

# Personal Firewall 2.3.3



AppGate is a trademark of AppGate Network Security AB.  
Other brands and product names may be trademarks of their  
respective companies or organizations.

The contents of this document are subject to revision and can be  
changed without notice. AppGate Network Security AB shall  
have no liability for any error or damage resulting from the usage  
of this document.



---

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Client Installation	3
2.1.1	Requirements	3
2.1.2	Installation method	3
2.1.3	Upgrade	4
2.1.4	Generating installation packages with custom rule sets	4
2.2	AppGate Security Server	4
2.2.1	Installing a firewall policy	4
2.2.2	Access rule configuration (optional)	4
2.2.3	AppGate Client configuration	5
2.2.4	Using Multiple AppGate servers	5
2.3	Using Personal Firewall with Policy Manager	5
2.3.1	Installation and configuration	5
<b>3</b>	<b>Using the firewall</b>	<b>7</b>
3.1	Rule Sets	7
3.1.1	AppGate Server controlled rule set	7
3.1.2	Policy Manager controlled rule set	7
3.1.3	Default rule set	7
3.2	Counting Matched Rules	9
3.3	Rule Numbering	9
3.4	Using Multiple Firewalls	10
<b>4</b>	<b>Rules</b>	<b>11</b>
4.1	What Is Filtered	11
4.2	Syntax	11
4.2.1	Rule set preamble	11
4.2.2	Rule groups	12
4.2.3	Comments	12
4.3	High-level Rules	12
4.3.1	Some notes on High-level rules	13
4.4	Low-level Rules	13
4.4.1	Actions (pass, block, setflag, clearflag)	14
4.4.2	Direction (in, out, inout)	14
4.4.3	Logging (log, log-verbose, log-crit)	14
4.4.4	Address (from, to)	15
4.4.5	Protocol	16
4.4.6	IP Options (with-ipopts)	16
4.4.7	Short IP fragments (short-frag)	16
4.4.8	Flags (hasflag, not-hasflag)	17
4.4.9	State (with-state, keep-state, hastcpstate, hasudpstate)	17
4.5	Examples	18
4.5.1	A simple rule set	18
4.5.2	ssh	18
4.5.3	NetBIOS share access with browsing	19

---

4.5.4 FTP .....	19
<b>Appendix A Default Rule Set .....</b>	<b>20</b>
<b>Appendix B High-Level Rules.....</b>	<b>21</b>
<b>Appendix C Low-level Rule Syntax.....</b>	<b>26</b>
<b>Appendix D More About States .....</b>	<b>27</b>
<b>Appendix E ICMP Types and Codes .....</b>	<b>31</b>
<b>Appendix F Registry Settings.....</b>	<b>33</b>

# 1 Introduction

The AppGate Personal Firewall is a network-level firewall designed to be used together with the AppGate Security Server or the AppGate Policy Manager. It is a stateful IP filter that protects the PCs from network based attacks by ensuring that the clients can connect only to specific IP addresses on specific ports and by blocking unwanted connections to the user's workstation. The firewall checks all inbound and outbound traffic according to a well-defined firewall policy.

The AppGate Personal Firewall can use different policies depending on the circumstances. When used with the AppGate Security Server, the Personal Firewall can be configured to only allow network communication to the AppGate server while the user is connected. The AppGate Policy Manager can provide the Personal Firewall with different rule sets depending on e.g. where the client is located. The administrator can also configure a default rule set which is used when the client is neither in contact with an AppGate Security Server nor an AppGate Policy Manager.

The firewall rules are applied to all network adapters on the client machine of either dial-up or Ethernet type. Filters may be set up on the following characteristics of traffic:

- Direction of packets (incoming/outgoing)
- Source and/or destination IP addresses
- IP protocol type (e.g. TCP/UDP/ICMP)
- Source and/or destination TCP/UDP ports
- ICMP type
- Bad packets (fragmented packets, malformed packets, etc.)

The firewall keeps state for established connections which means that TCP, UDP and ICMP packets belonging to a session that has already been approved can be passed automatically. Whether a session is allowed or not may also depend on already established sessions, for example can a rule set allow FTP data connections only when a corresponding FTP control connection is active.



## 2 Installation

### 2.1 Client Installation

---

Please read this entire section before beginning the installation of the firewall.

#### 2.1.1 Requirements

The following operating systems are supported by the AppGate Personal Firewall:

- Windows 2000 Service Pack 2 or higher
- Windows XP (x86 or X64 architectures)
- Windows 2003 Server
- 32-bit Windows Vista

Windows 98, ME and NT4 are not longer supported, but an older version of the AppGate Personal Firewall can be used for those systems.

#### 2.1.2 Installation method

To start the installation, log in as Administrator and run 'agpfw.exe' on the Personal Firewall CD. The 'agpfw.exe' file must reside on a local disk (i.e not a network share) when the installation is started.

For the installation to start you must read and accept the End-User License Agreement.

During the installation it is possible to choose the directory in which the Personal Firewall should be installed or accept the default (recommended).

It is possible to do a silent (unattended) installation by providing the "/S" switch to agpfw.exe.

#### **Note for Windows 2000, XP and Vista users**

Windows 2000, XP and Vista use digital signatures on device drivers. Drivers may only be signed by Microsoft. There is a local security policy parameter that controls whether unsigned drivers may be installed or not. This parameter may only be changed by an administrator. The parameter has three values:

- Ignore - Install all files regardless of signature.
- Warn - Display a message before installing an unsigned file.
- Block - Prevent the installation of unsigned files.

The default setting is 'Warn'. If 'Warn' is set, the user will be warned once for each network adapter that is installed on the computer when installing the AppGate Personal Firewall. If 'Block' is set, the firewall may not be installed at all.

To install the Personal Firewall, click 'Continue Anyway' each time the dialog box appears.



**WARNING!** For all version of Windows the user will be prompted to reboot the computer after the installation of the Personal Firewall is complete. A reboot is **REQUIRED** in order to properly install the firewall. Network connections and network disks may be rendered unusable following the installation before a reboot is performed. Therefore, it is recommended to reboot immediately following the installation.

### 2.1.3 Upgrade

If you run an older version of the AppGate Personal Firewall then you must uninstall this first before the new version is installed. A backup of the rules file must be made manually if it has been changed and these changes should be kept.

### 2.1.4 Generating installation packages with custom rule sets

To facilitate deployment throughout an organization it is possible to generate custom installation packages where the default rule set is replaced with a custom rule set.

These packages are built by installing the “AppGate Personal Firewall Packaging Tool” on a windows computer. This is done by running “agpfw-pkgtool.exe” from the cd.

Once the packaging tool has been installed the following steps are performed to create a custom installation package:

1. Create a rules file using any text editor and save it.
2. Run the “buildpkg.bat” script in a command window with the following arguments:  
`buildpkg.bat -d <rules file> -o <package file>`  
Where “rules file” is the name of the rules file that should be packaged with the installation package and “package file” is the name of the resulting package file.

## 2.2 AppGate Security Server

---

### 2.2.1 Installing a firewall policy

An AppGate Server license is required in order for this functionality to be available. (Please see the AppGate Server documentation for how to enter server licenses, if required.)

In order to configure the AppGate Server to download a set of rules to its clients, the rule set is defined on the server via the AppGate Administration Console. Different rule sets can be sent to different users depending on user name and role.

The rules in the matching rule set will be propagated to clients that have the Personal Firewall installed and enabled and the policy enabled when they are connected to this AppGate Server.

