 60** to put PPP 1 out of service for 60 seconds.

The default time allowed by the stateful rule for a connection to open may be overridden by using the **{t=secs}** option. E.g. To override the default TCP opening time of 60 seconds to 10 seconds:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10
```

A socket will now only have 10 seconds to become established (i.e. exchange SYNs) before the stateful rule will expire and be tagged as a failure.

It is possible to configure the firewall so that the interface is only set to OOS after a number of consecutive failures occur. To do this, use the **{c=count}** option. For example:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10 c=5
```

PPP 2 will now only be set OOS after 5 consecutive failures.

It is possible to deactivate the interface after a number of consecutive failures. This is useful for GPRS interfaces, which may get into a state where the PPP connection appears to be operational, but in fact no packets are passing. In this case, deactivating and reactivating the interface will sometimes fix the problem.

For example:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10 c=5 d=10
```

Now, PPP 2 will be deactivated after 10 consecutive failures.

### Keeping a route out of service and using recovery

It may be that the user wants to keep the interface OOS until he is sure that a future connection will work. To help achieve this, one or more recovery options may be specified. These options get the unit to test connectivity between the unit and the destination IP address of the packet that established the stateful rule. The recovery can be in the form of a PING or a TCP socket connection. An interval between recovery checks must also be specified. For example:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10 c=5 d=10 r=tcp,120
```

Now the interface will be set to OOS for 60 seconds after 5 consecutive failures. After the 60 seconds elapses, the recovery procedure will be initiated. In this example the recovery will consist of TCP connection attempts executed at 2 minute intervals. The interface will remain OOS until the recovery procedure completes successfully. The destination IP address in this case will be 10.1.2.1.

To override the default socket connection time, it is possible to specify an additional recovery option. For example:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10 c=5 d=10 r=tcp,120,10
```

Now, 10 seconds is allowed for each recovery attempt. If the socket connects within that time, the recovery is successful, else the recovery is unsuccessful.

There is also an option `{rd=x}` to disconnect the interface after a recovery attempt completes. This option can be used to deactivate the interface after a recovery failure, success, or either. “x” is a bit-mask indicating the cases where the interface should be deactivated. Bit 0 is used to deactivate the interface after a recovery failure. Bit one is used to deactivate the interface after a recovery success, i.e.

- ◆ rd=1 – means deactivate after a recovery failure
- ◆ rd=2 – means deactivate after a recovery success
- ◆ rd=3 – means deactivate after either recovery success or recovery failure

Extending our firewall rule to include this option gives:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10 c=5 d=10 r=tcp,120,10 rd=3
```

Now the interface will be deactivated after a recovery success or failure.

If the `{rd=x}` option is not used, the interface will remain up until its inactivity timer expires, or it is deactivated by some other means.

The `{dt=secs}` option may be used to indicate that the interface is to remain OOS when it is disconnected, and that it should be reactivated some time after it last disconnected. Recovery procedures will take place after the interface connects.

Extending our firewall rule to include this option gives:

```
pass out break end on PPP 2 proto TCP from 10.1.1.1 to 10.1.2.1 port=telnet
flags S!A inspect-state oos 60 t=10 c=5 d=10 r=tcp,120,10 rd=3 dt=60
```

Now the interface will be reconnected 60 seconds after it disconnects and recovery procedures will start after the interface connects. This option would normally be used with the `{rd=x}` option so that recovery has control over when the interface connects and disconnects.

### Keeping a route out of service and using recovery with a list of addresses

This expands on the functionality above and gives the ability to check connectivity to a range of addresses using a ping. It is possible to specify an address list that the recovery mechanism will ping in turn to see if any respond. This will help ensure that even when 1 or maybe 2 or 3 destinations cant be reached due to an outage on the remote network, the connection will be made available again if at least one of the addresses in the list responds.

The address lists are created using the following syntax:

```
#addrs <list-name> <address1,address2,address3,address4>
```

Address lists can span multiple lines if required, for example:

```
#addrs <list-name> <address1,address2>
#addrs <list-name> <address3,address4>
```

The address list is called using the recovery option pingl. An example firewall rule would be:

```
pass out break end on PPP 1 proto ICMP from 10.1.1.1 to 10.1.2.1 inspect-state oos
60 t=10 c=5 d=10 r=pingl listA ,120,10 rd=3 dt=60
```

This rule would allow pings outbound and on detecting a communication failure it will use pings to a address list named listA. The address list named listA could look like this:

```
#addrs listA 10.1.2.1,10.1.3.1,10.1.4.1,10.1.5.1
#addrs listA 10.1.6.1,10.2.1.1,10.2.2.1
```

This causes the recovery to ping the range of address shown in the list above.

---

## 11. Debugging a Firewall

During the creation and management of firewall scripts, firewall scripts may need debugging to ensure that packets are being processed correctly. To assist in this, a rule with the debug action may be used. If a rule with the debug action is encountered, an entry is made in the FWLOG.TXT pseudo-file each time the packet in question matches a rule from that point on. This gives the administrator the ability to follow a packet through a rule set, and can help determine what, if any, changes are required to the rule set. Rules that specify the debug action would typically be placed near the top of the rule set, so that all matching rules from that point on are entered into the log file.

Entries the FWLOG.TXT file created as the result of a **debug** rule may be identified by the short description "FW\_DEBUG" at the top of the log entry.

An example rule set using a **debug** rule:

```
debug in on ppp 2 proto tcp from any to any port=http
pass in break end proto tcp from any to any port=http flags s/sa inspect state
pass out break end proto udp
```

If placed at the top of the rule set, any packet received on interface PPP 2 to destination port 80 will generate a debug entry in the log file for each subsequent rule that it matches. In the example rule set above, a packet that matched the second rule would also match the first rule, and would therefore create two log entries. The same packet would not match the third rule, and so no log entry would be made for this rule.

Because of the extra processor time required to add all of these additional log entries, debug rules should be removed (or commented out) once the rule set is operating as desired.