Rocky Enterprise Linux 9.2 Manual Pages on command 'ipset.8'

$ man ipset.8

IPSET(8) IPSET(8)

NAME

ipset ? administration tool for IP sets

SYNOPSIS

ipset [OPTIONS] COMMAND [COMMAND-OPTIONS]

COMMANDS := { create | add | del | test | destroy | list | save | re?

store | flush | rename | swap | help | version | - }

OPTIONS := { -exist | -output { plain | save | xml } | -quiet | -re?

solve | -sorted | -name | -terse | -file filename }

ipset create SETNAME TYPENAME [CREATE-OPTIONS]

ipset add SETNAME ADD-ENTRY [ADD-OPTIONS]

ipset del SETNAME DEL-ENTRY [DEL-OPTIONS]

ipset test SETNAME TEST-ENTRY [TEST-OPTIONS]

ipset destroy [SETNAME]

ipset list [SETNAME]

ipset save [SETNAME]

ipset restore

ipset flush [SETNAME]

ipset rename SETNAME-FROM SETNAME-TO

ipset swap SETNAME-FROM SETNAME-TO

ipset help [TYPENAME]

ipset version

ipset -

DESCRIPTION

This tool is deprecated in Red Hat Enterprise Linux. It is maintenance only and will not receive new features. New setups should use nft(8). Existing setups should migrate to nft(8) when possible. See ?https://red.ht/nft_your_tables? for details.

ipset is used to set up, maintain and inspect so called IP sets in the Linux kernel. Depending on the type of the set, an IP set may store IP(v4/v6) addresses, (TCP/UDP) port numbers, IP and MAC address pairs, IP address and port number pairs, etc. See the set type definitions be? low.

Iptables matches and targets referring to sets create references, which protect the given sets in the kernel. A set cannot be destroyed while there is a single reference pointing to it.

OPTIONS

The options that are recognized by ipset can be divided into several different groups.

COMMANDS

These options specify the desired action to perform. Only one of them can be specified on the command line unless otherwise specified below. For all the long versions of the command names, you need to use only enough letters to ensure that ipset can differentiate it from all other commands. The ipset parser follows the order here when looking for the shortest match in the long command names.

n, create SETNAME TYPENAME [CREATE-OPTIONS]

Create a set identified with setname and specified type. The type may require type specific options. If the -exist option is specified, ipset ignores the error otherwise raised when the same set (setname and create parameters are identical) already

exists.

add SETNAME ADD-ENTRY [ADD-OPTIONS]

Add a given entry to the set. If the -exist option is specified, ipset ignores if the entry already added to the set.

del SETNAME DEL-ENTRY [DEL-OPTIONS]

Delete an entry from a set. If the -exist option is specified and the entry is not in the set (maybe already expired), then the command is ignored.

test SETNAME TEST-ENTRY [TEST-OPTIONS]

Test whether an entry is in a set or not. Exit status number is zero if the tested entry is in the set and nonzero if it is missing from the set.

x, destroy [SETNAME]

Destroy the specified set or all the sets if none is given. If the set has got reference(s), nothing is done and no set de? stroyed.

list [SETNAME] [OPTIONS]

List the header data and the entries for the specified set, or for all sets if none is given. The -resolve option can be used to force name lookups (which may be slow). When the -sorted op? tion is given, the entries are listed/saved sorted (which may be slow). The option -output can be used to control the format of the listing: plain, save or xml. (The default is plain.) If the option -name is specified, just the names of the existing sets are listed. If the option -terse is specified, just the set names and headers are listed. The output is printed to stdout, the option -file can be used to specify a filename instead of stdout.

save [SETNAME]

Save the given set, or all sets if none is given to stdout in a format that restore can read. The option -file can be used to specify a filename instead of stdout.

restore

Restore a saved session generated by save. The saved session can be fed from stdin or the option -file can be used to specify a filename instead of stdin.

Please note, existing sets and elements are not erased by re? store unless specified so in the restore file. All commands are allowed in restore mode except list, help, version, interactive mode and restore itself.

flush [SETNAME]

Flush all entries from the specified set or flush all sets if none is given.

e, rename SETNAME-FROM SETNAME-TO

Rename a set. Set identified by SETNAME-TO must not exist.

w, swap SETNAME-FROM SETNAME-TO

Swap the content of two sets, or in another words, exchange the name of two sets. The referred sets must exist and compatible type of sets can be swapped only.

help [TYPENAME]

Print help and set type specific help if TYPENAME is specified.

version

Print program version.