Please refer to the *AppGate Administration Guide* for further information on access rules and groups.

### 2.2.2 Access rule configuration (optional)

Users can be forced to have the Personal Firewall installed on their machine in order to be able to access resources protected by the AppGate Server. This adds an extra layer of security to the client machine, and when used in conjunction with a strong set of rules, can provide an extremely high level of security.

Services may now be associated with the access rule requiring the Personal Firewall to be active. Note that if a user has no available roles, he/she cannot log into the system.

Please refer to the *AppGate Administration Guide* for further information on access rules and groups.

### 2.2.3 AppGate Client configuration

When a user connects to an AppGate Server that is configured to use the Personal Firewall capability, a *server controlled rule set* will be fetched from the server and be active during the connection. Server controlled rule sets are uneditable by the client and will not appear in any configuration file readable by the user. If the firewall is installed the client will attempt to fetch the firewall rule set from the server after a role has been selected by the user.

It is possible to disable the personal firewall by manually editing the `agclient.conf` file. The `dont_use_pfw` parameter can be inserted in the global parameters section outside of the server blocks.

NOTE: The user may be FORCED to use the Personal Firewall in order to be able to access certain services through the use of an access rule, as described in section 2.2.2.

### 2.2.4 Using Multiple AppGate servers

If an AppGate Client is connected to more than one AppGate Server that has firewall rules, the rules from the different servers will be concatenated (in the same order as the servers were connected) by the AppGate Client before they are installed in the firewall. Therefore, an AppGate Server administrator that wants to be sure that a user cannot be connected to another AppGate Server at the same time must add an explicit 'block' rule at the end of the rule set of the first server. This behavior may change in future versions.

The behaviour of the 'allow-appgate' high-level rule in pfw version prior to 2.3.0 only allowed the already established ssh session to the AppGate server. That meant that it was not possible to open new connections to the AppGate server, nor did roaming work if the 'allow-appgate' rule was the only way to access the AppGate server. With pfw 2.3.0 or later and AppGate Client 8.1 or later, the 'allow-appgate' high-level rule will allow roaming and multiple connections to the same AppGate server provided that the client connects directly to the AppGate server. With an earlier version of pfw, an earlier version of the AppGate Client or when the AppGate Client connects via a proxy the old behavior is used.

## 2.3 Using Personal Firewall with Policy Manager

---

### 2.3.1 Installation and configuration

The easiest way to install AppGate Personal Firewall when it is to be used with the Policy Manager is to use an installation package created by the Policy Manager. This will install Personal Firewall preconfigured with the correct policy.

To configure an AppGate Personal Firewall to use a Policy Manager without using a special crafted installation package generated by the Policy Manager, the policy file must be manually installed in the file specified by the registry entry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\AppGate\pfw\PolicyFile
```

which normally points to the file:

`%WINDIR%\AppGate Personal Firewall\policy.cfg`

Although the configuration file is in a plain text format, it should not be created or edited by hand but by the Policy Manager, please see that documentation for details.

The client must be able to communicate with the Policy Manager using UDP on port 38203. If any other personal firewalls are installed on the client, they must allow this communication.

## 3 Using the firewall

This section provides a basic explanation of how the Personal Firewall works, and how it is used.

### 3.1 Rule Sets

---

There are three rule sets available in the Personal Firewall. The *default rule set* which is active when no connection to an AppGate Server exists, the *server controlled rule set* which is activated by AppGate Client or AppGate Connect when the user connects to an AppGate Server and the *Policy Manager controlled rule set*. Rule sets controlled by an AppGate Server or a Policy Manager are uneditable by the client and will not appear in any configuration file readable by the user.

#### 3.1.1 AppGate Server controlled rule set

As described previously, the server controlled rule sets are managed via the administration console. It can be edited to comply with the system owners security policy and different rule sets can be chosen depending on user name and role. This rule set is received and installed by the AppGate Clients when the user connects to an AppGate Server.

#### 3.1.2 Policy Manager controlled rule set

The client can be configured to receive its rule set from a Policy Manager. In this configuration the Policy Manager provides the client with two rulesets; one to use when the client can contact the Policy Manager and a default rule set to use when the Policy Manager isn't reachable. A client can therefore have one rule set when it's connected to the inside corporate network and another rule set when it's connected to an external network.

#### 3.1.3 Default rule set

The default rule set protects the workstation when the user is not connected to an AppGate Server or Policy Manager. When the Personal Firewall is configured to receive its policy from a Policy Manager, the Policy Manager provides the default rule set. When the Personal Firewall is not configured to be controlled by a Policy Manager the default rule set is stored in the file specified by the registry entry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\AppGate\pfw\RulesFile
```

which normally points to the file:

```
%WINDIR%\AppGate\rules.cfg
```

The default rule set after the initial installation is shown in Appendix A 'Default Rule Set'.

Note: This rule file is created when the personal firewall is installed. However, if the configuration file is removed, i.e. does not exist or if it contains syntax errors, the personal firewall will use built-in rules which are identical to the rules in the configuration file at installation shown in Appendix A 'Default Rule Set'.



Note: The configuration file and log file can be created and/or edited by users unless Windows file access control prevents it. It is the Administrator's responsibility to set the proper access rights to these files and directories! This also applies to the registry key which is changeable by Power Users on Windows 2000/XP systems. If needed, the permissions should be changed!

### **Updating the default rule set**

The filter rules are updated, i.e. the current rule set is parsed, when:

1. The system is booting and the filter is installed.
2. An event occurs on a network adapter, for example when a new IP address is registered.
3. The command-line utility `agpfwctl` is asked to install a new set of rules (see below).
4. A rule set is downloaded from an AppGate Security Server
5. A new policy is fetched from the Policy Manager

The `agpfwctl` program is placed in the installation directory:

```
C:\Program Files\AppGate\pfw\Program\agpfwctl.exe
```

The program accepts the following arguments:

```
-h          Display this usage text.
-f <rule_file> Load the ruleset from
               <rule_file>.
-r          Load the ruleset from the file
               that is specified in the
               registry.
-s          Print installed rules.
-v          Be verbose when displaying rules.
             High-level rules are expanded.
-c          Display counters for number of
               matched packets for each rule.
             If '-v' flag is used counters are
               given for the expanded ruleset.
-n          Display rule numbers
-z          Install an empty ruleset
-d          Install the default ruleset. This
               will allow outgoing connections,
               DNS queries and DHCP requests.
--enable-filter Enable the filter. All IP packets
               are matched against the filter,
               ARP, RARP and 802.1X packets are let
               through and all other packets are
               dropped.
--disable-filter Disable the filter. All packets
               are passed through without
               inspection.
```

- filter-status** Query the status of the filter (enabled/disabled).
- driver-version** Display the version of the kernel-driver.
- clear-states** Remove all states held by the filter.
- enable-ethlog** Log Ethernet frame types.
- disable-ethlog** Don't log Ethernet frame types.

Note that a request to disable the filter is only active during the current session, i.e. until the system is rebooted. During the boot process, the filter rules are re-read again from the configuration file. It can also be enabled automatically by pfwd because of a policy dictated by an AppGate Security Server or an AppGate Policy Manager. To avoid that the filter gets enabled in this case, the “AppGate Personal Firewall Service” has to be turned off by stopping the service from the Windows Services control panel.

