Chapter 13. Advanced Settings

This chapter describes the configurable advanced settings for NetDefendOS. The settings are divided up into the following categories:

Note
After an advanced setting is changed a reconfiguration must be performed in order for the new NetDefendOS configuration to be uploaded to the D-Link Firewall and the new value to take effect.

- IP Level Settings, page 304
- TCP Level Settings, page 307
- ICMP Level Settings, page 311
- ARP Settings, page 312
- Stateful Inspection Settings, page 314
- Connection Timeouts, page 316
- Size Limits by Protocol, page 318
- Fragmentation Settings, page 320
- Local Fragment Reassembly Settings, page 324
- DHCP Settings, page 325
- DHCPRelay Settings, page 326
- DHCPServer Settings, page 327
- IPsec Settings, page 328
- Logging Settings, page 330
- Time Synchronization Settings, page 331
- PPP Settings, page 333
- Hardware Monitor Settings, page 334
- Packet Re-assembly Settings, page 335
- Miscellaneous Settings, page 336

13.1. IP Level Settings

LogChecksumErrors

Logs occurrences of IP packets containing erroneous checksums. Normally, this is the result of the packet being damaged during network transport. All network units, both routers and workstations, drop IP packets that contain checksum errors. However, it is highly unlikely for an attack to be based on illegal checksums.

Default: Enabled
LogNonIP4
Logs occurrences of IP packets that are not version 4. NetDefendOS only accepts version 4 IP packets; everything else is discarded.
Default: 256

LogReceivedTTL0
Logs occurrences of IP packets received with the "Time To Live" (TTL) value set to zero. Under no circumstances should any network unit send packets with a TTL of 0.
Default: Enabled

Block0000Src
Block 0.0.0.0 as source address.
Default: Drop

Block0Net
Block 0.* as source addresses.
Default: DropLog

Block127Net
Block 127.* as source addresses.
Default: DropLog

BlockMulticastSrc
Block multicast both source addresses (224.0.0.0 - 255.255.255.255).
Default: DropLog

TTLMin
The minimum TTL value accepted on receipt.
Default: 3

TTLOnLow
Determines the action taken on packets whose TTL falls below the stipulated TTLMin value.
Default: DropLog

DefaultTTL
Indicates which TTL NetDefendOS is to use when originating a packet. These values are usually between 64 and 255.
Default: 255

LayerSizeConsistency
Verifies that the size information contained in each "layer" (Ethernet, IP, TCP, UDP, ICMP) is consistent with that of other layers.

Default: ValidateLogBad

**IPOptionSizes**

Verifies the size of "IP options". These options are small blocks of information that may be added to the end of each IP header. This function checks the size of well-known option types and ensures that no option exceeds the size limit stipulated by the IP header itself.

Default: ValidateLogBad

**IPOPT_SR**

Indicates whether source routing options are to be permitted. These options allow the sender of the packet to control how the packet is to be routed through each router and firewall. These constitute an enormous security risk. NetDefendOS never obeys the source routes specified by these options, regardless of this setting.

Default: DropLog

**IPOPT_TS**

Time stamp options instruct each router and firewall on the packet's route to indicate at what time the packet was forwarded along the route. These options do not occur in normal traffic. Time stamps may also be used to "record" the route a packet has taken from sender to final destination. NetDefendOS never enters information into these options, regardless of this setting.

Default: DropLog

**IPOPT_OTHER**

All options other than those specified above.

Default: DropLog

**DirectedBroadcasts**

Indicates whether NetDefendOS will forward packets which are directed to the broadcast address of its directly connected networks. It is possible to achieve this functionality by adding lines to the Rules section, but it is also included here for simplicity’s sake. This form of validation is faster than entries in the Rules section since it is more specialized.

Default: DropLog

**IPRF**

Indicates what NetDefendOS will do if there is data in the "reserved" fields of IP headers. In normal circumstances, these fields should read 0. Used by OS Fingerprinting.

Default: DropLog

**StripDFOnSmall**

Strip the Don’t Fragment flag for packets equal to or smaller than the size specified by this setting.

Default: 65535 bytes
13.2. TCP Level Settings

**TCPOptionSizes**

Verifies the size of TCP options. This function acts in the same way as IPOptionSizes described above.

Default: *ValidateLogBad*

**TCPMSSMin**

Determines the minimum permissible size of the TCP MSS. Packets containing maximum segment sizes below this limit are handled according to the next setting.

Default: *100 bytes*

**TCPMSSOnLow**

Determines the action taken on packets whose TCP MSS option falls below the stipulated TCPMSSMin value. Values that are too low could cause problems in poorly written TCP stacks.

Default: *DropLog*

**TCPMSSMax**

Determines the maximum permissible TCP MSS size. Packets containing maximum segment sizes exceeding this limit are handled according to the next setting.