- If a dash is specified as command, then ipset enters a simple interactive mode and the commands are read from the standard in? put. The interactive mode can be finished by entering the pseudo-command quit.

OTHER OPTIONS

The following additional options can be specified. The long option names cannot be abbreviated.

-!, -exist

Ignore errors when exactly the same set is to be created or al? ready added entry is added or missing entry is deleted.

-o, -output { plain | save | xml }

Select the output format to the list command.

-q, -quiet

Suppress any output to stdout and stderr. ipset will still exit with error if it cannot continue.

-r, -resolve

When listing sets, enforce name lookup. The program will try to display the IP entries resolved to host names which requires slow DNS lookups.

-s, -sorted

Sorted output. When listing or saving sets, the entries are listed sorted.

-n, -name

List just the names of the existing sets, i.e. suppress listing of set headers and members.

-t, -terse

List the set names and headers, i.e. suppress listing of set members.

-f, -file filename

Specify a filename to print into instead of stdout (list or save commands) or read from instead of stdin (restore command).

INTRODUCTION

A set type comprises of the storage method by which the data is stored and the data type(s) which are stored in the set. Therefore the TYPE? NAME parameter of the create command follows the syntax

TYPENAME := method:datatype[,datatype[,datatype]]

where the current list of the methods are bitmap, hash, and list and the possible data types are ip, net, mac, port and iface. The dimen? sion of a set is equal to the number of data types in its type name.

When adding, deleting or testing entries in a set, the same comma sepa? rated data syntax must be used for the entry parameter of the commands, i.e

ipset add foo ipaddr,portnum,ipaddr

If host names or service names with dash in the name are used instead of IP addresses or service numbers, then the host name or service name must be enclosed in square brackets. Example:

ipset add foo [test-hostname],[ftp-data]

In the case of host names the DNS resolver is called internally by ipset but if it returns multiple IP addresses, only the first one is used.

The bitmap and list types use a fixed sized storage. The hash types use a hash to store the elements. In order to avoid clashes in the hash, a limited number of chaining, and if that is exhausted, the doubling of the hash size is performed when adding entries by the ipset command. When entries added by the SET target of iptables/ip6tables, then the hash size is fixed and the set won't be duplicated, even if the new en? try cannot be added to the set.

GENERIC CREATE AND ADD OPTIONS

timeout

All set types supports the optional timeout parameter when creating a set and adding entries. The value of the timeout parameter for the cre? ate command means the default timeout value (in seconds) for new en? tries. If a set is created with timeout support, then the same timeout option can be used to specify non-default timeout values when adding entries. Zero timeout value means the entry is added permanent to the set. The timeout value of already added elements can be changed by re-adding the element using the -exist option. The largest possible time? out value is 2147483 (in seconds). Example:

ipset create test hash:ip timeout 300

ipset add test 192.168.0.1 timeout 60

ipset -exist add test 192.168.0.1 timeout 600

When listing the set, the number of entries printed in the header might be larger than the listed number of entries for sets with the timeout extensions: the number of entries in the set is updated when elements added/deleted to the set and periodically when the garbage collector evicts the timed out entries.

counters, packets, bytes

All set types support the optional counters option when creating a set. If the option is specified then the set is created with packet and byte

counters per element support. The packet and byte counters are initial?
ized to zero when the elements are (re-)added to the set, unless the
packet and byte counter values are explicitly specified by the packets
and bytes options. An example when an element is added to a set with
non-zero counter values:

ipset create foo hash:ip counters

ipset add foo 192.168.1.1 packets 42 bytes 1024

comment

All set types support the optional comment extension. Enabling this
extension on an ipset enables you to annotate an ipset entry with an
arbitrary string. This string is completely ignored by both the kernel
and ipset itself and is purely for providing a convenient means to doc?
ument the reason for an entry's existence. Comments must not contain
any quotation marks and the usual escape character (\) has no meaning.
For example, the following shell command is illegal:

ipset add foo 1.1.1.1 comment "this comment is \"bad\""

