Rocky Enterprise Linux 9.2 Manual Pages on command 'libnftables-json.5'

$ man libnftables-json.5

LIBNFTABLES-JSON(5) LIBNFTABLES-JSON(5)

NAME

libnftables-json - Supported JSON schema by libnftables

SYNOPSIS

{ "nftables": [OBJECTS] }

OBJECTS := LIST_OBJECTS | CMD_OBJECTS

LIST_OBJECTS := LIST_OBJECT [, LIST_OBJECTS]

CMD_OBJECTS := CMD_OBJECT [, CMD_OBJECTS]

CMD_OBJECT := { CMD: LIST_OBJECT } | METAINFO_OBJECT

CMD := "add" | "replace" | "create" | "insert" | "delete" | "list" |

"reset" | "flush" | "rename"

LIST_OBJECT := TABLE | CHAIN | RULE | SET | MAP | ELEMENT | FLOWTABLE |

COUNTER | QUOTA | CT_HELPER | LIMIT | METAINFO_OBJECT | CT_TIMEOUT |

CT_EXPECTATION

DESCRIPTION

libnftables supports JSON formatted input and output. This is

implemented as an alternative frontend to the standard CLI syntax

parser, therefore basic behaviour is identical and, for (almost) any

operation available in standard syntax, there should be an equivalent one in JSON.

JSON input may be provided in a single string as parameter to nft_run_cmd_from_buffer() or in a file identified by the filename parameter of the nft_run_cmd_from_filename() function.

JSON output has to be enabled via the nft_ctx_output_set_json() function, turning library standard output into JSON format. Error output remains unaffected.

GLOBAL STRUCTURE

In general, any JSON input or output is enclosed in an object with a single property named nftables. Its value is an array containing commands (for input) or ruleset elements (for output).

A command is an object with a single property whose name identifies the command. Its value is a ruleset element - basically identical to output elements, apart from certain properties which may be interpreted differently or are required when output generally omits them.

METAINFO OBJECT

In output, the first object in an nftables array is a special one containing library information. Its content is as follows:

```
{ "metainfo": {
        "version": STRING,
        "release_name": STRING,
        "json_schema_version": NUMBER
}}
```

The values of version and release_name properties are equal to the package version and release name as printed by nft -v. The value of the json_schema_version property is an integer indicating the schema version.

If supplied in library input, the parser will verify the json_schema_version value to not exceed the internally hardcoded one (to make sure the given schema is fully understood). In future, a lower number than the internal one may activate compatibility mode to parse outdated and incompatible JSON input.

COMMAND OBJECTS

The structure accepts an arbitrary amount of commands which are interpreted in order of appearance. For instance, the following standard syntax input:

```
flush ruleset
add table inet mytable
add chain inet mytable mychain
add rule inet mytable mychain tcp dport 22 accept
```

translates into JSON as such:

```
{ "nftables": [
        { "flush": { "ruleset": null }},
        { "add": { "table": {
                        "family": "inet",
                        "name": "mytable"
        }}},
        { "add": { "chain": {
                        "family": "inet",
                        "table": "mytable",
                        "name": "mychain"
        }}},
        { "add": { "rule": {
                        "family": "inet",
                        "table": "mytable",
                        "chain": "mychain",
                        "expr": [
                                { "match": {
                                        "op": "==",
                                        "left": { "payload": {
                                                        "protocol": "tcp",
                                                        "field": "dport"
                                        }},
                                        "right": 22
                                }},
```

```
                    { "accept": null }
                ]
            }}}
        ]}
```

ADD

```
    { "add": ADD_OBJECT }
    ADD_OBJECT := TABLE | CHAIN | RULE | SET | MAP | ELEMENT |
                  FLOWTABLE | COUNTER | QUOTA | CT_HELPER | LIMIT |
                  CT_TIMEOUT | CT_EXPECTATION
```

Add a new ruleset element to the kernel.

REPLACE

```
    { "replace": RULE }
```

Replace a rule. In RULE, the handle property is mandatory and identifies the rule to be replaced.