Default: *1460 bytes*

**TCPMSSVPNMax**

As is the case with TCPMSSMax, this is the highest Maximum Segment Size allowed. However, this setting only controls MSS in VPN connections. This way, NetDefendOS can reduce the effective segment size used by TCP in all VPN connections. This reduces TCP fragmentation in the VPN connection even if hosts do not know how to perform MTU discovery.

Default: *1400 bytes*

**TCPMSSOnHigh**

Determines the action taken on packets whose TCP MSS option exceeds the stipulated TCPMSSMax value. Values that are too high could cause problems in poorly written TCP stacks or give rise to large quantities of fragmented packets, which will adversely affect performance.

Default: *Adjust*

**TCPMSSAutoClamping**

Automatically clamp TCP MSS according to MTU of involved interfaces, in addition to TCPMSSMax.

Default: *Enabled*

**TCPMSSLogLevel**

Determines when to log regarding too high TCP MSS, if not logged by TCPMSSOnHigh.
TCPZeroUnusedACK

Determines whether NetDefendOS should set the ACK sequence number field in TCP packets to zero if it is not used. Some operating systems reveal sequence number information this way, which can make it easier for intruders wanting to hijack established connections.

Default: Enabled

TCPZeroUnusedURG

Strip the URG pointers from all packets.

Default: Enabled

TCPOPT_WSOPT

Determines how NetDefendOS will handle window-scaling options. These are used to increase the size of the windows used by TCP; that is to say, the amount of information that can be sent before the sender expects ACK. They are also used by OS Fingerprinting. WSOPT is a common occurrence in modern networks.

Default: ValidateLogBad

TCPOPT_SACK

Determines how NetDefendOS will handle selective acknowledgement options. These options are used to ACK individual packets instead of entire series, which can increase the performance of connections experiencing extensive packet loss. They are also used by OS Fingerprinting. SACK is a common occurrence in modern networks.

Default: ValidateLogBad

TCPOPT_TSOPT

Determines how NetDefendOS will handle time stamp options. As stipulated by the PAWS (Protect Against Wrapped Sequence numbers) method, TSOPT is used to prevent the sequence numbers (a 32-bit figure) from “exceeding” their upper limit without the recipient being aware of it. This is not normally a problem. Using TSOPT, some TCP stacks optimize their connection by measuring the time it takes for a packet to travel to and from its destination. This information can then be used to generate resends faster than is usually the case. It is also used by OS Fingerprinting. TSOPT is a common occurrence in modern networks.

Default: ValidateLogBad

TCPOPT_ALTCHKREQ

Determines how NetDefendOS will handle alternate checksum request options. These options were initially intended to be used in negotiating for the use of better checksums in TCP. However, these are not understood by any today's standard systems. As NetDefendOS cannot understand checksum algorithms other than the standard algorithm, these options can never be accepted. The ALTCHKREQ option is normally never seen on modern networks.

Default: StripLog

TCPOPT_ALTCHKDATA

Determines how NetDefendOS will handle alternate checksum data options. These options are used
to transport alternate checksums where permitted by ALTCHKREQ above. Normally never seen on modern networks.

Default: StripLog

**TCPOPT_CC**

Determines how NetDefendOS will handle connection count options.

Default: StripLogBad

**TCPOPT_OTHER**

Specifies how NetDefendOS will deal with TCP options not covered by the above settings. These options usually never appear on modern networks.

Default: StripLog

**TCPSynUrg**

Specifies how NetDefendOS will deal with TCP packets with SYN (Synchronize) flags and URG (Urgent data) flags both turned on. The presence of a SYN flag indicates that a new connection is in the process of being opened, and an URG flag means that the packet contains data requiring urgent attention. These two flags should not be turned on in a single packet as they are used exclusively to crash computers with poorly implemented TCP stacks.

Default: DropLog

**TCPSynPsh**

Specifies how NetDefendOS will deal with TCP packets with SYN and PSH (Push) flags both turned on. The PSH flag means that the recipient stack should immediately send the information in the packet to the destination application in the computer. These two flags should not be turned on at the same time as it could pose a crash risk for poorly implemented TCP stacks. However, many Macintosh computers do not implement TCP correctly, meaning that they always send out SYN packets with the PSH flag turned on. This is why NetDefendOS normally removes the PSH flag and allows the packet through despite the fact that the normal setting should be dropping such packets.

Default: StripSilent

**TCPFinUrg**

Specifies how NetDefendOS will deal with TCP packets with both FIN (Finish, close connection) and URG flags turned on. This should normally never occur, as you do not usually attempt to close a connection at the same time as sending "important" data. This flag combination could be used to crash poorly implemented TCP stacks and is also used by OS Fingerprinting.

Default: DropLog

**TCPUrg**

Specifies how NetDefendOS will deal with TCP packets with the URG flag turned on, regardless of any other flags. Many TCP stacks and applications deal with Urgent flags in the wrong way and can, in the worst case scenario, cease working. Note however that some programs, such as FTP and MS SQL Server, nearly always use the URG flag.