## 3.2 Counting Matched Rules

---

The firewall counts number of packets matching each rule. The command `agpfwctl -c` can be used to display the number of packets matching a rule. Typical output from this command is:

```
c:> agpfwctl -s -c
    block-bad
# 8 packets, 320 bytes
    allow-outgoing
# 2887 packets, 883017 bytes
    allow-util
# 0 packets, 0 bytes
    allow-out dhcp
# 8 packets, 2624 bytes
```

Using the “-v” flag with the `agpfwctl` command will cause all high level rules to be expanded and number of matches for each low-level rule will be displayed.

Also, please note that for TCP traffic, most packets in a session will match the “pass with-state” rule (see Appendix D 'More About States' for a detailed discussion about states). The rule permitting the traffic such as “pass out to any port 80 proto TCP keep-state” will only count the first packet in a session.

## 3.3 Rule Numbering

---

When a log entry is generated it includes `[group:rule]` to indicate which rule generated it. To make it easier to find which rules corresponds to which log entry, it's possible to include the group and rule numbers when listing the rule set by giving the `-n` flag to `agpfwctl`. For example, `agpfwctl -s -v -n` gives a verbose listing that includes the rule numbers.

## 3.4 Using Multiple Firewalls

---

It is normally possible to use multiple firewalls on the same client machine. The behavior should be that packets are passed between the firewalls in a chain manner, and for a packet to be accepted, all firewalls must accept the packet and pass it on.

Although AppGate can take no responsibility for the behavior of other third party firewalls, our preliminary tests indicate that all tested firewalls seem to work flawlessly together with the AppGate Personal Firewall.

## 4 Rules

### 4.1 What Is Filtered

---

When the filter is enabled, all ethernet frames with the type field set to the hexadecimal value 0800 (IPv4) are passed to the filter engine for evaluation. Ethernet frames with the type field set to 0806 (ARP), 8035 (RARP) and 888e (802.1X) are passed directly without comparing the frame to any rules, and *all other frames are blocked*. The packet interceptor also handles 802.2 SNAP frames correctly. This means that e.g. IPv6, IPX and NetBEUI traffic is blocked by the firewall and cannot be enabled through filter rules.

The payload of the frame (i.e. not including ethernet addresses or ethernet type) is passed to the filter engine. Any packet that is not of type IPv4 or is shorter than the minimum IP header (20 bytes) is silently dropped. Packets that pass this initial sanity check are then compared to the rule set and the first ‘pass’ or ‘block’ rule that matches the packet determines whether the packet should be allowed or not. If the packet does not match any rule, it is silently dropped, i.e. an implicit `drop` rule is always present at the end of the rule set.

### 4.2 Syntax

---

Rules can be split into several physical lines by placing a backslash (“\”) last on the line. More than one rule can be placed on the same line by inserting a semicolon (“;”) between them.

#### 4.2.1 Rule set preamble

A rule set starts with the line `version 1` to specify which version of the rule set syntax is needed to parse the rule set. The current version is 3. If no version line is present, a version is assumed based on the options present. Earlier versions of the AppGate Personal Firewall did not support a version specification, but old rule sets can be parsed by newer versions of the AppGate Personal Firewall. Note however that version 3 of the rule set syntax as specified in this document not necessarily can be parsed by versions of the AppGate Personal Firewall prior to 2.3.0.

After the version specification but before the rule set, `opt` settings may be specified. Version 1 of the rule set syntax supports the following setting:

```
opt clear-states-on-load true|false
```

*This option specifies whether the state table should be cleared whenever a rule set is loaded. If this is `true`, no states will be kept when a rule set is loaded. If this is `false`, all states already known by the AppGate Personal Firewall will be kept, which means that a `pass with-state` rule may permit traffic that would otherwise not be allowed by the rule set. The default is `true`.*

Version 2 adds the following option:

```
opt enable-filter true|false
```

*This option specifies whether the state filter should be enabled or not. If this is set to `false`, the filter engine will become completely disabled and all kind of packets will be let through. The default is `true`.*

Version 3 (current version) adds the following option:

```
opt default-window-scale <integer>
```

*This option specifies a default TCP window scale factor. This factor will be used by the filter engine as the TCP windows scale factor in the cases that the initial of a TCP session was missed by a state-keeping rule. This could be the case if the allow-appgate is being used and a new ruleset is downloaded from the AppGate server. Windows Vista makes frequent use of TCP Window scaling. In this case it is recommended to set this option to 8. Default is 0.*

## 4.2.2 Rule groups

A rule set consists of a number of groups where each group can contain a number of rules. As long as the order of the rules is the same, the grouping of the rules does not have any functional significance. The organization of rules in groups is meant to increase the readability of rules by placing related rules into the same group. Adding rules to a group also has the benefit that it does not affect the numbering of rules in other groups, which can be convenient for example when examining log files.

A group can be explicitly defined by placing a number of rules inside curly brackets (“{” and “}”). A comment of up to 127 characters can be associated with the group, using a quoted string after the opening bracket. Any rule not inside an explicitly defined group forms its own group. A group cannot contain other groups. The following example defines two groups:

```
{
  "Comment for the first group"
  block with-ipopts "block bad packets"
  pass out to any port 67 proto UDP
}
pass in from 172.16.0.0/16 "only one rule in the second group"
```

## 4.2.3 Comments

A hash sign (“#”) and anything following it on the same line is ignored (i.e. immediately discarded) and can be used for comments in the rule file. Another type of comment is placed between quotation marks (“”) and is displayed when the rule set is listed (with the `agpfwctl -s` command). Such comments can be used for a group or for a rule, for example:

```
{
  "Comment for the group"
  block with-ipopts "Comment for one rule"
}
```

## 4.3 High-level Rules

---

High-level rules are composite rules that are defined to cover some common cases in order to simplify the rule set. The rules are expanded into low-level rules that can be displayed with the `agpfwctl -s -v` command. However, in many cases only high-level rules are needed and the administrators do not have to care about the low-level rules.

The high-level rules are:

- block-bad [<log-level>]

- allow-outgoing [<log-level>]
- allow-util [<log-level>]
- allow-out [to <host>] (port <portnum> [udp|tcp] | dns | dhcp | ping) [<log-level>]
- allow-out udp-broadcast to [<broadcast-address>] port <port>[-<port>] [<log-level>]
- allow-in [from <host>] (port <portnum> [udp|tcp] | dns | dhcp | ping) [<log-level>]
- allow-in udp-broadcast [from <network>/<mask>] to port <port>[-<port>] [<log-level>]
- allow-ipt [<log-level>]
- allow-appgate [<log-level>]
- allow-policy-manager [<log-level>]

The high-level rules are described in detail in Appendix B 'High-Level Rules'.

### 4.3.1 Some notes on High-level rules

1. It might be worth noticing that 'allow-out DHCP' and 'allow-in DHCP' are not symmetrical. 'allow-out dhcp' is used for DHCP clients that do not necessarily have an IP address, while 'allow-in dhcp' assumes that the network interfaces are configured. For this reason, allow-out DHCP cannot simply be replaced with an allow-outgoing rule.
2. If the rule 'pass with-state' does not exist in the rule set before the expansion of a high-level rule that uses a state, such a rule will be prepended to the expansion of the high-level rule (see the discussion about states in Appendix D). Similarly, rules for setting up the `from-myaddress` and `to-myaddress` flags will automatically be generated when needed by a high-level rule.
3. High-level rules for TCP connections include two rules: a rule that passes the connection and keeps a state for it, followed by a rule that drops packets going in the other direction with the A flag set and the S and R flags cleared. This makes it possible to pick up connections that were already open when the filter was enabled. The first packet seen by the filter that goes in the "right" direction creates a state, while packets going in the other direction are silently dropped before the state is established. If this is not done and a 'block return RST proto TCP' rule comes later in the rule set, the RST will cause a disconnect.