CREATE

```
    { "create": ADD_OBJECT }
```

Identical to add command, but returns an error if the object already exists.

INSERT

```
    { "insert": RULE }
```

This command is identical to add for rules, but instead of appending the rule to the chain by default, it inserts at first position. If a handle or index property is given, the rule is inserted before the rule identified by those properties.

DELETE

```
    { "delete": ADD_OBJECT }
```

Delete an object from the ruleset. Only the minimal number of properties required to uniquely identify an object is generally needed in ADD_OBJECT. For most ruleset elements, this is family and table plus either handle or name (except rules since they don?t have a name).

LIST

```
    { "list": LIST_OBJECT }
    LIST_OBJECT := TABLE | TABLES | CHAIN | CHAINS | SET | SETS |
```

MAP | MAPS | COUNTER | COUNTERS | QUOTA | QUOTAS |

CT_HELPER | CT_HELPERS | LIMIT | LIMITS | RULESET |

METER | METERS | FLOWTABLE | FLOWTABLES |

CT_TIMEOUT | CT_EXPECTATION

List ruleset elements. The plural forms are used to list all objects of

that kind, optionally filtered by family and for some, also table.

RESET

{ "reset": RESET_OBJECT }

RESET_OBJECT := COUNTER | COUNTERS | QUOTA | QUOTAS

Reset state in suitable objects, i.e. zero their internal counter.

FLUSH

{ "flush": FLUSH_OBJECT }

FLUSH_OBJECT := TABLE | CHAIN | SET | MAP | METER | RULESET

Empty contents in given object, e.g. remove all chains from given table

or remove all elements from given set.

RENAME

{ "rename": CHAIN }

Rename a chain. The new name is expected in a dedicated property named

newname.

RULESET ELEMENTS

TABLE

{ "table": {

"family": STRING,

"name": STRING,

"handle": NUMBER

}}

This object describes a table.

family

The table?s family, e.g. "ip" or "ip6".

name

The table?s name.

handle

The table?s handle. In input, it is used only in delete command as

alternative to name.

CHAIN

{ "chain": {

"family": STRING,

"table": STRING,

"name": STRING,

"newname": STRING,

"handle": NUMBER,

"type": STRING,

"hook": STRING,

"prio": NUMBER,

"dev": STRING,

"policy": STRING

}}

This object describes a chain.

family

The table?s family.

table

The table?s name.

name

The chain?s name.

handle

The chain?s handle. In input, it is used only in delete command as

alternative to name.

newname

A new name for the chain, only relevant in the rename command.

The following properties are required for base chains:

type

The chain?s type.

hook

The chain?s hook.

prio

The chain?s priority.

dev

The chain?s bound interface (if in the netdev family).

policy

The chain?s policy.

RULE

```
{ "rule": {
        "family": STRING,
        "table": STRING,
        "chain": STRING,
        "expr": [ STATEMENTS ],
        "handle": NUMBER,
        "index": NUMBER,
        "comment": STRING
}}
STATEMENTS := STATEMENT [, STATEMENTS ]
```

This object describes a rule. Basic building blocks of rules are statements. Each rule consists of at least one.

family

The table?s family.

table

The table?s name.

chain

The chain?s name.

expr

An array of statements this rule consists of. In input, it is used in add/insert/replace commands only.

handle

The rule?s handle. In delete/replace commands, it serves as an identifier of the rule to delete/replace. In add/insert commands, it serves as an identifier of an existing rule to append/prepend the rule to.

index

The rule?s position for add/insert commands. It is used as an

alternative to handle then.

comment

Optional rule comment.

