NAME

IPTables::IPv4 – Perl module for manipulating iptables rules for the IPv4 protocol

SYNOPSIS

```
use IPTables::IPv4;

$table = IPTables::IPv4::init('tablename');

%IPTables::IPv4 = (
    filter => {
                INPUT => {
                            rules => [
                                        {
                                          source => '10.0.0.0/8',
                                          jump => 'ACCEPT'
                                        }
                                      ],
                            pcnt => 50000,
                            bcnt => 1000000,
                            policy => 'DROP'
                          }
              }
);
```

DESCRIPTION

This package provides a nice interface to the IP Tables control API that fairly closely parallels the C API exported in libiptc for manipulating firewalling and forwarding rules for IPv4 packets. Also, a tied multi-layer data structure has been built, allowing the tables, chains, rules and fields to be manipulated in a more natural fashion.

Wrappers have been implemented for all methods except one (`iptc_check_packet()`), and according to Harald Welte and Rusty Russell, the unimplemented call will likely remain so. Protocol-specific match modules have been implemented for TCP, UDP and ICMP. Several target and non-protocol match modules have been implemented.

The module will be built with a default library path built into it. That can be overridden using the `IPT_IPV4_MODPATH` environment variable. If your script is being called suid root, you may want to do `delete $ENV{IPT_IPV4_MODPATH};` to ensure that someone isn't subverting your script. Make sure you do this before you `use IPTables::IPv4;` to ensure that it never loads from an unapproved path.

METHODS

Most methods will return 1 for success, or 0 for failure (and on failure, set $! to a string describing the reason for the failure). Unless otherwise noted, you can assume that all methods will use this convention.

Initialization

$table = IPTables::IPv4::init('tablename')

This sets up the connection to the kernel-level netfilter subsystem. `tablename` corresponds to the name of a table (`filter`, `nat`, `mangle`, or `dropped`) to manipulate. The call returns an object of type IPTables::IPv4::Table, which all the other methods are to be called against, if the named table exists. If it does not exist, `undef` will be returned.

Chain Operations

$is_builtin = `$table`->builtin('chainname')

This checks if the chain `chainname` is built into the current table. The method will return 1 if `chainname` is a built-in chain, or 0 if it is not.

$success = $table->create_chain('chainname')
This attempts to create the chain chainname.

$success = $table->delete_chain('chainname')
This attempts to delete the chain chainname.

($policy, $pcnt, $bcnt) = $table->get_policy('chainname')
This returns an array containing the default policy, and the number of packets and bytes which have reached the default policy, in the chain chainname. If chainname does not exist, or if it is not a built-in chain, an empty array will be returned, and $! will be set to a string containing the reason.

$refcnt = $table->get_references('chainname')
This returns the reference count for the chain chainname if it exists and is a user-defined chain. If chainname does not exist, or is a built-in chain, -1 will be returned, and $! will be set to a string containing the reason.

$is_chain = $table->is_chain('chainname')
This checks to verify that the chain chainname exists in the current table. The method will return 1 if chainname is a chain, 0 if not.

@chains = $table->*list_chains()*
This returns an array containing names of all existing chains in the table that $table points to.

$success = $table->rename_chain('oldname', 'newname')
This attempts to rename the chain oldname to newname.

$success = $table->set_policy('chainname', 'target')
$success = $table->set_policy('chainname', 'target', {pcnt => count, bcnt => count})
This attempts to set the default target for the chain chainname to target. It also allows the packet and byte counters on a chain to be set using the (optional) third argument. Those values must be passed as a hash ref, as shown.

Rule Operations

$success = $table->append_entry('chainname', $hashref)
This attempts to append the rule described in the hash referenced by $hashref to the chain chainname.

$success = $table->delete_entry('chainname', $hashref)
This attempts to delete a rule matching that described in the hash referenced by $hashref from the chain chainname.

$success = $table->delete_num_entry('chainname', $rulenum)
This attempts to delete the rule $rulenum from the chain chainname.

$success = $table->flush_entries('chainname')
This deletes all rules from the chain chainname.

$success = $table->insert_entry('chainname', $hashref, $rulenum)
This attempts to insert the rule described in the hash referenced by $hashref at index $rulenum in the chain chainname.

@rules = $table->list_rules('chainname')
This returns an array of hash references, which contain descriptions of each rule in the chain chainname.