Default: StripLog

**TCPECN**
Specifies how NetDefendOS will deal with TCP packets with either the Xmas or Ymas flag turned on. These flags are currently mostly used by OS Fingerprinting.

Note: an upcoming standard called Explicit Congestion Notification also makes use of these TCP flags, but as long as there are only a few operating systems supporting this standard, the flags should be stripped.

Default: StripLog

TCPRF

Specifies how NetDefendOS will deal with information present in the "reserved field" in the TCP header, which should normally be 0. This field is not the same as the Xmas and Ymas flags. Used by OS Fingerprinting.

Default: DropLog

TCPNULL

Specifies how NetDefendOS will deal with TCP packets that do not have any of the SYN, ACK, FIN or RST flags turned on. According to the TCP standard, such packets are illegal and are used by both OS Fingerprinting and stealth port scanners, as some firewalls are unable to detect them.

Default: DropLog

TCPSequenceNumbers

This setting determines if the sequence number range occupied by a TCP segment will be compared to the receive window announced by the receiving peer before the segment is forwarded. If the setting is set to ValidateLogBad or ValidateSilent, segments that do not match the receive window announced by the receiving peer will be dropped. If the setting is set to ValidateLogBad such drops will also be logged.

TCP sequence number validation is only possible on connections tracked by the state-engine (not on packets forwarded using a FwdFast rule).

Default: ValidateLogBad
13.3. ICMP Level Settings

ICMPSendPerSecLimit

Specifies the maximum number of ICMP messages NetDefendOS may generate per second. This includes ping replies, destination unreachable messages and also TCP RST packets. In other words, this setting limits how many Rejects per second may be generated by the Reject rules in the Rules section.

Default: 20 per second

SilentlyDropStateICMPErrors

Specifies if NetDefendOS should silently drop ICMP errors pertaining to statefully tracked open connections. If these errors are not dropped by this setting, they are passed to the rule set for evaluation just like any other packet.

Default: Enabled
13.4. ARP Settings

**ARPMatchEnetSender**

Determines if NetDefendOS will require the sender address at Ethernet level to comply with the hardware address reported in the ARP data.

Default: *DropLog*

**ARPQueryNoSenderIP**

What to do with ARP queries that have a sender IP of 0.0.0.0. Such sender IPs are never valid in responses, but network units that have not yet learned of their IP address sometimes ask ARP questions with an "unspecified" sender IP.

Default: *DropLog*

**ARPSenderIP**

Determines if the IP sender address must comply with the rules in the Access section.

Default: *Validate*

**UnsolicitedARPReplies**

Determines how NetDefendOS will handle ARP replies that it has not asked for. According to the ARP specification, the recipient should accept these. However, because this can facilitate hijacking of local connections, it is not normally allowed.

Default: *DropLog*

**ARPRequests**

Determines if NetDefendOS will automatically add the data in ARP requests to its ARP table. The ARP specification states that this should be done, but as this procedure can facilitate hijacking of local connections, it is not normally allowed. Even if ARPRequests is set to "Drop", meaning that the packet is discarded without being stored, NetDefendOS will, provided that other rules approve the request, reply to it.

Default: *Drop*

**ARPChanges**

Determines how NetDefendOS will deal with situations where a received ARP reply or ARP request would alter an existing item in the ARP table. Allowing this to take place may facilitate hijacking of local connections. However, not allowing this may cause problems if, for example, a network adapter is replaced, as NetDefendOS will not accept the new address until the previous ARP table entry has timed out.

Default: *AcceptLog*

**StaticARPChanges**

Determines how NetDefendOS will handle situations where a received ARP reply or ARP request would alter a static item in the ARP table. Of course, this is never allowed to happen. However, this setting does allow you to specify whether or not such situations are to be logged.

Default: *DropLog*
ARPExpire

Specifies how long a normal dynamic item in the ARP table is to be retained before it is removed from the table.

Default: 900 seconds (15 minutes)

ARPExpireUnknown

Specifies how long NetDefendOS is to remember addresses that cannot be reached. This is done to ensure that NetDefendOS does not continuously request such addresses.

Default: 3 seconds

ARPMulticast

Determines how NetDefendOS is to deal with ARP requests and ARP replies that state that they are multicast addresses. Such claims are usually never correct, with the exception of certain load balancing and redundancy devices, which make use of hardware layer multicast addresses.

Default: DropLog

ARPBroadcast

Determines how NetDefendOS is to deal with ARP requests and ARP replies that state that they are broadcast addresses. Such claims are usually never correct.

Default: DropLog

ARPCacheSize

How many ARP entries there can be in the cache in total.

Default: 4096

ARPHashSize

So-called "hash tables" are used to rapidly look up entries in a table. For maximum efficiency, a hash should be twice as large as the table it is indexing, so if the largest directly-connected LAN contains 500 IP addresses, the size of the ARP entry hash should be at least 1000 entries.