SET / MAP

```
{ "set": {
        "family": STRING,
        "table": STRING,
        "name": STRING,
        "handle": NUMBER,
        "type": SET_TYPE,
        "policy": SET_POLICY,
        "flags": [ SET_FLAG_LIST ],
        "elem": SET_ELEMENTS,
        "timeout": NUMBER,
        "gc-interval": NUMBER,
        "size": NUMBER
}}
{ "map": {
        "family": STRING,
        "table": STRING,
        "name": STRING,
        "handle": NUMBER,
        "type": SET_TYPE,
        "map": STRING,
        "policy": SET_POLICY,
        "flags": [ SET_FLAG_LIST ],
        "elem": SET_ELEMENTS,
        "timeout": NUMBER,
        "gc-interval": NUMBER,
        "size": NUMBER
}}
```

SET_TYPE := STRING | [SET_TYPE_LIST]

SET_TYPE_LIST := STRING [, SET_TYPE_LIST]

SET_POLICY := "performance" | "memory"

SET_FLAG_LIST := SET_FLAG [, SET_FLAG_LIST]

SET_FLAG := "constant" | "interval" | "timeout"

SET_ELEMENTS := EXPRESSION | [EXPRESSION_LIST]

EXPRESSION_LIST := EXPRESSION [, EXPRESSION_LIST]

These objects describe a named set or map. Maps are a special form of sets in that they translate a unique key to a value.

family

The table?s family.

table

The table?s name.

name

The set?s name.

handle

The set?s handle. For input, it is used in the delete command only.

type

The set?s datatype, see below.

map

Type of values this set maps to (i.e. this set is a map).

policy

The set?s policy.

flags

The set?s flags.

elem

Initial set element(s), see below.

timeout

Element timeout in seconds.

gc-interval

Garbage collector interval in seconds.

size

Maximum number of elements supported.

TYPE

The set type might be a string, such as "ipv4_addr" or an array

consisting of strings (for concatenated types).

ELEM

A single set element might be given as string, integer or boolean value for simple cases. If additional properties are required, a formal elem object may be used.

Multiple elements may be given in an array.

ELEMENT

```
{ "element": {
    "family": STRING,
    "table": STRING,
    "name": STRING,
    "elem": SET_ELEM
}}
```

SET_ELEM := EXPRESSION | [EXPRESSION_LIST]

EXPRESSION_LIST := EXPRESSION [, EXPRESSION]

Manipulate element(s) in a named set.

family

The table?s family.

table

The table?s name.

name

The set?s name.

elem

See elem property of set object.

FLOWTABLE

```
{ "flowtable": {
    "family": STRING,
    "table": STRING,
    "name": STRING,
    "handle": NUMBER,
    "hook": STRING,
    "prio": NUMBER,
    "dev": FT_INTERFACE
```

}}

FT_INTERFACE := STRING | [FT_INTERFACE_LIST]

FT_INTERFACE_LIST := STRING [, STRING]

This object represents a named flowtable.

family

The table?s family.

table

The table?s name.

name

The flow table?s name.

handle

The flow table?s handle. In input, it is used by the delete command only.

hook

The flow table?s hook.

prio

The flow table?s priority.

dev

The flow table?s interface(s).

COUNTER

```
{ "counter": {
        "family": STRING,
        "table": STRING,
        "name": STRING,
        "handle": NUMBER,
        "packets": NUMBER,
        "bytes": NUMBER
}}
```

This object represents a named counter.

family

The table?s family.

table

The table?s name.

name

The counter?s name.

handle

The counter?s handle. In input, it is used by the delete command only.

packets

Packet counter value.

bytes

Byte counter value.

QUOTA

```
{ "quota": {
        "family": STRING,
        "table": STRING,
        "name": STRING,
        "handle": NUMBER,
        "bytes": NUMBER,
        "used": NUMBER,
        "inv": BOOLEAN
}}
```

This object represents a named quota.

family

The table?s family.

table

The table?s name.

name

The quota?s name.

handle

The quota?s handle. In input, it is used by the delete command only.

bytes

Quota threshold.

used

Quota used so far.

inv

If true, match if the quota has been exceeded.

CT HELPER

{ "ct helper": {

"family": STRING,

"table": STRING,

"name": STRING,

"handle": ... ',

"type": 'STRING,

"protocol": CTH_PROTO,

"l3proto": STRING

}}

CTH_PROTO := "tcp" | "udp"

This object represents a named conntrack helper.

family

The table?s family.

table

The table?s name.

name

The ct helper?s name.

handle

The ct helper?s handle. In input, it is used by the delete command

only.

type

The ct helper type name, e.g. "ftp" or "tftp".

protocol

The ct helper?s layer 4 protocol.

l3proto

The ct helper?s layer 3 protocol, e.g. "ip" or "ip6".