## 4.4 Low-level Rules

---

Low-level rules are intended for administrators who need to take exact control over the firewall. Occasionally, there are protocols or tasks that need a low-level rule to be used. However, *most administrators do not have to care about the low-level rules*. The most common use of low-level rules is to log certain packets, for example to detect certain types of attacks or to allow complex protocols that uses multiple related connections.

In addition, care has been taken in the high-level rules to keep existing connections, whereas it is the administrators task to make sure that low-level rules allow existing connections to remain active (i.e. not to tear down existing connections and just allow new connections), see 4.3.1.

The following sections show the different parts of the rule set and what they accomplish. The exact rule syntax is described in Appendix C 'Low-level Rule Syntax'.

#### 4.4.1 Actions (pass, block, setflag, clearflag)

If the action is 'pass' or 'block' and all the other conditions are met, the parsing of the rule set is aborted and the packet is either allowed or disallowed. If the action is 'setflag' or 'clearflag' and all the other conditions are met, the appropriate flag is set or cleared and the parsing of the rule set continues (flags are described in section 4.4.8). If not all conditions are met, the rule is ignored and the packet is compared to the next rule.

An extra argument can be used with 'block' which causes a return packet to be sent:

- RST - Send a TCP RST packet. This only works if the original packet is a TCP packet.
- HOSTUNREACH - send an ICMP Host Unreachable packet.
- PORTUNREACH - send an ICMP Port Unreachable packet. This is only valid if the original packet is a TCP or UDP packet.
- PROHIB - send an ICMP Filter Prohibited packet.

Example: `block return RST log-verbose`

If no return type is specified or if the original packet is not of the appropriate type for the return type, no reply packet is generated. No reply packets are generated for ICMP messages. Returning a RST (reset) packet as in the example above, helps the application trying to establish a session since it does not have to wait for a timeout. On the other hand, it tells the sender (or potential attacker) that there is a computer receiving the packets.

#### 4.4.2 Direction (in, out, inout)

If a direction is specified, only packets going in that direction will match the rule. 'in' matches packets coming in on a network interface and 'out' matches packets that are to be sent out over an interface.



Note that if the in/out keyword is missing, the rule applies to both incoming and outgoing packets, thus writing a rule like:

```
pass to any port 22 proto TCP
```

allows outgoing connections as well as incoming connections to port 22 to and from all networks! This might or might not be the intended behavior. The above rule is identical to:

```
pass inout to any port 22 proto TCP
```

which is the recommended way to write this rule since it explicitly states that it is intended to apply for both directions.

#### 4.4.3 Logging (log, log-verbose, log-crit)

The 'log' keyword causes a log entry to be generated if the packet meets all the conditions for the rule. It gives a short log entry containing only the current date and time, rule number in square brackets [*group-number:rule-number*], action and source and destination IP numbers:

```
20030128 16:05:19: [11:1] block 172.23.2.59 -> 172.23.2.255
```

Please note that the rule numbers refer to low-level rules which may be shown with the `agpfwctl -s -v -n` command.

The 'log-verbose' keyword gives a more detailed log entry also containing the source and destination port numbers, TCP flags (A=ACK, S=SYN, P=PUSH, W=window size) and the packet size in bytes in square brackets:

```
20030205 14:30:47: [11:1] block in UDP 172.23.2.183:137 ->
172.23.2.255:137 [78]
```

```
20030205 14:46:57: [10:1] block in TCP 172.23.2.20:8080 ->
172.23.2.197:3956 AP S:2257787529 A:2545199600 W:8760 [1216]
```

Messages logged with 'log-crit' are identical to 'log-verbose' but the messages are also sent to Windows system event log.

The log information is placed in a file, `pfw.log`, in the installation directory, typically:

```
C:\Program Files\AppGate\pfw\pfw.log
```

The location can be changed by editing the path in the registry.

Since the log file is kept open by the firewall, it can only be cleared by the firewall software itself. This can be done by right-clicking the personal firewall icon in the taskbar notification area (next to the clock) and selecting "Clear Log".

#### 4.4.4 Address (from, to)

The 'from' and 'to' keywords are used to select packets on their source and destination address. An address consists of two parts: IP-address and port.

The following table describes the IP-address field:

**Table 1: IP Address**

Usage	Description
any	Any IP-address matches.
myaddress	Any of the client's IP addresses
mynet	Any address on any of the locally connected networks
1.2.3.4	Only IP-address 1.2.3.4 matches.
1.2.3.4/28	All IP-addresses in the same subnet as 1.2.3.4 match, where the subnet mask is 28 bits (addresses 1.2.3.0 to 1.2.3.15).
not 1.2.3.4	All IP-addresses except 1.2.3.4 match.
not 1.2.3.4/28	All IP-addresses except the ones in the 1.2.3.0/28 subnet match.

The field defining port numbers is described in the following table:

**Table 2: Ports**

Usage	Description
2000	Only port 2000 matches.
2000-2999	Ports 2000 to 2999 (inclusively) match.
not 2000	All ports except 2000 match.
not 2000-2999	All ports except 2000 to 2999 match.

The port numbers are only valid if the packet is TCP or UDP. Note that if no protocol is specified (see below) and port is specified, the rule will be applied to both TCP and UDP packets.

#### 4.4.5 Protocol

If a protocol is specified, only packets of that type will match the rule. The protocol can be specified either as the protocol number or with the keywords ICMP, or UDP.

For ICMP, the ICMP type and ICMP code can be specified. The ICMP types and codes are listed in Appendix E 'ICMP Types and Codes'.

For TCP, TCP flags can be specified. The TCP flags are:

- F - FIN, S - SYN, R - RST, P - PUSH, A - ACK, U - URG

If TCP flags are specified, all the specified flags must be present in the TCP header in order for a packet to match. For example:

```
"flags A/SRA"
```

means that the flags S, R and A will be tested (specified after the slash), and A must be set (specified before the slash) and consequently S and R may not be set. Since the flags F, P and U were not specified, they are not tested. It is also possible to write:

```
"flags SA"
```

which is identical to "flags SA/SA".

#### 4.4.6 IP Options (with-ipopts)

If the 'with-ipopts' keyword is specified, only IP packets with IP options are matched. If options are specified, all of the specified options must be present in order for the packet to match, unless 'any' is specified. In that case, it suffices that one of the options is present in the IP packet. If an option mask is specified, none of the options in the mask that are not also specified in the option list must be present in the IP packet.

We recommend that a rule removing all packets with IP options is present as the first rule in all the rule sets (see the `block-bad` high level rule in Appendix B 'High-Level Rules').



#### 4.4.7 Short IP fragments (short-frag)

Normally, only the IP header is compared for IP fragments with an offset greater than 0. Since no reassembly of IP fragments is performed by the filter, such packets do not contain enough information for filtering on protocol specific data or states, and those

packets are considered to match existing states. This is normally harmless since the first fragment is required in order to reassemble the IP packet, and if that is dropped by the filter the IP stack cannot perform the reassembly. There is, however, a potential security problem caused by fragments overwriting part of the protocol header. This can be done by crafting the first fragment so that it contains a protocol header that is passed by the rule set, and a later fragment that modifies this information and bypass the filter. If the keyword ‘short-frag’ is used, such packets can be intercepted. This keyword matches packets that are either the first fragment but do not contain at least an entire protocol header, or non-first fragments that start inside the protocol header. The size of the header can be specified with the ‘hsize’ argument, this defaults to 20 bytes for TCP packets and 8 bytes for UDP and ICMP packets.