Note that if the chain chainname does not exist, an empty list will be returned, as will listing an empty chain. Be sure to verify that the chain exists *before* you try to list the rules.

$success = $table->replace_entry('chainname', $hashref, $rulenum)
This attempts to replace the rule at index $rulenum in the chain chainname with the rule described in the hash referenced by $hashref.

$success = `$table->zero_entries('chainname')`
This zeroes all packet counters in the chain `chainname`.

Cleanup

$success = `$table->`*commit()*
This attempts to commit all changes made to the IP chains in the table that `$table` points to, and closes the connection to the kernel-level netfilter subsystem.

RULE STRUCTURE

The rules in the libiptc interface are expressed as `struct ipt_entrys`. However, I have decided to express the rules as hashes. The rules are passed around as hash references, and may contain the following fields:

source
The source address of a packet. This will appear in one of the following forms:

```
ip.add.re.ss
ip.add.re.ss/maskwidth
ip.add.re.ss/ne.t.ma.sk
```

It may be prefixed with a '!', to indicate the inverse sense of the address (i.e., match anything EXCEPT the address or address range specified).

destination
The destination address of a packet. It will appear in one of the same forms as `source` (see above).

in-interface
The network device which received the packet. Some chains cannot accept a rule with `in-interface` set (such as the `PREROUTING` chain). This may show up as a full interface name (such as `eth0`), or as a wildcarded interface name (such as `eth+`, where + is the wildcard character, which can only be used at the end of a wildcarded interface string). It may be prefixed with a '!', to indicate the inverse sense of the interface (i.e., match anything EXCEPT the interface specified).

out-interface
The network device that a packet will be sent out via. Some chains cannot accept a rule with `out-interface` set (such as the `INPUT` chain). The format is the same as that for `in-interface` (see above).

protocol
The name of the protocol of an incoming packet. It may be prefixed with a '!', to indicate the inverse sense of the protocol (i.e., match anything EXCEPT this protocol).

fragment
An integer value. 1 indicates the rule should match only fragments, 0 indicates the rule should not match any fragments. Don't set this unless you really want to either match all or no fragments.

jump
The target or chain to jump to if the rule matches.

pcnt/bcnt
The number of packets and bytes that have matched this rule since the rule was put in place, or since its counters were last zeroed.

matches
An array reference, containing a list of all the match modules which are to be used as part of the rule.

[target]-target-raw
This contains, as a string, the raw target data for a rule, if the needed module can't be found. *[target]* should be the name of the target. There will, of course, only be one of these per rule (as each rule can only have one target).

[match]−match−raw

This contains, as a string, the raw match data for a rule, if the needed module can't be found. *[match]* should be the name of the match. There can be more than one of these in one rule. If a match is specified in `matches`, and no match module is available, raw data must be provided.

MODULE-SPECIFIC RULE OPTIONS

Each module, for protocols, non-protocol matches, and non-standard targets, has specific keys associated with specific options.

TCP protocol options

source-port

The port, or range of ports (separated by a colon), that a packet is expected to come from. It can be passed as an integer, or a string. The ports may be designated by number, or by name, if the name is in */etc/services*. Its value can be prefixed with an '!' to denote inverted sense.

destination-port

The port, or range of ports (separated by a colon), that a packet is expected to go to. It can be passed as an integer, or a string. The ports may be designated by number, or by name, if the name is in */etc/services*. Its value can be prefixed with an '!' to denote inverted sense.

tcp-flags

The TCP packet flags to mask for and to compare against. It is expressed as a hash reference, containing the following keys:

`mask`

The bits to AND the TCP flags with. This expresses the flags we care about. These are expressed as an array reference, containing one or more of the TCP flag names (`SYN`, `ACK`, `FIN`, `PSH`, `RST`, and `ACK`), or the special names `ALL` or `NONE`.

comp

The bits which must be set among those which are part of the mask. If a TCP flag is listed in the mask, but not here, then it must be cleared. These are expressed as in the `mask` field.

inv Invert the sense of the TCP flag check. If this key exists in the hash, it will invert the sense of the flag check – the value is not checked.

tcp-option

The TCP option number to check for. It can be passed as an integer, or a string. Its value can be prefixed with an '!' to denote inverted sense.