Default: 512

ARPHashSizeVLAN

So-called "hash tables" are used to rapidly look up entries in a table. For maximum efficiency, a hash should be twice as large as the table it is indexing, so if the largest directly-connected LAN contains 500 IP addresses, the size of the ARP entry hash should be at least 1000 entries.

Default: 64

ARPICollision

Determines the behaviour when receiving an ARP request with a sender IP address that collides with one already used on the receive interface. Possible actions: Drop or Notify.

Default: Drop
13.5. Stateful Inspection Settings

LogConnectionUsage

This generates a log message for every packet that passes through a connection that is set up in the NetDefendOS state-engine. Traffic whose destination is the D-Link Firewall itself, for example NetDefendOS management traffic, is not subject to this setting.

The log message includes port, service, source/destination IP address and interface. This setting should only be enabled for diagnostic and testing purposes since it generates unwieldy volumes of log messages and can also significantly impair throughput performance.

Default: Disabled

ConnReplace

Allows new additions to NetDefendOS’s connection list to replace the oldest connections if there is no available space.

Default: ReplaceLog

LogOpenFails

In some instances where the Rules section determines that a packet should be allowed through, the stateful inspection mechanism may subsequently decide that the packet cannot open a new connection. One example of this is a TCP packet that, although allowed by the Rules section and not being part of an established connection, has its SYN flag off. Such packets can never open new connections. In addition, new connections can never be opened by ICMP messages other than ICMP ECHO (Ping). This setting determines if NetDefendOS is to log the occurrence of such packets.

Default: Enabled

LogReverseOpens

Determines if NetDefendOS logs packets that attempt to open a new connection back through one that is already open. This only applies to TCP packets with the SYN flag turned on and to ICMP ECHO packets. In the case of other protocols such as UDP, there is no way of determining whether the remote peer is attempting to open a new connection.

Default: Enabled

LogStateViolations

Determines if NetDefendOS logs packets that violate the expected state switching diagram of a connection, for instance, getting TCP FIN packets in response to TCP SYN packets.

Default: Enabled

MaxConnections

Specifies how many connections NetDefendOS may keep open at any one time. Each connection consumes approximately 150 bytes RAM. When this setting is dynamic, NetDefendOS will try to use as many connections as is allowed by product.

Default: <dynamic>

LogConnections

Specifies how NetDefendOS, will log connections:
• **NoLog** – Does not log any connections; consequently, it will not matter if logging is enabled for either Allow or NAT rules in the Rules section; they will not be logged. However, FwdFast, Drop and Reject rules will be logged as stipulated by the settings in the Rules section.

• **Log** – Logs connections in short form; gives a short description of the connection, which rule allowed it to be made and any SAT rules that apply. Connections will also be logged when they are closed.

• **LogOC** – As for Log, but includes the two packets that cause the connection to be opened and closed. If a connection is closed as the result of a timeout, no ending packet will be logged

• **LogOCAll** – Logs all packets involved in opening and closing the connection. In the case of TCP, this covers all packets with SYN, FIN or RST flags turned on

• **LogAll** – Logs all packets in the connection.

Default: **Log**
13.6. Connection Timeouts

The settings in this section specify how long a connection can remain idle, i.e. no data being sent through it, before it is automatically closed. Please note that each connection has two timeout values: one for each direction. A connection is closed if either of the two values reaches 0.

ConnLife_TCP_SYN

Specifies how long a not yet been fully established TCP connection may idle before being closed.

Default: 60 seconds

ConnLife_TCP

Specifies how long a fully established TCP connection may idle before being closed. Connections become fully established once packets with their SYN flags off have traveled in both directions.

Default: 262144 seconds

ConnLife_TCP_FIN

Specifies how long a TCP connection about to close may idle before finally being closed. Connections reach this state when a packet with its FIN flag on has passed in any direction.

Default: 80 seconds

ConnLife_UDP

Specifies how long UDP connections may idle before being closed. This timeout value is usually low, as UDP has no way of signaling when the connection is about to close.

Default: 130 seconds

ConnLife_Ping

Specifies how long a Ping (ICMP ECHO) connection can remain idle before it is closed.

Default: 8 seconds

ConnLife_Other

Specifies how long connections using an unknown protocol can remain idle before it is closed.

Default: 130 seconds

ConnLife_IGMP

Connection lifetime for IGMP

Default: 12 seconds

AllowBothSidesToKeepConnAlive_UDP

The UDP Bidirectional keep-alive setting allows both sides to keep a UDP connection alive. The default is for NetDefendOS to mark a connection as alive (not idle) every time data is sent from the side that opened the connection. Connections that don’t receive any data from the opening side within the UDP lifetime will therefore be closed even if the other side continues to transmit data.
Default: False
13.7. Size Limits by Protocol

This section contains information about the size limits imposed on the protocols directly under IP level, i.e., TCP, UDP, ICMP, etc.