We recommend that a rule removing short IP fragments is present in all rule sets (see the `block-bad` high level rule in Appendix B ‘High-Level Rules’).

#### 4.4.8 Flags (hasflag, not-hasflag)

Rules with the ‘setflag’ and ‘clearflag’ actions manipulate flags that can be checked in later rules. Flags make it possible to “mark an event” if a rule matches, and then further on and in another rule to check whether the event has occurred.

Up to 256 different flags can be defined and up to four flags can be inspected by one rule. Each flag is represented by a character string of up to 31 characters. All flags are cleared before a packet is compared to the rule set. Flags can then be set or cleared by individual rules with the ‘setflag’ and ‘clearflag’ actions.

Examples of the use of flags is when ‘to myaddress’ or ‘from myaddress’ is specified in a rule. The personal firewall will internally replace ‘myaddress’ with a flag test: ‘hasflag to-myaddress’ and ‘hasflag from-myaddress’. The expansion can be demonstrated by listing the expanded rules with the `agpfwctl -s -v` command. To make the test possible, a group of rules setting the flags will be defined and inserted before the rule using them. This expansion is performed by the Personal Firewall when the rule set is loaded and the flags will be set for all addresses that match a configured network interface. There are four flags that are used and defined by the Personal Firewall itself: from-myaddress, to-myaddress, from-mynet and to-mynet.

#### 4.4.9 State (with-state, keep-state, hastcpstate, hasudpstate)

‘with-state’ only matches packets for which a state is previously known. If ‘keep-state’ is specified for a packet that meets all the conditions in the rule, a state is established based on the packet.

*NOTE: Most low-level rules allowing outgoing TCP and UDP traffic should have a “keep-state” keyword to allow incoming data traffic!*

If a ‘keep-state’ rule explicitly defines a destination address that is the directed broadcast address for a locally connected network, a special broadcast state will be defined that allows returning packets from any IP belonging to that network.

Rules can depend on already established states with ‘hastcpstate’ and ‘hasudpstate’. This can be used to permit complex protocols with multiple related connections such as e.g. FTP or SIP.

See Appendix D for an in-depth discussion about states.

## 4.5 Examples

---

### 4.5.1 A simple rule set

This example is a simple complete rule set that allows outgoing but not incoming traffic.

The first rule removes malformed IP packets and packets that normally only occurs in an attack. All packets matching this rule are logged as critical events which means that they will be sent to the event log as well as to the Personal Firewall log file.

The second rule allows all outgoing UDP packets to the locally connected networks and any incoming replies. A reply is accepted only if it uses the same UDP ports as the original packet and has a source IP address in the locally connected network to which the original packet was sent. Note that this rule comes before the third rule, otherwise the broadcast addresses wouldn't be recognized as such but treated as normal IP addresses which would mean that the reply packets wouldn't match the generated state.

The third rule allows all traffic initiated from the local machine and any reply packets such as TCP packets in the same session, UDP reply packets with the same IP/port combination as the original packet and ICMP messages belonging to any outgoing packet.

The fourth rule allows outgoing DHCP requests and incoming DHCP replies. This isn't covered by the previous rules since the machine does not necessarily have an IP address when it sends out its request.

The last rule blocks all remaining packets. The default action is to block all packets that don't match any rule, but by adding this rule explicitly anything that matches this rule is logged.

```
block-bad log-crit
allow-out udp-broadcast to port 1-65535
allow-outgoing
allow-out dhcp
block log
```

### 4.5.2 ssh

This is a part of a rule set that allows outgoing ssh connections to any ssh server (or more accurately outgoing TCP-connections to port 22 on any machine). It assumes that no rules earlier in the rule set will block this traffic. The `pass with-state` rule does not have to be immediately before the ssh rule; one `pass with-state` rule is sufficient for the entire rule set. In this example, only outgoing packets with the SYN flag but not the ACK flag, which corresponds to the very first packet in a TCP session. This means that the state will not be picked up if the Personal Firewall is activated or the rule set is loaded when the session is already active or if the state times out. The default time out for states belonging to an established TCP session is 24 hours, if the session is idle longer than this the state will be removed and the session will stop working. If the rule instead is defined similar to the TCP rule in the `allow-outgoing` high-level rule `pfw` can pick up the state for an existing connection.

```
pass with-state
pass out from myaddress to any port 22 proto tcp \
    flags S/SA keep-state
```

### 4.5.3 NetBIOS share access with browsing

This is a part of a rule set that allows accessing NetBIOS shares as well as browsing on the local networks. Other clients on the net will not be able to browse or mount NetBIOS shares from the machine with this rule set. The `allow-out udp-broadcast` high-level rules will allow browsing while the actual share access is done on port 139 and 145.

```
allow-out udp-broadcast to port 137
allow-out udp-broadcast to port 138
allow-out port 139 tcp
allow-out port 445 tcp
```

### 4.5.4 FTP

FTP is protocol that can be difficult to allow through firewalls without opening up a big hole. The problem is that an FTP session consists of two TCP connections, but only one uses a known port number. By using related states the AppGate Personal Firewall can allow the first connection with a normal rule, and the second connection can be allowed if and only if the first connection is established. In the example below, the first connection is allowed by letting through any outgoing connections to port 21 (the FTP control connection) in the first rule. The second connection (the FTP data connection) can be either inbound or outbound depending on whether active or passive mode is used. The second rule takes care of outbound data connections (passive FTP) and the third rule handles inbound data connections (active FTP). The data connections will only be allowed if there is already a state for a TCP session from the local machine to port 21 on the remote machine.

Note that the example below is only part of a rule set, and the rule `pass with-state` must be present in the rule set for this example to work.

```
{ "Allow outgoing FTP"
  pass out from myaddress to any port 21 proto tcp keep-state \
    "FTP control connection"

  pass out from myaddress to any port 1024-65535 proto tcp \
  hastcpstate local l_ip port 1024-65535 remote r_ip port 21 \
  keep-state \
    "data connection for passive FTP"

  pass in to myaddress port 1024-65535 proto tcp \
  hastcpstate local l_ip port 1024-65535 remote r_ip port 21 \
  keep-state \
    "data connection for active FTP"
}
```

## Appendix A Default Rule Set

The following rule set is the default rule set when the firewall is installed. The rule set is built into the firewall and used if no configuration file is present or if there are syntax errors in the existing configuration file.

```
block-bad
allow-outgoing
allow-util
allow-out DHCP
block
```

These rules are expanded into the following low-level rules (for an explanation, see Appendix B 'High-Level Rules'):

```
{
  block with-iptables "block-bad"
  block short-frag
  block from 127.0.0.0/8
  block to 127.0.0.0/8
}
{
  pass with-state
  pass out proto TCP keep-state
  block in proto TCP flags A/SRA
  pass out proto UDP keep-state
  pass out proto ICMP/ECHO keep-state
  pass out proto ICMP
}
{
  pass proto ICMP/UNREACH
  pass proto ICMP/SOURCEQUENCH
  pass proto ICMP/TIMXCEED
  pass proto ICMP/PARAMPROB
}
{
  pass out to any port 67 proto UDP "DHCP"
  pass in to any port 68 proto UDP
}
block
```

## Appendix B High-Level Rules

This is a list of the high-level rule set together with their expansion into corresponding low-level rules. Most administrators do not have to care about the low-level rule expansion. However, it is included for educational reasons since they do tell how the low-level rule-set is made up.