UDP protocol options

source-port

The port, or range of ports (separated by a colon), that a packet is expected to come from. It can be passed as an integer, or a string. The ports may be designated by number, or by name, if the name is in */etc/services*. Its value can be prefixed with an '!' to denote inverted sense.

destination-port

The port, or range of ports (separated by a colon), that a packet is expected to go to. It can be passed as an integer, or a string. The ports may be designated by number, or by name, if the name is in */etc/services*. Its value can be prefixed with an '!' to denote inverted sense.

ICMP protocol options

icmp-type

The ICMP packet type to match. This can be passed as one of the named ICMP types, or in one of the following forms:

```
typenum
typenum/codenum
typenum/codemin-codemax
```

The value can be prefixed with an '!' to denote inverse sense.

dscp match options

This match will allow a rule to match the Differentiated Services code-point field in the IP headers of incoming packets. Only one of the following options may be used in any one rule.

DSCP supplants TOS as a mechanism for indicating the type of service being provided by a packet stream.

dscp
: Specify the numeric Differentiated Services value explicitly. The value may be prefixed with a '!' to indicate inverse sense.

dscp-class
: Specify a named Differentiated Services class. This can be one of BE, EF, CS0–CS7, AF11–AF13, AF21–AF23, AF31–AF33, AF41–AF43. The value may be prefixed with a '!' to indicate inverse sense.

iplimit match options

This match allows a rule to match on the number of simultaneous TCP connections from individual hosts, or groups of hosts. The `iplimit-above` field is required for this match.

iplimit-above
: Specify the minimum number of TCP connections which will activate this match. This must be a positive integer value. It can be passed as an integer or string. The value can be prefixed with a '!' to indicate inverse sense (i.e. match on no more than the specified number).

iplimit-mask
: Specify a mask width to group the matches by. This can be used to group connections by subnet. This can only be an integer, from 0 to 32, inclusive. The default is 32, which will count by individual IPs.

ipv4options match options

This match module allows a rule to match a packet based on certain IPv4 header flags. Only one of the source-routing flags may be set on any one rule, and the `any-opt` flag will conflict with any of the other flags if they are set in an opposing fashion.

ssrr Match packets with the strict source-routing flag set. The option's presence enables it. Its value is ignored.

lsrr Match packets with the loose source-routing flag set. The option's presence enables it. Its value is ignored.

no-srr
: Match packets with no source-routing flags set. The option's presence enables it. Its value is ignored.

rr Match packets with, or without, the record-route flag set. The value must be integer. A zero value means match packets without the flag, a nonzero value means match packets with the flag.

ts Match packets with, or without, the timestamp flag set. The value must be integer. A zero value means match packets without the flag, a nonzero value means match packets with the flag.

ra Match packets with, or without, the router-alert flag set. The value must be integer. A zero value means match packets without the flag, a nonzero value means match packets with the flag.

any-opt
: Match packets with any flag, or no flags, set. The value must be integer. A zero value means match packets with no flags set, a nonzero values means match packets with any flag set.

length match options

The `length` match allows matching based on the payload size of a packet.

length
: The packet length, or range of lengths, to match on. This value may be passed as an integer or a string. To specify a range, specify the minimum value first, followed by the maximum, separated by a colon (':') character. The value may be prefixed with a '!' to indicate inverse sense.

limit match options

The `limit` match module allows matching based on number of packets received over a period of time (specified via the `limit` field).

limit
: This field expresses the packet rate limit in terms of count per unit time. If no time unit is specified, seconds are assumed. The time unit can be any of 'sec' (or 'second'), 'min' (or 'minute'), 'hour' (or 'hr'), or 'day'. If unit time is to be specified, the limit should be specified as 'count/unit', where `count` is the number of packets to accept, and `unit` is one of the units specified above.

limit-burst
: This field indicates the initial number of packets to match. As packets are received, an internal counter will be decremented. When it reaches zero, packets will no longer meet the match criteria. As time passes at the rate specified in the `limit` field, the counter will be incremented up to this value.

mac match options

The `mac` match module allows matching based on the source MAC address of a received packet. This match is only effective in the `INPUT` chain of the `filter` table, or the `PREROUTING` chain of the `nat` table, or a user-created chain linked from either of those chains.

mac-source
: This field expresses the hardware Ethernet address to compare against an incoming packet's source MAC address. It should be passed as a string containing six hex bytes separated by colons. It can be prefixed with a '!' character, to indicate inverse sense.

mark match options

The `mark` match module allows matching based on a previously-set packet mark value, or specific bits in the mark value (with a mask value).