In the above, your shell will of course escape the quotation marks and
ipset will see the quote marks in the argument for the comment, which
will result in a parse error. If you are writing your own system, you
should avoid creating comments containing a quotation mark if you do
not want to break "ipset save" and "ipset restore", nonetheless, the
kernel will not stop you from doing so. The following is perfectly ac?
ceptable:

ipset create foo hash:ip comment

ipset add foo 192.168.1.1/24 comment "allow access to SMB share
on \\\\fileserv\\"

the above would appear as: "allow access to SMB share on \\file?
serv\"

skbinfo, skbmark, skbprio, skbqueue

All set types support the optional skbinfo extension. This extension
allows you to store the metainfo (firewall mark, tc class and hardware
queue) with every entry and map it to packets by usage of SET netfilter
target with --map-set option. skbmark option format: MARK or

MARK/MASK, where MARK and MASK are 32bit hex numbers with 0x prefix. If only mark is specified mask 0xffffffff are used. skbprio option has tc class format: MAJOR:MINOR, where major and minor numbers are hex with? out 0x prefix. skbqueue option is just decimal number.

ipset create foo hash:ip skbinfo

ipset add foo skbmark 0x1111/0xff00ffff skbprio 1:10 skbqueue 10

hashsize

This parameter is valid for the create command of all hash type sets. It defines the initial hash size for the set, default is 1024. The hash size must be a power of two, the kernel automatically rounds up non power of two hash sizes to the first correct value. Example:

ipset create test hash:ip hashsize 1536

maxelem

This parameter is valid for the create command of all hash type sets. It defines the maximal number of elements which can be stored in the set, default 65536. Example:

ipset create test hash:ip maxelem 2048

bucketsize

This parameter is valid for the create command of all hash type sets. It specifies the maximal number of elements which can be stored in a hash bucket. Possible values are any even number between 2-14 and the default is 14. Setting the value lower forces ipset to create larger hashes which consumes more memory but gives more speed at matching in the set. Example:

ipset create test hash:ip bucketsize 2

family { inet | inet6 }

This parameter is valid for the create command of all hash type sets except for hash:mac. It defines the protocol family of the IP ad? dresses to be stored in the set. The default is inet, i.e IPv4. For the inet family one can add or delete multiple entries by specifying a range or a network of IPv4 addresses in the IP address part of the en? try:

ipaddr := { ip | fromaddr-toaddr | ip/cidr }

netaddr := { fromaddr-toaddr | ip/cidr }

Example:

ipset create test hash:ip family inet6

nomatch

The hash set types which can store net type of data (i.e. hash:*net*) support the optional nomatch option when adding entries. When matching elements in the set, entries marked as nomatch are skipped as if those were not added to the set, which makes possible to build up sets with exceptions. See the example at hash type hash:net below.

When elements are tested by ipset, the nomatch flags are taken into ac? count. If one wants to test the existence of an element marked with no? match in a set, then the flag must be specified too.

forceadd

All hash set types support the optional forceadd parameter when creat? ing a set. When sets created with this option become full the next ad? dition to the set may succeed and evict a random entry from the set.

ipset create foo hash:ip forceadd

wildcard

This flag is valid when adding elements to a hash:net,iface set. If the flag is set, then prefix matching is used when comparing with this ele? ment. For example, an element containing the interface name "eth" will match any name with that prefix.

SET TYPES

bitmap:ip

The bitmap:ip set type uses a memory range to store either IPv4 host (default) or IPv4 network addresses. A bitmap:ip type of set can store up to 65536 entries.

CREATE-OPTIONS := range fromip-toip|ip/cidr [netmask cidr] [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := { ip | fromip-toip | ip/cidr }

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := { ip | fromip-toip | ip/cidr }

TEST-ENTRY := ip

Mandatory create options:

range fromip-toip|ip/cidr

Create the set from the specified inclusive address range ex?
pressed in an IPv4 address range or network. The size of the
range (in entries) cannot exceed the limit of maximum 65536 ele?
ments.

Optional create options:

netmask cidr

When the optional netmask parameter specified, network addresses
will be stored in the set instead of IP host addresses. The cidr
prefix value must be between 1-32. An IP address will be in the
set if the network address, which is resulted by masking the ad?
dress with the specified netmask, can be found in the set.