The values specified here concern the IP data contained in packets. In the case of Ethernet, a single packet can contain up to 1480 bytes of IP data without fragmentation. In addition to that, there is a further 20 bytes of IP header and 14 bytes of Ethernet header, corresponding to the maximum media transmission unit on Ethernet networks of 1514 bytes.

MaxTCPLen

Specifies the maximum size of a TCP packet including the header. This value usually correlates with the amount of IP data that can be accommodated in an unfragmented packet, since TCP usually adapts the segments it sends to fit the maximum packet size. However, this value may need to be increased by 20-50 bytes on some less common VPN systems.

Default: 1480

MaxUDPLen

Specifies the maximum size of a UDP packet including the header. This value may well need to be quite high, since many real-time applications use large, fragmented UDP packets. If no such protocols are used, the size limit imposed on UDP packets can probably be lowered to 1480 bytes.

Default: 60000 bytes

MaxICMPLen

Specifies the maximum size of an ICMP packet. ICMP error messages should never exceed 600 bytes, although Ping packets can be larger if so requested. This value may be lowered to 1000 bytes if you do not wish to use large Ping packets.

Default: 10000 bytes

MaxGRELen

Specifies the maximum size of a GRE packet. GRE, Generic Routing Encapsulation, has various uses, including the transportation of PPTP, Point to Point Tunneling Protocol, data. This value should be set at the size of the largest packet allowed to pass through the VPN connections, regardless of its original protocol, plus approx. 50 bytes.

Default: 2000 bytes

MaxESPLen

Specifies the maximum size of an ESP packet. ESP, Encapsulation Security Payload, is used by IPsec where encryption is applied. This value should be set at the size of the largest packet allowed to pass through the VPN connections, regardless of its original protocol, plus approx. 50 bytes.

Default: 2000 bytes

MaxAHLen

Specifies the maximum size of an AH packet. AH, Authentication Header, is used by IPsec where only authentication is applied. This value should be set at the size of the largest packet allowed to pass through the VPN connections, regardless of its original protocol, plus approx. 50 bytes.

Default: 2000 bytes
MaxSKIPLen

Specifies the maximum size of a SKIP packet.

Default: 2000 bytes

MaxOSPFLen

Specifies the maximum size of an OSPF packet. OSPF is a routing protocol mainly used in larger LANs.

Default: 1480

MaxIPIPLen

Specifies the maximum size of an IP-in-IP packet. IP-in-IP is used by Checkpoint Firewall-1 VPN connections when IPsec is not used. This value should be set at the size of the largest packet allowed to pass through the VPN connections, regardless of its original protocol, plus approx. 50 bytes.

Default: 2000 bytes

MaxIPCompLen

Specifies the maximum size of an IPComp packet.

Default: 2000 bytes

MaxL2TPLen

Specifies the maximum size of a Layer 2 Tunneling Protocol packet.

Default: 2000 bytes

MaxOtherSubIPLen

Specifies the maximum size of packets belonging to protocols that are not specified above.

Default: 1480 bytes

LogOversizedPackets

Specifies if NetDefendOS will log oversized packets.

Default: Enabled
13.8. Fragmentation Settings

IP is able to transport up to 65536 bytes of data. However, most media, such as Ethernet, cannot carry such huge packets. To compensate, the IP stack fragments the data to be sent into separate packets, each one given their own IP header and information that will help the recipient reassemble the original packet correctly.

However, many IP stacks are unable to handle incorrectly fragmented packets, a fact that can be exploited by intruders to crash such systems. NetDefendOS provides protection against fragmentation attacks in a number of ways.

**PseudoReass_MAXConcurrent**

Maximum number of concurrent fragment reassemblies. To drop all fragmented packets, set PseudoReass_MAXConcurrent to 0.

Default: 1024

**IllegalFrgs**

Determines how NetDefendOS will handle incorrectly constructed fragments. The term "incorrectly constructed" refers to overlapping fragments, duplicate fragments with different data, incorrect fragment sizes, etc. Possible settings include:

- **Drop** – Discards the illegal fragment without logging it. Also remembers that the packet that is being reassembled is "suspect", which can be used for logging further down the track.
- **DropLog** – Discards and logs the illegal fragment. Also remembers that the packet that is being reassembled is "suspect", which can be used for logging further down the track.
- **DropPacket** – Discards the illegal fragment and all previously stored fragments. Will not allow further fragments of this packet to pass through during ReassIllegalLinger seconds.
- **DropLogPacket** – As DropPacket, but also logs the event.
- **DropLogAll** – As DropLogPacket, but also logs further fragments belonging to this packet that arrive during ReassIllegalLinger seconds.