mark
: This field indicates what mark value, or what bits of a mark value, should be matched on. It may be specified as a full value, or as a value with a mask (in value/mask form). The mask and value may be in hex, decimal, or octal. This may be preceded by a '!' character, to indicate inverse sense. This field is required when the `mark` match is used.

mport match options

The `mport` match allows a list of single ports and/or port ranges to be used as part of the matching criteria for a rule. `mport` must be specified as part of the `matches` list, as documented above, to be used. Only one of the fields listed below may be used in a rule. The value for the field must be a reference to an array, containing port names, port numbers, or ranges consisting of a start and end, separated by a colon (':') character, specified as either port numbers or names (either may be used, mixing is allowed).

ports
: This specifies ports to match as either the source or destination port of a packet.

source-ports
: This specifies ports to match as source ports only.

destination-ports
: This specifies ports to match as destination ports only.

multiport match options

The `multiport` match module allows a list of up to 15 individual ports to be specified as part of a rule. `multiport` must be specified as part of the `matches` parameter, as documented above, in order to be used. Only one of the following options to `multiport` may be used at once. The list of ports must be passed as an array reference. The ports may be specified by name or by number. `multiport` will only work as part of rules that match either TCP or UDP protocol.

ports
: This specifies ports to match as either the source or destination port of a packet.

source-ports
: This specifies ports to match as source ports only.

destination-ports
: This specifies ports to match as destination ports only.

owner match options

The `owner` match module allows packets (in the `OUTPUT` chain of the `filter` table only) to be matched on which user, (primary) group, process ID or process group (session) ID is associated with them. `owner` must be specified as part of the `matches` parameter, as documented above, in order to be used.

uid-owner
: This specifies the UID to match on. The parameter may be either a UID number, or an actual username. The parameter may be prefixed with a '!' to indicate inverse sense.

gid-owner
: This specifies the GID to match on. The parameter may be either a GID number, or an actual group name. The parameter may be prefixed with a '!' to indicate inverse sense.

pid-owner
: This specifies the PID to match on. The parameter may be prefixed with a '!' to indicate inverse sense.

sid-owner
: This specifies the SID (or process group ID) to match on. The parameter may be prefixed with a '!' to indicate inverse sense.

state match options

The `state` match module allows stateful packet matching, using netfilter's conntrack system to match based on the state of a connection. `state` must be specified as part of the `matches` parameter, as documented above, in order to be used.

state
: This specifies which connection states to match on. This parameter must be passed as an array reference, containing scalar elements, consisting of one or more of the keywords `NEW`, `ESTABLISHED`, `RELATED`, and `INVALID`.

tcpmss match options

This specifies a range of TCP Maximum Send Size values which a rule should accept. This match only applies to TCP packets.

mss
: The Maximum Send Size value, or range of values, to match on. This value may be passed as an integer or a string. To specify a range, specify the minimum value first, followed by the maximum, separated by a colon (':') character. The value may be prefixed with a '!' to indicate inverse sense.

tos match options

The `tos` match allows matching based on Type of Service flags. (Note that TOS is deprecated in preference to DSCP.)

tos
: This specifies which Type of Service flag to match on. The Type of Service values include 'Normal−Service' (1), 'Minimize−Cost' (2), 'Maximize−Reliability' (4), 'Maximize−Throughput' (8), and 'Minimize−Delay' (16). Only one of these may be used at a time. The TOS value may be specified as a string or a number, and only these known TOS values will be accepted. This field is required when the `tos` match is used.

ttl match options

The `ttl` match allows matching based on the TTL (Time to Live) values of packets. One of the fields listed below is required, but only one may be used per rule.

ttl-eq
: This field specifies an exact Time to Live value to match on. The value may be specified as a string or a number. It may be prefixed with a '!' character to denote inverse sense (i.e., anything not equal to the specified value).

ttl-gt
: This field specifies a minimum Time to Live value to match on, i.e., only values greater than the one passed as part of this field, will be matched. The value may be specified as a string or a number.

ttl-lt
: This field specifies a maximum Time to Live value to match on, i.e., only values less than the one passed as part of this field, will be matched. The value may be specified as a string or a number.

unclean match options

The `unclean` match module matches packets which appear to be ill-formed or otherwise unusual, containing unusual/undefined bits set, invalid flag combinations, and other invalid packet configurations. This match has no option fields. It is still considered experimental. It is known in some older kernels to block packets with ECN bits set. Use it with caution.