- **block-bad** [**<log-level>**]

*This rule should always be the first rule of any rule set. It drops all packets with IP options, all packets that are too short to include a full protocol header, fragments that start inside a protocol header and packets with source or destination address 127.0.0.1. The block-bad high-level rule expands to:*

```
block <log-level> with ip-opts
block <log-level> short- frags
block <log-level> from 127.0.0.0/8
block <log-level> to 127.0.0.0/8
```

- **allow-outgoing** [**<log-level>**]

*This rule allows all outgoing TCP, UDP and ICMP packets and any incoming packets that belong to the same session. The allow-outgoing high-level rule expands to:*

```
pass out <log-level> proto TCP keep-state
block in proto TCP flags A/SAR
pass out <log-level> proto UDP keep-state
pass out <log-level> proto ICMP/ECHO keep-state
pass out <log-level> proto ICMP
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. The purpose of the block rule is to avoid that established connections are reset if a block return-RST rule appears after this rule and the first packet in the session happens to be inbound when the filter is enabled. With this rule it will be silently discarded and the state will be picked up eventually when the first outbound packet is seen.*

- **allow-util** [**<log-level>**]

*This rule enables the use of some ICMP packets necessary for good network performance. The allow-util high-level rule expands to:*

```
pass <log-level> proto ICMP/UNREACH
pass <log-level> proto ICMP/SOURCEQUENCH
pass <log-level> proto ICMP/TIMXCEED
pass <log-level> proto ICMP/PARAMPROB
```

- **allow-out** [**to <host>**] **port <portnum>** **udp** [**<log-level>**]

*This rule allows outgoing UDP packets and corresponding return packets. This allow-out high-level rule expands to:*

```
pass out <log-level> from myaddress to <host> port <portnum> \
    proto UDP keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress flag will automatically be inserted before this rule.*

- **allow-out** [**to <host>**] **port <portnum>** [**tcp**] [**<log-level>**]

*This rule allows an outgoing TCP session. This allow-out high-level rule expands to:*

```
pass out <log-level> from myaddress to <host> port <portnum> \
```

```
proto TCP keep-state
block in from <host> port <portnum> to myaddress \
proto TCP flags A/SAR
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress and to-myaddress flags will automatically be inserted before this rule. The purpose of the block rule is to avoid that established connections are reset if a block return RST rule appears after this rule and the first packet in the session happens to be inbound when the filter is enabled. With this rule it will be silently discarded and the state will be picked up eventually when the first outbound packet is seen.*

- **allow-out [to <host>] dns [<log-level>]**

*This rule allows outgoing DNS requests and the corresponding incoming answers. This allow-out high-level rule expands to:*

```
pass out <log-level> from myaddress to <host> port 53 \
proto TCP keep-state
block in from <host> port 53 to myaddress \
proto TCP flags A/SAR
pass out <log-level> from myaddress to <host> port 53 \
proto UDP keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress and to-myaddress flags will automatically be inserted before this rule. The purpose of the block rule is to avoid that established connections are reset if a block return RST rule appears after this rule and the first packet in the session happens to be inbound when the filter is enabled. With this rule it will be silently discarded and the state will be picked up eventually when the first outbound packet is seen.*

- **allow-out [to <host>] dhcp [<log-level>]**

*This rule allows outgoing DHCP requests and the corresponding incoming answers. This allow-out high-level rule expands to:*

```
pass out <log-level> to <host> port 67 proto UDP
pass in <log-level> from <host> to any port 68 proto UDP
```

- **allow-out [to <host>] ping [<log-level>]**

*This rule allows outgoing ICMP echo-request messages and the corresponding incoming ICMP echo-reply messages. This allow-out high-level rule expands to:*

```
pass out <log-level> from myaddress to <host> \
proto ICMP/ECHO keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress flag will automatically be inserted before this rule.*

- **allow-out udp-broadcast to [<broadcast-address>] port <port>[-<port>] [<log-level>]**

*This rule allows outgoing UDP-broadcasts to locally connected networks and the corresponding incoming answers. The replies can come from any IP address on the corresponding network as a reply to an outgoing UDP packet to the broadcast address. This allow-out high-level rule expands to one low-level rule for each broadcast address:*

```
pass out <log-level> from myaddress to <broadcast-address> \
port <portspec> proto UDP keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted.*

*If necessary, rules for setting up the from-myaddress flag will automatically be inserted before this rule.*

- **allow-in [from <host>] port <portnum> udp [<log-level>]**

*This rule allows incoming UDP packets and corresponding return packets. This allow-in high-level rule expands to:*

```
pass in <log-level> from <host> to myaddress port <portnum> \  
    proto UDP keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the to-myaddress flag will automatically be inserted before this rule.*

- **allow-in [from <host>] port <portnum> [tcp] [<log-level>]**

*This rule allows an incoming TCP session. This allow-in high-level rule expands to:*

```
pass in <log-level> from <host> to myaddress port <portnum> \  
    proto TCP keep-state  
block out from myaddress port <portnum> to <host> \  
    proto TCP flags A/SAR
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress and to-myaddress flags will automatically be inserted before this rule. The purpose of the block rule is to avoid that established connections are reset if a block return RST rule appears after this rule and the first packet in the session happens to be outbound when the filter is enabled. With this rule it will be silently discarded and the state will be picked up eventually when the first inbound packet is seen.*

- **allow-in [from <host>] dns [<log-level>]**

*This rule allows incoming DNS requests and the corresponding incoming answers. It would normally be used on a DNS server and not a client. This allow-in high-level rule expands to:*

```
pass in <log-level> from <host> to myaddress port 53 \  
    proto TCP keep-state  
block out from myaddress port 53 to <host> \  
    proto TCP flags A/SAR  
pass in <log-level> from <host> to myaddress port 53 \  
    proto UDP keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress and to-myaddress flags will automatically be inserted before this rule. The purpose of the block rule is to avoid that established connections are reset if a block return RST rule appears after this rule and the first packet in the session happens to be outbound when the filter is enabled. With this rule it will be silently discarded and the state will be picked up eventually when the first inbound packet is seen.*

- **allow-in [from <host>] dhcp [<log-level>]**

*This rule allows incoming DHCP requests and the corresponding outgoing answers. It would normally be used on a DHCP server and not a client. This allow-in high-level rule expands to:*

```
pass in <log-level> from <host> to myaddress port 67 proto UDP  
pass out <log-level> from myaddress to <host> port 68 \  
    proto UDP
```

*If necessary, rules for setting up the from-myaddress and to-myaddress flags will automatically be inserted before this rule.*

- **allow-in [from <host>] ping [<log-level>]**

*This rule allows incoming ICMP echo-request messages and the corresponding outgoing ICMP echo-reply messages. This allow-in high-level rule expands to:*

```
pass in <log-level> from <host> to myaddress \
    proto ICMP/ECHO keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the to-myaddress flag will automatically be inserted before this rule.*

- **allow-in udp-broadcast [from <network>/<mask>] to port <port>[-<port>] [<log-level>]**

*This rule allows incoming UDP-broadcasts from locally connected networks and the corresponding outgoing answers. This allow-in high-level rule expands to one low-level rule for each broadcast address:*

```
pass in <log-level> from <address> to <broadcast-address> \
    port <portspec> proto UDP keep-state
```

*If no pass with-state rule occurs before this rule, one is automatically inserted.*