The bitmap:ip type supports adding or deleting multiple entries in one
command.

Examples:

ipset create foo bitmap:ip range 192.168.0.0/16
ipset add foo 192.168.1/24
ipset test foo 192.168.1.1

bitmap:ip,mac

The bitmap:ip,mac set type uses a memory range to store IPv4 and a MAC
address pairs. A bitmap:ip,mac type of set can store up to 65536 en?
tries.

CREATE-OPTIONS := range fromip-toip|ip/cidr [timeout value] [coun?
ters] [comment] [skbinfo]

ADD-ENTRY := ip[,macaddr]

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [
comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := ip[,macaddr]

TEST-ENTRY := ip[,macaddr]

Mandatory options to use when creating a bitmap:ip,mac type of set:

range fromip-toip|ip/cidr

Create the set from the specified inclusive address range ex?
pressed in an IPv4 address range or network. The size of the
range cannot exceed the limit of maximum 65536 entries.

The bitmap:ip,mac type is exceptional in the sense that the MAC part can be left out when adding/deleting/testing entries in the set. If we add an entry without the MAC address specified, then when the first time the entry is matched by the kernel, it will automatically fill out the missing MAC address with the MAC address from the packet. The source MAC address is used if the entry matched due to a src parameter of the set match, and the destination MAC address is used if available and the entry matched due to a dst parameter. If the entry was speci? fied with a timeout value, the timer starts off when the IP and MAC ad? dress pair is complete.

The bitmap:ip,mac type of sets require two src/dst parameters of the set match and SET target netfilter kernel modules. For matches on des? tination MAC addresses, see COMMENTS below.

Examples:

```
ipset create foo bitmap:ip,mac range 192.168.0.0/16
ipset add foo 192.168.1.1,12:34:56:78:9A:BC
ipset test foo 192.168.1.1
```

bitmap:port

The bitmap:port set type uses a memory range to store port numbers and such a set can store up to 65536 ports.

CREATE-OPTIONS := range fromport-toport [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := { [proto:]port | [proto:]fromport-toport }

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := { [proto:]port | [proto:]fromport-toport }

TEST-ENTRY := [proto:]port

Mandatory options to use when creating a bitmap:port type of set:

range [proto:]fromport-toport

Create the set from the specified inclusive port range.

The set match and SET target netfilter kernel modules interpret the stored numbers as TCP or UDP port numbers.

proto only needs to be specified if a service name is used and that name does not exist as a TCP service. The protocol is never stored in the set, just the port number of the service.

Examples:

```
ipset create foo bitmap:port range 0-1024
ipset add foo 80
ipset test foo 80
ipset del foo udp:[macon-udp]-[tn-tl-w2]
```

hash:ip

The hash:ip set type uses a hash to store IP host addresses (default) or network addresses. Zero valued IP address cannot be stored in a hash:ip type of set.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max? elem value] [bucketsize value] [netmask cidr] [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := ipaddr

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := ipaddr

TEST-ENTRY := ipaddr

Optional create options:

netmask cidr

When the optional netmask parameter specified, network addresses will be stored in the set instead of IP host addresses. The cidr prefix value must be between 1-32 for IPv4 and between 1-128 for IPv6. An IP address will be in the set if the network address, which is resulted by masking the address with the netmask, can be found in the set. Examples:

```
ipset create foo hash:ip netmask 30
ipset add foo 192.168.1.0/24
ipset test foo 192.168.1.2
```

hash:mac

The hash:mac set type uses a hash to store MAC addresses. Zero valued MAC addresses cannot be stored in a hash:mac type of set. For matches on destination MAC addresses, see COMMENTS below.