DNAT target options

The `DNAT` target allows an incoming connection to be redirected to a specific address, or one of a group of addresses, and one or more ports. The only option to the `DNAT` target, described below, is required to indicate where a connection is to be redirected to.

to-destination
: This specifies the redirection address, or addresses, for the `DNAT` target. The addresses can be specified as a single IP address or a range, or no address with a port specification, or both. (You must have at least one of the above.) One address or address range can be specified as a scalar string, or several can be passed in an array reference (as scalar strings). They should be passed in one of the following forms:

```
ad.d.re.ss
ad.d.re.ss-ad.d.re.ss
ad.d.re.ss:port
ad.d.re.ss:port-port
ad.d.re.ss-ad.d.re.ss:port
ad.d.re.ss-ad.d.re.ss:port-port
:port
:port-port
```

DSCP target options

This target will allow a rule to apply a Differentiated Services code-point value to the IP headers of outgoing packets. Only one of the following options may be used in any one rule. This target may only be used in the `mangle` table.

DSCP supplants TOS as a mechanism for indicating the type of service being provided by a packet stream.

set-dscp
: Specify the numeric Differentiated Services value explicitly.

set-dscp-class
: Specify a named Differentiated Services class. This can be one of BE, EF, CS0–CS7, AF11–AF13, AF21–AF23, AF31–AF33, AF41–AF43.

FTOS target options

The `FTOS` target sets the Type of Service field in a packet. Unlike the `TOS` target, it allows setting an arbitrary value. This value is often used by modern routing hardware to decide how packets should be routed, depending on the needs described by the TOS value. This target is only valid in the `mangle` table. The `set-ftos` field, described below, is required when this target is used.

set-ftos
: This field specifies the Type-of-Service value to be set. The value must be non–negative, and no greater than 255.

LOG target options

The `LOG` target uses the kernel logger (`klogd`) to log a message regarding the presence of a packet matching the conditions of the rule. Certain options may be passed to modify the behavior of the `LOG` target.

log-level
: This specifies the logging level, or priority, of the message which the rule should emit when matched. The value can be numeric, or can be a log level name, which is any of `alert`, `crit`, `debug`, `emerg`, `error` (deprecated), `info`, `notice`, `panic` (a synonym for `emerg`, deprecated), or `warning`.

log-prefix
: This specifies a prefix string, which will be added to the beginning of the log entry, and can be used as an identifier.

log-tcp-sequence
: The presence of this flag indicates that the TCP sequence number of the received packet (if it is a TCP packet) will be appended to the log entry. Note that logging the sequence numbers could be a security hazard. Don't say you weren't warned. The value associated with the key is not checked.

log-tcp-options
: The presence of this flag indicates that any TCP option bits specified in the TCP packet header (if it is a TCP packet) will be appended to the log entry. The value associated with the key is not checked.

log-ip-options
: The presence of this flag indicates that any IP option bits specified in the IP packet header will be appended to the log entry. The value associated with the key is not checked.

MARK target options

The `MARK` target may be used to set a mark value on a packet. This is commonly used as a way to tag packets for policy-routing later (use iproute2 for this). This target is only valid for use in the `mangle` table.

set-mark
: This option specifies the value to mark the packet with. This field is required for the `MARK` target.

MASQUERADE target options

The MASQUERADE target causes packets to be rewritten before being sent to appear to be destined from the interface they are being routed via. This target is only valid in the nat table, in the POSTROUTING chain, or in a user-created chain in the nat table (only has an effect if it is linked to from the POSTROUTING chain).

to-ports
: This option specifies the port or port range to indicate as the packet's source port. If a range is used, the first port in the range will be tried, and if it is unavailable, successive ports will be tried through the last in the range. (If it can't be forwarded because of not having an available port, the packet will be thrown out.) Normally, the MASQUERADE target will try to map to the same port as the packet came from on the originating host whenever possible.

MIRROR target options

The MIRROR target causes a received packet to be directed back to the sending host immediately. This target has no options. It is considered experimental, and generally for testing purposes only. You could really piss some people off with this, so if you choose to use it, use it with caution, and don't say you weren't warned.