The choice of whether to discard individual fragments or disallow the entire packet is governed by two factors:

- It is safer to discard the whole packet.
- If, as the result of receiving an illegal fragment, you choose to discard the whole packet, attackers will be able to disrupt communication by sending illegal fragments during a reassembly, and in this way block almost all communication.

Default: DropLog – discards individual fragments and remembers that the reassembly attempt is "suspect".

**DuplicateFrgData**

If the same fragment arrives more than once, this can mean either that it has been duplicated at some point on its journey to the recipient or that an attacker is trying to disrupt the reassembly of the packet. In order to determine which is more likely, NetDefendOS compares the data components of the fragment. The comparison can be made in 2 to 512 random locations in the fragment, four bytes of each location being sampled. If the comparison is made in a larger number of samples, it is more likely to find mismatching duplicates. However, more comparisons result in higher CPU load.
FragReassemblyFail

Reassemblies may fail due to one of the following causes:

- Some of the fragments did not arrive within the time stipulated by the ReassTimeout or ReassTimeLimit settings. This may mean that one or more fragments were lost on their way across the Internet, which is a quite common occurrence.

- NetDefendOS was forced to interrupt the reassembly procedure due to new fragmented packets arriving and the system temporarily running out of resources. In situations such as these, old reassembly attempts are either discarded or marked as "failed".

- An attacker has attempted to send an incorrectly fragmented packet.

Under normal circumstances, you would not want to log failures as they occur frequently. However, it may be useful to log failures involving "suspect" fragments. Such failures may arise if, for example, the IllegalFrag setting has been set to Drop rather than DropPacket.

The following settings are available for FragReassemblyFail:

- **NoLog** - No logging is done when a reassembly attempt fails.
- **LogSuspect** - Logs failed reassembly attempts only if "suspect" fragments have been involved.
- **LogSuspectSubseq** - As LogSuspect, but also logs subsequent fragments of the packet as and when they arrive.
- **LogAll** - Logs all failed reassembly attempts.
- **LogAllSubseq** - As LogAll, but also logs subsequent fragments of the packet as and when they arrive.

Default: **LogSuspectSubseq**

DroppedFrags

If a packet is denied entry to the system as the result of the settings in the Rules section, it may also be worth logging individual fragments of that packet. The DroppedFrags setting specifies how NetDefendOS will act. Possible settings for this rule are as follows:

- **NoLog** – No logging is carried out over and above that which is stipulated in the rule set.
- **LogSuspect** - Logs individual dropped fragments of reassembly attempts affected by "suspect" fragments.
- **LogAll** - Always logs individual dropped fragments.

Default: **LogSuspect**

DuplicateFrags

If the same fragment arrives more than once, this can mean either that it has been duplicated at some point on its journey to the recipient or that an attacker is trying to disrupt the reassembly of the packet. DuplicateFrags determines whether such a fragment should be logged. Note that DuplicateFragData can also cause such fragments to be logged if the data contained in them does
not match up. Possible settings are as follows:

- **NoLog** - No logging is carried out under normal circumstances.

- **LogSuspect** - Logs duplicated fragments if the reassembly procedure has been affected by "suspect" fragments.

- **LogAll** - Always logs duplicated fragments.

Default: *LogSuspect*

**FragmentedICMP**

Other than ICMP ECHO (Ping), ICMP messages should not normally be fragmented as they contain so little data that fragmentation should never be necessary. FragmentedICMP determines the action taken when NetDefendOS receives fragmented ICMP messages that are not either ICMP ECHO or ECHOREPLY.

Default: *DropLog*

**MinimumFragLength**

MinimumFragLength determines how small all fragments, with the exception of the final fragment, of a packet can be. Although the arrival of too many fragments that are too small may cause problems for IP stacks, it is usually not possible to set this limit too high. It is rarely the case that senders create very small fragments. However, a sender may send 1480 byte fragments and a router or VPN tunnel on the route to the recipient subsequently reduce the effective MTU to 1440 bytes. This would result in the creation of a number of 1440 byte fragments and an equal number of 40 byte fragments. Because of potential problems this can cause, the default settings in NetDefendOS has been designed to allow the smallest possible fragments, 8 bytes, to pass. For internal use, where all media sizes are known, this value can be raised to 200 bytes or more.

Default: *8 bytes*

**ReassTimeout**

A reassembly attempt will be interrupted if no further fragments arrive within ReassTimeout seconds of receipt of the previous fragment.

Default: *65 seconds*

**ReassTimeLimit**

A reassembly attempt will always be interrupted ReassTimeLimit seconds after the first received fragment arrived.

Default: *90 seconds*

**ReassDoneLinger**

Once a packet has been reassembled, NetDefendOS is able to remember this for a short period of time in order to prevent further fragments, for example old duplicate fragments, of that packet from arriving.

Default: *20 seconds*
Once a whole packet has been marked as illegal, NetDefendOS is able to retain this in its memory in order to prevent further fragments of that packet from arriving.