CREATE-OPTIONS := [hashsize value] [maxelem value] [bucketsize value] [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := macaddr

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := macaddr

TEST-ENTRY := macaddr

Examples:

ipset create foo hash:mac

ipset add foo 01:02:03:04:05:06

ipset test foo 01:02:03:04:05:06

hash:ip,mac

The hash:ip,mac set type uses a hash to store IP and a MAC address pairs. Zero valued MAC addresses cannot be stored in a hash:ip,mac type of set. For matches on destination MAC addresses, see COMMENTS below.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max? elem value] [bucketsize value] [timeout value] [counters] [com? ment] [skbinfo]

ADD-ENTRY := ipaddr,macaddr

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := ipaddr,macaddr

TEST-ENTRY := ipaddr,macaddr

Examples:

ipset create foo hash:ip,mac

ipset add foo 1.1.1.1,01:02:03:04:05:06

ipset test foo 1.1.1.1,01:02:03:04:05:06

hash:net

The hash:net set type uses a hash to store different sized IP network

addresses. Network address with zero prefix size cannot be stored in this type of sets.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max? elem value] [bucketsize value] [timeout value] [counters] [com? ment] [skbinfo]

ADD-ENTRY := netaddr

ADD-OPTIONS := [timeout value] [nomatch] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := netaddr

TEST-ENTRY := netaddr

where netaddr := ip[/cidr]

When adding/deleting/testing entries, if the cidr prefix parameter is not specified, then the host prefix value is assumed. When adding/deleting entries, the exact element is added/deleted and over? lapping elements are not checked by the kernel. When testing entries, if a host address is tested, then the kernel tries to match the host address in the networks added to the set and reports the result accord? ingly.

From the set netfilter match point of view the searching for a match always starts from the smallest size of netblock (most specific prefix) to the largest one (least specific prefix) added to the set.

When adding/deleting IP addresses to the set by the SET netfilter target, it will be added/deleted by the most specific prefix which can be found in the set, or by the host prefix value if the set is empty.

The lookup time grows linearly with the number of the different prefix values added to the set.

Example:

```
ipset create foo hash:net
ipset add foo 192.168.0.0/24
ipset add foo 10.1.0.0/16
ipset add foo 192.168.0/24
```

ipset add foo 192.168.0/30 nomatch

When matching the elements in the set above, all IP addresses will match from the networks 192.168.0.0/24, 10.1.0.0/16 and 192.168.0/24 except the ones from 192.168.0/30.

hash:net,net

The hash:net,net set type uses a hash to store pairs of different sized IP network addresses. Bear in mind that the first parameter has precedence over the second, so a nomatch entry could be potentially be ineffective if a more specific first parameter existed with a suitable second parameter. Network address with zero prefix size cannot be stored in this type of set.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max? elem value] [bucketsize value] [timeout value] [counters] [com? ment] [skbinfo]

ADD-ENTRY := netaddr,netaddr

ADD-OPTIONS := [timeout value] [nomatch] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := netaddr,netaddr

TEST-ENTRY := netaddr,netaddr

where netaddr := ip[/cidr]

When adding/deleting/testing entries, if the cidr prefix parameter is not specified, then the host prefix value is assumed. When adding/deleting entries, the exact element is added/deleted and over? lapping elements are not checked by the kernel. When testing entries, if a host address is tested, then the kernel tries to match the host address in the networks added to the set and reports the result accord? ingly.

From the set netfilter match point of view the searching for a match always starts from the smallest size of netblock (most specific prefix) to the largest one (least specific prefix) with the first param having precedence. When adding/deleting IP addresses to the set by the SET netfilter target, it will be added/deleted by the most

specific prefix which can be found in the set, or by the host prefix value if the set is empty.

The lookup time grows linearly with the number of the different prefix values added to the first parameter of the set. The number of secondary prefixes further increases this as the list of secondary prefixes is traversed per primary prefix.

Example:

```
ipset create foo hash:net,net
ipset add foo 192.168.0.0/24,10.0.1.0/24
ipset add foo 10.1.0.0/16,10.255.0.0/24
ipset add foo 192.168.0/24,192.168.54.0-192.168.54.255
ipset add foo 192.168.0/30,192.168.64/30 nomatch
```

When matching the elements in the set above, all IP addresses will match from the networks 192.168.0.0/24<->10.0.1.0/24, 10.1.0.0/16<->10.255.0.0/24 and 192.168.0/24<->192.168.54.0/24 except the ones from 192.168.0/30<->192.168.64/30.

hash:ip,port

The hash:ip,port set type uses a hash to store IP address and port number pairs. The port number is interpreted together with a protocol (default TCP) and zero protocol number cannot be used.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [maxelem value] [bucketsize value] [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := ipaddr,[proto:]port

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := ipaddr,[proto:]port

TEST-ENTRY := ipaddr,[proto:]port

The [proto:]port part of the elements may be expressed in the following forms, where the range variations are valid when adding or deleting entries:

portname[-portname]