- **allow-ipt [<log-level>]**

*This rule allows traffic over the AppGate IP Tunneling interface. It is only meaningful when using the IP Tunneling feature and the AppGate IP Tunneling Driver is installed on the client computer. The allow-ipt high-level rule expands to:*

```
pass out <log-level> from <IPT-address> to any
pass in <log-level> from any to <IPT-address>
```

*The IP tunneling address is automatically inserted, and if multiple IP Tunneling addresses are allocated there will be one pair of low-level rules for each.*



*The allow-ipt rule is not suitable for clients which are directly connected to networks which may contain hostile machines. An attacker can send special crafted packets addressed to the client's MAC address and the IP-tunnel IP address. The allow-ipt rule will allow such packets. Direct (level 2) access to the client is required for this attack, it is not possible via a properly configured router.*

- **allow-appgate [<log-level>]**

*This rule allows ssh traffic to the AppGate Security Servers to which the client has connected. This rule is only meaningful when the rule set is downloaded from an AppGate Server. If the connection to the AppGate server is via a proxy the allow-appgate high-level rule expands to:*

```
pass out <log-level> from myaddress port <localport> \
    to <AppGate IP> port <ssh port> proto TCP keep-state
pass in <log-level> from <AppGate IP> port <ssh port> \
    to myaddress port <localport> proto TCP flags A/SA
```

*and if the connection is direct it expands to:*

```
pass out <log-level> from myaddress \
    to <AppGate IP> port <ssh port> proto TCP keep-state
pass in <log-level> from <AppGate IP> port <ssh port> \
    to myaddress port <localport> proto TCP flags A/SA
```

*The AppGate IP number and the port numbers are automatically determined, and if multiple connections to AppGate servers are active when the rule set is loaded there will be one pair of low-level rules for each. If a proxy (SOCKS or HTTP) is used to connect to the AppGate, the proxy connection is allowed. If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules*

*for setting up the from-myaddress and to-myaddress flags will automatically be inserted before this rule.*

- **allow-policy-manager [<log-level>]**

*This rule allows communication with the Policy Manager. The allow-policy-manager high-level rule expands to:*

```
pass out <log-level> from myaddress to <Policy Manager IP> \
    pory <port> proto UDP keep-state
```

*The Policy Manager IP and port number are automatically inserted, and if multiple Policy Managers are configured there will be one low-level rule for each. If no pass with-state rule occurs before this rule, one is automatically inserted. If necessary, rules for setting up the from-myaddress flag will automatically be inserted before this rule.*

## Appendix C Low-level Rule Syntax

### Conventions:

- [abcd]      The text “abcd” may be present
- (t1 | t2)    Either t1 or t2 *must* be present
- <token>    To be replaced with the correct parameter or option

```
rule: <action> [<direction>] [<log spec>]
      [from <addr>] [to <addr>] [<proto>]
      [<iptopts>] [short-frag] [<flags>]
      [<related state>] [with-state] [keep-state]
      [count]

action: pass
       | block [return (RST | HOSTUNREACH |
                    PORTUNREACH | PROHIB)]
       | setflag <flagname>
       | clearflag <flagname>

direction: in | out | inout

log spec: log | log-verbose | log-crit

addr: [not] (<ip>[/prefixlen] | any | myaddress | mynet)
      [<portspec>]

portspec: port [not] <port number>[-<port number>]

proto: proto (<icmp-spec> | <tcp-spec> | <udp-spec> |
             <protocol number>)

icmp-spec: icmp [<icmptype>[/<icmpcode>]]

tcp-spec: tcp [flags <tcpflags>[/<tcpflags>]]

tcpflags: [F][S][R][P][A][U]

udp-spec: udp

iptopts: with-iptopts[(any | (<iptoptlist>[/<iptoptlist>]))]

iptoptlist: [NOP] [SEC] [LSR] [SSR] [RR] [SID] [TS] [?]

flags: <flag> [<flag> [<flag> [<flag>]]]

flag: (hasflag | not-hasflag) <flagname>

related state: (hastcpstate | hasudpstate)
               local <statespec> remote <statespec>

statespec: (l_ip | r_ip | <ip>[/<prefixlen>]) port
           (l_port | r_port | <port number>[-<port number>])
```

## Appendix D More About States

Up to 1000 different states can be kept by the packet filter. This limit is enforced in order to avoid kernel memory to be exhausted, which could hang or crash the computer. The registry entry `MaxNumStates` can be used to change this limit. There are two different types of states: slow-expiring states and quick-expiring states. Quick-expiring states expire in 30 seconds, while slow-expiring states are kept for 24 hours. Slow-expiring states are used for open TCP sessions while quick-expiring states are used for UDP and ICMP as well as TCP states that are in the process of being opened or closed.

When a packet belonging to a known state matches a rule with the `'with-state'` or `'keep-state'` keyword, the state is updated and the timeout counter is reset. It is advisable to have a `'pass with-state'` rule early in the rule set in order to avoid problems if a packet belonging to a state is matched by another rule discarding the packet.

There are two sides of a state: the remote side and the local side. For incoming packets, the destination is the local side and the source is the remote side, and vice versa. The information recorded for a state depends on the protocol.

### IP Fragments

Normally, only the IP header is compared for IP fragments with an offset greater than zero. Since no reassembly of IP fragments is performed by the filter, such packets do not contain enough information for filtering on protocol specific data or states, and those packets are considered to match existing states. This is normally harmless since the first fragment is required in order to reassemble the IP packet, and if that is dropped by the filter the IP stack can not perform the reassembly. There is however a potential problem caused by fragments overwriting part of the protocol header. This can be done so that the first fragment contains a protocol header that is allowed by the rule set, and a later fragment can modify this information and bypassing the filter. If the keyword `"short-frag"` is used such packets can be caught. This keyword matches packets that either are the first fragment but do not contain at least an entire protocol header, or non-first fragments that start inside the protocol header. The size of the header can be specified with the `"hsize"` argument, this defaults to 20 bytes for TCP packets and 8 bytes for UDP and ICMP packets. The high-level rule `"block-bad"` will block short fragments and it is recommended that this rule should be the first one in any rule set.

### UDP States

For a UDP packet to match a state, the local and remote IP addresses and the local and remote port numbers must match. ICMP messages that refer to a known UDP state are also considered to belong to the state. UDP states are quick-expiring, and the expiration counter is restarted each time a packet matches an existing UDP state.

If a `'keep-state'` rule explicitly defines a destination address that is the directed broadcast address for a locally connected network, a special broadcast state will be defined that allows returning packets from any IP belonging to that network.

### ICMP States

For the ICMP types `'ECHO'`, `'TIMESTAMP'`, and `'INFOREQUEST'`, a state can be created that will match the corresponding `'ECHOREPLY'`, `'TIMESTAMPREPLY'`, or `'INFOREPLY'`. The ICMP states are quick-expiring, and the expiry time is not updated when a reply packet is matched. ICMP packets related to TCP or UDP states do not create any separate ICMP state; they belong to the corresponding TCP or UDP state.

### TCP States

Except for local and remote IP address and port number, a TCP state also contains information about sequence numbers, acknowledgment numbers, and window sizes for the local and remote side. Two different TCP states with the same combination of IP addresses and port numbers cannot exist simultaneously.