Default: *60 seconds*
13.9. Local Fragment Reassembly Settings

**LocalReass_MaxConcurrent**

Maximum number of concurrent local reassemblies.

Default: 256

**LocalReass_MaxSize**

Maximum size of a locally reassembled packet.

Default: **10000**

**LocalReass_NumLarge**

Number of large (over 2K) local reassembly buffers (of the above size).

Default: **32**
13.10. DHCP Settings

**DHCP_MinimumLeaseTime**

Minimum lease time (seconds) accepted from the DHCP server.

Default: 60

**DHCP_VaidateBcast**

Require that the assigned broadcast address is the highest address in the assigned network.

Default: Enabled

**DHCP_AllowGlobalBcast**

Allow DHCP server to assign 255.255.255.255 as broadcast. (Non-standard.)

Default: Disabled

**DHCP_UseLinkLocalIP**

If this is enabled NetDefendOS will use a Link Local IP (169.254.*.*) instead of 0.0.0.0 while waiting for a lease.

Default: Disabled

**DHCP_DisableArpOnOffer**

Disable the ARP check done by NetDefendOS on the offered IP. The check issues an ARP request to see if the IP address is already in use.

Default: Disabled
13.11. DHCPRelay Settings

**DHCPRelay_MaxTransactions**

Maximum number of transactions at the same time.

Default: 32

**DHCPRelay_TransactionTimeout**

For how long a dhcp transaction can take place.

Default: 10 seconds

**DHCPRelay_MaxPPMPerIface**

How many dhcp-packets a client can send to through NetDefendOS to the dhcp-server during one minute.

Default: 500 packets

**DHCPRelay_MaxHops**

How many hops the dhcp-request can take between the client and the dhcp-server.

Default: 5

**DHCPRelay_MaxLeaseTime**

The maximum leastime allowed through NetDefendOS, if the DHCP server have higher leases this value will be shorted down to this value.

Default: 10000 seconds

**DHCPRelay_MaxAutoRoutes**

How many relays that can be active at the same time.

Default: 256

**DHCPserver_SaveRelayPolicy**

What policy should be used to save the relay list to the disk, possible settings are Disabled, ReconfShut, or ReconfShutTimer.

Default: ReconfShut

**DHCPRelay_AutoSaveRelayInterval**

How often should the relay list be saved to disk if DHCPserver_SaveRelayPolicy is set to ReconfShutTimer.

Default: 86400
13.12. DHCP Server Settings

**DHCPServer_SaveLeasePolicy**

What policy should be used to save the lease database to the disk, possible settings are Disabled, ReconfShut, or ReconfShutTimer.

Default: *ReconfShut*

**DHCPServer_AutoSaveLeaseInterval**

How often should the leases database be saved to disk if DHCPServer_SaveLeasePolicy is set to ReconfShutTimer.

Default: 86400
13.13. IPsec Settings

**IKESendInitialContact**

Determines whether or not IKE should send the "Initial Contact" notification message. This message is sent to each remote gateway when a connection is opened to it and there are no previous IPsec SA using that gateway.

Default: Enabled

**IKESendCRLs**

Dictates whether or not CRLs (Certificate Revocation Lists) should be sent as part of the IKE exchange. Should typically be set to ENABLE except where the remote peer does not understand CRL payloads.

Default: Enabled

**IKECRLValidityTime**

A CRL contains a "next update" field that dictates the time and date when a new CRL will be available for download from the CA. The time between CRL updates can be anything from a few hours and upwards, depending on how the CA is configured. Most CA software allow the CA administrator to issue new CRLs at any time, so even if the "next update" field says that a new CRL is available in 12 hours, there may already be a new CRL for download.

This setting limits the time a CRL is considered valid. A new CRL is downloaded when IKECRLValidityTime expires or when the "next update" time occurs. Whichever happens first.

Default: 90000

**IKEMaxCAPath**

When the signature of a user certificate is verified, NetDefendOS looks at the 'issuer name' field in the user certificate to find the CA certificate the certificate was signed by. The CA certificate may in turn be signed by another CA, which may be signed by another CA, and so on. Each certificate will be verified until one that has been marked trusted is found, or until it is determined that none of the certificates were trusted.

If there are more certificates in this path than what this setting specifies, the user certificate will be considered invalid.

Default: 15

**IPsecCertCacheMaxCerts**

Maximum number of certificates/CRLs that can be held in the internal certificate cache. When the certificate cache is full, entries will be removed according to an LRU (Least Recently Used) algorithm.

Default: 1024

**IPsecBeforeRules**

Pass IKE & IPsec (ESP/AH) traffic sent to NetDefendOS directly to the IPsec engine without consulting the rule set.

Default: Enabled
IPsecDeleteSAOnIPValidationFailure

Controls what happens to the SAs if IP validation in Config Mode fails. If Enabled, the security associations (SAs) are deleted on failure.