TCP port or range of ports expressed in TCP portname identifiers

from /etc/services

portnumber[-portnumber]

TCP port or range of ports expressed in TCP port numbers

tcp|sctp|udp|udplite:portname|portnumber[-portname|portnumber]

TCP, SCTP, UDP or UDPLITE port or port range expressed in port name(s) or port number(s)

icmp:codename|type/code

ICMP codename or type/code. The supported ICMP codename identifiers can always be listed by the help command.

icmpv6:codename|type/code

ICMPv6 codename or type/code. The supported ICMPv6 codename identifiers can always be listed by the help command.

proto:0

All other protocols, as an identifier from /etc/protocols or number. The pseudo port number must be zero.

The hash:ip,port type of sets require two src/dst parameters of the set match and SET target kernel modules.

Examples:

```
ipset create foo hash:ip,port
ipset add foo 192.168.1.0/24,80-82
ipset add foo 192.168.1.1,udp:53
ipset add foo 192.168.1.1,vrrp:0
ipset test foo 192.168.1.1,80
```

hash:net,port

The hash:net,port set type uses a hash to store different sized IP network address and port pairs. The port number is interpreted together with a protocol (default TCP) and zero protocol number cannot be used. Network address with zero prefix size is not accepted either.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [maxelem value] [bucketsize value] [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := netaddr,[proto:]port

ADD-OPTIONS := [timeout value] [nomatch] [packets value] [bytes

value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := netaddr,[proto:]port

TEST-ENTRY := netaddr,[proto:]port

where netaddr := ip[/cidr]

For the netaddr part of the elements see the description at the hash:net set type. For the [proto:]port part of the elements see the description at the hash:ip,port set type.

When adding/deleting/testing entries, if the cidr prefix parameter is not specified, then the host prefix value is assumed. When adding/deleting entries, the exact element is added/deleted and over? lapping elements are not checked by the kernel. When testing entries, if a host address is tested, then the kernel tries to match the host address in the networks added to the set and reports the result accord? ingly.

From the set netfilter match point of view the searching for a match always starts from the smallest size of netblock (most specific prefix) to the largest one (least specific prefix) added to the set.

When adding/deleting IP addresses to the set by the SET netfilter target, it will be added/deleted by the most specific prefix which can be found in the set, or by the host prefix value if the set is empty.

The lookup time grows linearly with the number of the different prefix values added to the set.

Examples:

```
ipset create foo hash:net,port
ipset add foo 192.168.0/24,25
ipset add foo 10.1.0.0/16,80
ipset test foo 192.168.0/24,25
```

hash:ip,port,ip

The hash:ip,port,ip set type uses a hash to store IP address, port num? ber and a second IP address triples. The port number is interpreted to? gether with a protocol (default TCP) and zero protocol number cannot be

used.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max?
elem value] [bucketsize value] [timeout value] [counters] [com?
ment] [skbinfo]

ADD-ENTRY := ipaddr,[proto:]port,ip

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [
comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := ipaddr,[proto:]port,ip

TEST-ENTRY := ipaddr,[proto:]port,ip

For the first ipaddr and [proto:]port parts of the elements see the de?
scriptions at the hash:ip,port set type.

The hash:ip,port,ip type of sets require three src/dst parameters of
the set match and SET target kernel modules.

Examples:

ipset create foo hash:ip,port,ip

ipset add foo 192.168.1.1,80,10.0.0.1

ipset test foo 192.168.1.1,udp:53,10.0.0.1

hash:ip,port,net

The hash:ip,port,net set type uses a hash to store IP address, port
number and IP network address triples. The port number is interpreted
together with a protocol (default TCP) and zero protocol number cannot
be used. Network address with zero prefix size cannot be stored either.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max?
elem value] [bucketsize value] [timeout value] [counters] [com?
ment] [skbinfo]

ADD-ENTRY := ipaddr,[proto:]port,netaddr

ADD-OPTIONS := [timeout value] [nomatch] [packets value] [bytes
value] [comment string] [skbmark value] [skbprio value] [
skbqueue value]

DEL-ENTRY := ipaddr,[proto:]port,netaddr

TEST-ENTRY := ipaddr,[proto:]port,netaddr

where netaddr := ip[/cidr]

For the ipaddr and [proto:]port parts of the elements see the descrip?

tions at the hash:ip,port set type. For the netaddr part of the ele? ments see the description at the hash:net set type.