When a TCP packet matches a 'keep-state' rule, but no previous state for that packet exists, the following rules determine what happens:

1. If the RST flag is set, no state is created. Otherwise:
2. A quick-expiring TCP state is created where the IP addresses and ports are taken from the TCP packet.
3. The source 'maxwin' for the state is set to the window size in the TCP packet and the destination 'maxwin' is set to '0'.
4. The source 'minsent' is set to the sequence number of the packet plus the number of payload bytes. The source 'maxsent' is set to the same, unless the 'More Fragments' flag is set, in which case the source 'maxsent' is set to the sequence number plus 65535.
5. The destination "acked" is set to the sequence number minus 1.
6. If the ACK flag is set, the source "acked" is set to the acknowledgment number and the destination 'maxsent' and 'minsent' are set to the same.
7. If the SYN flag is set, the source and destination states are set to 'SYN'. Otherwise:
8. If the FIN flag is set, the source state is set to 'FIN' and the destination state is set to 'ESTAB'. Otherwise:
9. The source and destination states are set to 'ESTAB'.

When a TCP packet matches a 'keep-state' or 'with-state' rule and a state that matches the packet already exists, the following happens:

1. If the RST flag is set, the state is deleted. Otherwise:
2. If the sequence number plus the TCP payload is greater than the source 'minsent', the source 'minsent' is updated to this value. If the sequence number plus the TCP

- payload (or the destination 'maxwin' if the 'More Fragments' flag is set) is greater than the source 'maxsent', the source 'maxsent' is updated to this value.
3. If the ACK flag is set and the acknowledge number is greater than the source "acked", the source "acked" is updated to this value.
  4. If the window size is larger than the source 'maxwin', the source 'maxwin' is updated to this value.
  5. If the SYN flag is set, the source and destination states are set to 'SYN' and the expiry time is set to quick. Otherwise:
  6. If the FIN flag is set, the source state is set to 'FIN' and the expiry time is set to slow. Otherwise:
  7. The expiry time is set to slow, and, if the source state is not 'FIN' it is set to ESTAB.
  8. If both the source and destination states are set to 'FIN', the expiry time is set to quick.
- In order for a packet to match a TCP state, the following must be true:

1. The local and remote IP addresses must match.
2. The local and remote ports must match.
3. If the destination 'maxwin' is not '0', and the destination state is not 'SYN', then the remote "acked" plus the remote 'maxwin' plus 1 must be greater than or equal to the sequence number plus the size of the TCP payload.
4. If the destination 'maxwin' is not 0 and the ACK flag is set, the destination 'maxsent' must be greater than or equal to the acknowledgment number, and the destination 'minsent' minus the destination 'maxwin' minus 1 must be less than or equal to the acknowledgment number.
5. If the SYN flag is set, the remote state must be set to 'SYN'. Otherwise, either the ACK flag or the RST flag must be set.

### Related states

In some protocols several channels are related, e.g. FTP where the command and data channels both belong to the same connection. In these cases it's often not possible to create good rules for each channel since they may use dynamically allocated port numbers and it is therefore possible to create rules that depend on already established states. With the keywords "hastcpstate" and "hasudpstate" it is possible to specify that a packet must only match the rule if the filter has already recorded a related state. An example for active FTP can look as follows:

```
pass with-state
pass out from myaddress to any port 21 proto tcp keep-state
pass in to myaddress port 1024-65535 proto tcp \
    hastcpstate local l_ip port 1024-65535 remote r_ip port 21 \
    keep-state
```

The first rule allows all packets that match already established states. The second rule allows and sets up a state for outgoing FTP connections, which is possible since FTP uses the well-known port 21 for the command channel. The third rule allows incoming data connections if there exists an already established state for a related command channel. In

this example we assume that the FTP client listens for incoming data connections on a port in the range 1024-65535 and that it will come from the same IP as the command channel goes to.

The “related states” concept is powerful and makes it possible to create firewall rules for complex protocols, and a few things must be kept in mind when designing a rule set. It is a good idea to limit the “to” part of the rule as much as possible, it should not allow any incoming connections to statically allocated ports. In the example above “to myaddress port 1024-65535 proto tcp” makes sure that only TCP connections to the high range of ports are allowed. It is also important to limit the reference to the related state as much as possible. If we had used “hastcpstate local l\_ip port 1-65535” in the example above it may be possible for an attacker to connect to another service (e.g. port 80 if the machine runs a web server and the rule set allows external connections to it) and choosing a local port of 21, and then using that session as the related state in order to set up new connections.

## Appendix E ICMP Types and Codes

ICMP codes, along with their types, are listed in the following table. Some types have sub-types, which are demonstrated with slashes before the sub-type number. For instance, in the code

'20/3', 20 is the ICMP code and 3 is the code of the sub-type.

**Table 3: ICMP messages**

Code	Type
0	ECHOREPLY
3	UNREACH
3/0	NET
3/1	HOST
3/2	PROTOCOL
3/3	PORT
3/4	NEEDFRAG
3/5	SRCFAIL
3/6	NETUNKNOWN
3/7	HOSTUNKNOWN
3/8	ISOLATED
3/9	NETPROHIBITED
3/10	HOSTPROHIBITED
3/11	TOSNET
3/12	TOSHOST
3/13	PROHIBITED
3/14	HOSTPRECE- DENCE
3/15	PRECE- DENCECUTOFF
4	SOURCEQUENCH
5	REDIRECT
5/0	NET
5/1	HOST
5/2	TOSNET
5/3	TOSHOST
6	ALTHOSTADDR
8	ECHO
9	ROUTERADVERT
9/0	NORMAL
9/16	COMMON
10	ROUTERSOLICIT

**Table 3: ICMP messages**

Code	Type
11	TIMXCEED
11/0	TRANSIT
11/1	REASSEMBLY
12	PARAMPROB
12/0	ERRATPTR
12/1	OPTABSENT
12/2	LENGTH
13	TIMESTAMP
14	TIMESTAMPREPLY
15	INFOREQUEST
16	INFOREPLY
17	MASKREQUEST
18	MASKREPLY
30	TRACEROUTE
31	DATACONVERR
32	MOBILEREDIRECT
33	IPV6WHEREAREYOU
34	IPV6IAMHERE
35	MOBILEREGREQUEST
36	MOBILEREGREPLY
39	SKIP
40	PHOTURIS
40/1	UNKNOWNINDEX
40/2	AUTHFAILED
40/3	DECRYPTFAILED

## Appendix F Registry Settings

The following registry settings are used by AppGate Personal Firewall. Please note that the registry normally does not need to be changed, and the computer may be rendered inoperable if it is done incorrectly.

- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\RulesFile**  
Default value: %WINDIR%\AppGate\rules.cfg  
Type: REG\_SZ
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\PolicyFile**  
Default value: %WINDIR%\AppGate\policy.cfg  
Type: REG\_SZ
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\LogFile**  
Default value: %WINDIR%\AppGate\pfw.log  
Type: REG\_SZ
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\Port**  
Default value: 7271  
Type: REG\_DWORD
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\TaskbarPort**  
Default value: 7278  
Type: REG\_DWORD
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\DebugFile**  
Default value: %WINDIR%\AppGate\debug.log  
Type: REG\_SZ
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\DebugLevel**  
Default value: 0  
Type: REG\_DWORD
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\MaxLogSize**  
Default value: 1000  
Type: REG\_DWORD
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\NumSavedLogs**  
Default value: 4  
Type: REG\_DWORD
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\MaxNumStates**  
Default value: 1000  
Type: REG\_DWORD
- **HKEY\_LOCAL\_MACHINE\SOFTWARE\AppGate\pfw\AllowedEthTypes**  
Under this key the allowed ethernet types are listed. Each ethernet type that should be allowed to pass through the filter should be specified under this key as a value of type REG\_DWORD. The name of the value may be any descriptive name of the type and the value should be set to the specified ethernet type.