REDIRECT target options

The REDIRECT target causes incoming packets to be redirected to the address of the interface they just arrived on. This target is only valid in the nat table, in the PREROUTING chain, or in a user-created chain in the nat table (only has an effect if it is linked to from the PREROUTING chain).

to-ports
: This option specifies the port or port range to redirect the packet to. If a range is used, the first port in the range will be tried, and if it is unavailable, successive ports will be tried through the last in the range. (If it can't be redirected because of not having an available port, the packet will be thrown out.) Normally, the REDIRECT target will redirect the connection to the same port which it was destined for.

REJECT target options

The REJECT target allows incoming packets to be rejected, with an appropriate reply packet, instead of just ignoring them (as with the DROP built-in target).

reject-with
: This specifies a type of reply to send back as a rejection notice when the rule is matched. The supported types are icmp-net-unreachable (or net-unreach for short), icmp-host-unreachable (or host-unreach), icmp-port-unreachable (or port-unreach), icmp-proto-unreachable (or proto-unreach), icmp-net-prohibited (or net-prohib), icmp-host-prohibited (or host-prohib), and tcp-reset (only valid for rules which specifically match TCP packets).

SNAT target options

The SNAT target allows an incoming connection to be rewritten to appear to be from a different address. This target has only one option, described below, which is required to tell the SNAT target where the connections should be rewritten to appear to be from.

to-source
: This specifies the rewriting address, or addresses, for the SNAT target. The addresses can be specified as a single IP address or a range, or no address with a port specification, or both. (You must have at least one of the above.) One address or address range can be specified as a scalar string, or several can be passed in an array reference (as scalar strings). The address specification style is as described for the DNAT target's to-destination field, and can be seen above.

TCPMSS target options

The TCPMSS target allows limiting of the TCP Maximum Send Size value. This can be particularly useful for firewalls and NAT systems, particularly in cases where the MTU/MSS values differ between the connected networks. Only one of the fields below may be passed. This target is only valid in the mangle table. It is only allowed for TCP SYN packets.

set-mss
: This field explicitly specifies the Maximum Send Size value.

clamp-mss-to-pmtu
: This field instructs the TCPMSS target to lock the TCP Maximum Send Size value down to the Path MTU, generally determined via path MTU discovery. The path MTU is a value which indicates the largest packet which can travel unfragmented from source to destination. The TCP packet itself will actually be 40 bytes less than the path MTU value (due to IP and TCP headers). The presence of this field indicates the option is to be enabled. The value format does not matter, as the value will be ignored.

TOS target options

The TOS target sets the Type of Service bits in a packet. These bits are often used by modern routing hardware to decide how packets should be routed, depending on the needs described by the TOS bits. This target is only valid in the mangle table. (Note that TOS is deprecated in preference to DSCP.)

set-tos
: This field specifies the Type of Service flag which should be set. The Type of Service values include 'Normal−Service' (1), 'Minimize−Cost' (2), 'Maximize−Reliability' (4), 'Maximize−Throughput' (8), and 'Minimize−Delay' (16). Only one of these may be used at a time. The TOS value may be specified as a string or a number, and only these known TOS values will be accepted. This field is required when the TOS target is used.

TTL target options

The TTL target allows the time-to-live value embedded in an IP packet's header to be modified. Only one of the fields specified below may be used per rule. The value should be passed as a non-negative integer, no greater than 255. This target is only valid in the mangle table.

ttl-set
: Set the matched packet's TTL value to a specific value.

ttl-inc
: Increment the matched packet's TTL value by a specified amount. The amount cannot be zero.

ttl-dec
: Decrement the matched packet's TTL value by a specified amount. The amount cannot be zero. If the new TTL value is less than or equal to 0, a 'time−exceeded' reply will be sent back to the originating address.

ULOG target options

The ULOG target allows flexible, almost universal userspace logging. The ULOG target must be patched into your kernel and you need the ulogd daemon to use the userspace logging. More about ULOG can be found at the site of the ULOG author Harald Welte (*http://www.gnumonks.org/ftp/pub/netfilter*).

ulog-prefix
: This specifies a prefix string, which will be added to the beginning of the log entry, and can be used as an identifier.

ulog-nlgroup
: The netlink multicast group, which ulogd should bind to.

ulog-copy-range
: The amount of bytes copied for each packet to the userspace. Has to be in the range 0−1500 (0 means the whole package).

ulog-qthreshold
: The amount of packets batched together into one multipart netlink message. This reduces the number of context switches between kernel and userspace. Has to be in the range between 1 and 50.

AUTHOR

Derrik Pates, dpates@dsdk12.net

SEE ALSO

iptables (8).