From the set netfilter match point of view the searching for a match always starts from the smallest size of netblock (most specific cidr) to the largest one (least specific cidr) added to the set. When adding/deleting triples to the set by the SET netfilter target, it will be added/deleted by the most specific cidr which can be found in the set, or by the host cidr value if the set is empty.

The lookup time grows linearly with the number of the different cidr values added to the set.

The hash:ip,port,net type of sets require three src/dst parameters of the set match and SET target kernel modules.

Examples:

```
ipset create foo hash:ip,port,net
ipset add foo 192.168.1,80,10.0.0/24
ipset add foo 192.168.2,25,10.1.0.0/16
ipset test foo 192.168.1,80.10.0.0/24
```

hash:ip,mark

The hash:ip,mark set type uses a hash to store IP address and packet mark pairs.

CREATE-OPTIONS := [family { inet | inet6 }] [markmask value] [hashsize value] [maxelem value] [bucketsize value] [timeout value] [counters] [comment] [skbinfo]

ADD-ENTRY := ipaddr,mark

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := ipaddr,mark

TEST-ENTRY := ipaddr,mark

Optional create options:

markmask value

Allows you to set bits you are interested in the packet mark. This values is then used to perform bitwise AND operation for every mark added. markmask can be any value between 1 and

4294967295, by default all 32 bits are set.

The mark can be any value between 0 and 4294967295.

The hash:ip,mark type of sets require two src/dst parameters of the set match and SET target kernel modules.

Examples:

ipset create foo hash:ip,mark

ipset add foo 192.168.1.0/24,555

ipset add foo 192.168.1.1,0x63

ipset add foo 192.168.1.1,111236

hash:net,port,net

The hash:net,port,net set type behaves similarly to hash:ip,port,net but accepts a cidr value for both the first and last parameter. Either subnet is permitted to be a /0 should you wish to match port between all destinations.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max? elem value] [bucketsize value] [timeout value] [counters] [com? ment] [skbinfo]

ADD-ENTRY := netaddr,[proto:]port,netaddr

ADD-OPTIONS := [timeout value] [nomatch] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := netaddr,[proto:]port,netaddr

TEST-ENTRY := netaddr,[proto:]port,netaddr

where netaddr := ip[/cidr]

For the [proto:]port part of the elements see the description at the hash:ip,port set type. For the netaddr part of the elements see the de? scription at the hash:net set type.

From the set netfilter match point of view the searching for a match always starts from the smallest size of netblock (most specific cidr) to the largest one (least specific cidr) added to the set. When adding/deleting triples to the set by the SET netfilter target, it will be added/deleted by the most specific cidr which can be found in the set, or by the host cidr value if the set is empty. The first sub?

net has precedence when performing the most-specific lookup, just as for hash:net,net

The lookup time grows linearly with the number of the different cidr values added to the set and by the number of secondary cidr values per primary.

The hash:net,port,net type of sets require three src/dst parameters of the set match and SET target kernel modules.

Examples:

```
ipset create foo hash:net,port,net
ipset add foo 192.168.1.0/24,0,10.0.0/24
ipset add foo 192.168.2.0/24,25,10.1.0.0/16
ipset test foo 192.168.1.1,80,10.0.0.1
```

hash:net,iface

The hash:net,iface set type uses a hash to store different sized IP network address and interface name pairs.

CREATE-OPTIONS := [family { inet | inet6 }] [hashsize value] [max? elem value] [bucketsize value] [timeout value] [counters] [com? ment] [skbinfo]

ADD-ENTRY := netaddr,[physdev:]iface

ADD-OPTIONS := [timeout value] [nomatch] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value] [wildcard]

DEL-ENTRY := netaddr,[physdev:]iface

TEST-ENTRY := netaddr,[physdev:]iface

where netaddr := ip[/cidr]

For the netaddr part of the elements see the description at the hash:net set type.