Default: *Disabled*
13.14. Logging Settings

LogSendPerSecLimit

This setting limits how many log packets NetDefendOS may send out per second. This value should never be set too low, as this may result in important events not being logged, nor should it be set too high. One situation where setting too high a value may cause damage is when NetDefendOS sends a log message to a server whose log receiver is not active. The server will send back an ICMP UNREACHABLE message, which may cause NetDefendOS to send another log message, which in turn will result in another ICMP UNREACHABLE message, and so on. By limiting the number of log messages NetDefendOS sends every second, you avoid encountering such devastating bandwidth consuming scenarios.

Default: 3600 seconds, once an hour
13.15. Time Synchronization Settings

**TimeSync.SyncInterval**

Seconds between each resynchronization.

Default: 86400

**TimeSync.MaxAdjust**

Maximum time drift that a server is allowed to adjust.

Default: 3600

**TimeSync.ServerType**

Type of server for time synchronization, UDPTime or SNTP (Simple Network Time Protocol).

Default: SNTP

**TimeSync.GroupIntervalSize**

Interval according to which server responses will be grouped.

Default: 10

**TimeSync.TimeServerIP1**

DNS hostname or IP Address of Timeserver 1.

Default: none

**TimeSync.TimeServerIP2**

DNS hostname or IP Address of Timeserver 2.

Default: none

**TimeSync.TimeServerIP3**

DNS hostname or IP Address of Timeserver 3.

Default: none

**TimeSync.TimeZoneOffs**

Time zone offset in minutes.

Default: 0

**TimeSync.DSTEnabled**

Perform DST adjustment according to DSTOffs/DSTStartDate/DSTEndDate.

Default: OFF

**TimeSync.DSTOffs**
DST offset in minutes.
Default: 0

**TimeSync\_DSTStartDate**

What month and day DST starts, in the format MM-DD.
Default: *none*

**TimeSync\_DSTEndDate**

What month and day DST ends, in the format MM-DD.
Default: *none*
13.16. PPP Settings

**PPP_L2TPBeforeRules**

Pass L2TP traffic sent to the D-Link Firewall directly to the L2TP Server without consulting the rule set.

Default: *Enabled*

**PPP_PPTPBeforeRules**

Pass PPTP traffic sent to the D-Link Firewall directly to the PPTP Server without consulting the rule set.

Default: *Enabled*
13.17. Hardware Monitor Settings

**HWM_PollInterval**

Polling interval for Hardware Monitor which is the delay in milliseconds between reading of hardware monitor values. Minimum 100, Maximum 10000.

Default: 500 ms

**HWMMem_Interval**

Memory polling interval which is the delay in minutes between reading of memory values. Minimum 1, Maximum 200.

Default: 15 mins

**HWMMem_LogRepetition**

Should we send a log message for each poll result that is in the Alert, Critical or Warning level, or should we only send when a new level is reached. If True, a message is sent each time HWMMem_Interval is triggered If False, a message is sent when a value goes from one level to another.

Default: False

**HWMMem_UsePercent**

True if the memory monitor uses a percentage as the unit for monitoring, False if it uses Megabyte. Applies to HWMMem_AlertLevel, HWMMem_CriticalLevel and HWMMem_WarningLevel.

Default: True

**HWMMem_AlertLevel**

Generate an Alert log message if free memory is below this value. Disable by setting to 0. Maximum value is 10,000.

Default: 0

**HWMMem_CriticalLevel**

Generate a Critical log message if free memory is below this value. Disable by setting to 0. Maximum value is 10,000.

Default: 0

**HWMMem_WarningLevel**

Generate a Warning log message if free memory is below this value. Disable by setting to 0. Maximum value 10,000.

Default: 0
13.18. Packet Re-assembly Settings

Packet re-assembly collects IP fragments into complete IP datagrams and, for TCP, reorders segments so that they are processed in the correct order and also to keep track of potential segment overlaps and to inform other subsystems of such overlaps. The associated settings limit memory used by the re-assembly subsystem.

Reassembly_MaxConnections

This setting specifies how many connections can use the re-assembly system at the same time. It is expressed as a percentage of the total number of allowed connections. Minimum 1, Maximum 100.

Default: 80

Reassembly_MaxProcessingMem

This setting specifies how much memory that the re-assembly system can allocate to process packets. It is expressed as a percentage of the total memory available. Minimum 1, Maximum 100.

Default: 3
13.19. Miscellaneous Settings

BufFloodRebootTime

As a final way out, NetDefendOS automatically reboots if its buffers have been flooded for a long time. This setting specifies this amount of time.

Default: 3600

MaxPipeUsers

The maximum number of pipe users to allocate. As pipe users are only tracked for a 20th of a second, this number usually does not need to be anywhere near the number of actual users, or the number of statefully tracked connections. If there are no configured pipes, no pipe users will be allocated, regardless of this setting. For more information about pipes and pipe users, see chapter 10, Traffic Shaping.

Default: 512