When adding/deleting/testing entries, if the cidr prefix parameter is not specified, then the host prefix value is assumed. When adding/deleting entries, the exact element is added/deleted and over? lapping elements are not checked by the kernel. When testing entries, if a host address is tested, then the kernel tries to match the host address in the networks added to the set and reports the result accord?

ingly.

From the set netfilter match point of view the searching for a match always starts from the smallest size of netblock (most specific prefix) to the largest one (least specific prefix) added to the set.

When adding/deleting IP addresses to the set by the SET netfilter target, it will be added/deleted by the most specific prefix which can be found in the set, or by the host prefix value if the set is empty.

The second direction parameter of the set match and SET target modules corresponds to the incoming/outgoing interface: src to the incoming one (similar to the -i flag of iptables), while dst to the outgoing one (similar to the -o flag of iptables). When the interface is flagged with physdev:, the interface is interpreted as the incoming/outgoing bridge port.

The lookup time grows linearly with the number of the different prefix values added to the set.

The internal restriction of the hash:net,iface set type is that the same network prefix cannot be stored with more than 64 different inter? faces in a single set.

Examples:

```
ipset create foo hash:net,iface
ipset add foo 192.168.0/24,eth0
ipset add foo 10.1.0.0/16,eth1
ipset test foo 192.168.0/24,eth0
```

list:set

The list:set type uses a simple list in which you can store set names.

CREATE-OPTIONS := [size value] [timeout value] [counters] [com? ment] [skbinfo]

ADD-ENTRY := setname [{ before | after } setname]

ADD-OPTIONS := [timeout value] [packets value] [bytes value] [comment string] [skbmark value] [skbprio value] [skbqueue value]

DEL-ENTRY := setname [{ before | after } setname]

TEST-ENTRY := setname [{ before | after } setname]

Optional create options:

size value

The size of the list, the default is 8. The parameter is ignored since ipset version 6.24.

By the ipset command you can add, delete and test set names in a list:set type of set.

By the set match or SET target of netfilter you can test, add or delete entries in the sets added to the list:set type of set. The match will try to find a matching entry in the sets and the target will try to add an entry to the first set to which it can be added. The number of direction options of the match and target are important: sets which require more parameters than specified are skipped, while sets with equal or less parameters are checked, elements added/deleted. For example if a and b are list:set type of sets then in the command

iptables -m set --match-set a src,dst -j SET --add-set b src,dst

the match and target will skip any set in a and b which stores data triples, but will match all sets with single or double data storage in a set and stop matching at the first successful set, and add src to the first single or src,dst to the first double data storage set in b to which the entry can be added. You can imagine a list:set type of set as an ordered union of the set elements.

Please note: by the ipset command you can add, delete and test the set names in a list:set type of set, and not the presence of a set's member (such as an IP address).

GENERAL RESTRICTIONS

Zero valued set entries cannot be used with hash methods. Zero protocol value with ports cannot be used.

COMMENTS

If you want to store same size subnets from a given network (say /24 blocks from a /8 network), use the bitmap:ip set type. If you want to store random same size networks (say random /24 blocks), use the hash:ip set type. If you have got random size of netblocks, use hash:net.

Matching on destination MAC addresses using the dst parameter of the set match netfilter kernel modules will only work if the destination MAC address is available in the packet at the given processing stage, that is, it only applies for incoming packets in the PREROUTING, INPUT and FORWARD chains, against the MAC address as originally found in the received packet (typically, one of the MAC addresses of the local host). This is not the destination MAC address a destination IP address resolves to, after routing. If the MAC address is not available (e.g. in the OUTPUT chain), the packet will simply not match.

Backward compatibility is maintained and old ipset syntax is still sup? ported.

The iptree and iptreemap set types are removed: if you refer to them, they are automatically replaced by hash:ip type of sets.

DIAGNOSTICS

Various error messages are printed to standard error. The exit code is 0 for correct functioning.

BUGS

Bugs? No, just funny features. :-) OK, just kidding...

SEE ALSO

iptables(8), ip6tables(8), iptables-extensions(8), nft(8)

AUTHORS

Jozsef Kadlecsik wrote ipset, which is based on ippool by Joakim Axels? son, Patrick Schaaf and Martin Josefsson.

Sven Wegener wrote the iptreemap type.

LAST REMARK

I stand on the shoulders of giants.

Jozsef Kadlecsik Jun 25, 2015 IPSET(8)