LIMIT

{ "limit": {

"family": STRING,

"table": STRING,

"name": STRING,

"handle": NUMBER,

"rate": NUMBER,

"per": STRING,

"burst": NUMBER,

"unit": LIMIT_UNIT,

"inv": BOOLEAN

}}

LIMIT_UNIT := "packets" | "bytes"

This object represents a named limit.

family

The table?s family.

table

The table?s name.

name

The limit?s name.

handle

The limit?s handle. In input, it is used by the delete command only.

rate

The limit?s rate value.

per

Time unit to apply the limit to, e.g. "week", "day", "hour", etc.

If omitted, defaults to "second".

burst

The limit?s burst value. If omitted, defaults to 0.

unit

Unit of rate and burst values. If omitted, defaults to "packets".

inv

If true, match if limit was exceeded. If omitted, defaults to false.

CT TIMEOUT

{ "ct timeout": {

"family": STRING,

"table": STRING,

"name": STRING,

"handle": NUMBER,

"protocol": CTH_PROTO,

"state": STRING,

"value: NUMBER,

"l3proto": STRING

}}

CTH_PROTO := "tcp" | "udp" | "dccp" | "sctp" | "gre" | "icmpv6" | "icmp" | "generic"

This object represents a named conntrack timeout policy.

family

The table?s family.

table

The table?s name.

name

The ct timeout object?s name.

handle

The ct timeout object?s handle. In input, it is used by delete

command only.

protocol

The ct timeout object?s layer 4 protocol.

state

The connection state name, e.g. "established", "syn_sent", "close"

or "close_wait", for which the timeout value has to be updated.

value

The updated timeout value for the specified connection state.

l3proto

The ct timeout object?s layer 3 protocol, e.g. "ip" or "ip6".

CT EXPECTATION

{ "ct expectation": {

"family": STRING,

"table": STRING,

"name": STRING,

"handle": NUMBER,

"l3proto": STRING

"protocol":* CTH_PROTO,

"dport": NUMBER,

"timeout: NUMBER,

"size: NUMBER,

*}}

CTH_PROTO := "tcp" | "udp" | "dccp" | "sctp" | "gre" | "icmpv6" | "icmp" | "generic"

This object represents a named conntrack expectation.

family

The table?s family.

table

The table?s name.

name

The ct expectation object?s name.

handle

The ct expectation object?s handle. In input, it is used by delete

command only.

l3proto

The ct expectation object?s layer 3 protocol, e.g. "ip" or "ip6".

protocol

The ct expectation object?s layer 4 protocol.

dport

The destination port of the expected connection.

timeout

The time in millisecond that this expectation will live.

size

The maximum count of expectations to be living in the same time.

STATEMENTS

Statements are the building blocks for rules. Each rule consists of at

least one.

VERDICT

{ "accept": null }

{ "drop": null }

{ "continue": null }

{ "return": null }

{ "jump": { "target": * STRING *}}

{ "goto": { "target": * STRING *}}

A verdict either terminates packet traversal through the current chain or delegates to a different one.

jump and goto statements expect a target chain name.

MATCH

{ "match": {

"left": EXPRESSION,

"right": EXPRESSION,

"op": STRING

}}

This matches the expression on left hand side (typically a packet header or packet meta info) with the expression on right hand side (typically a constant value). If the statement evaluates to true, the next statement in this rule is considered. If not, processing continues with the next rule in the same chain.

left

Left hand side of this match.

right

Right hand side of this match.

op

Operator indicating the type of comparison.

OPERATORS

& Binary AND

| Binary OR

^ Binary XOR

<< Left shift

>> Right shift

== Equal

!= Not equal

< Less than

> Greater than

? Less than or equal to

>= Greater than or equal to

in Perform a lookup, i.e.
test if bits on RHS are
contained in LHS value

Unlike with the standard API, the operator is mandatory here. In the standard API, a missing operator may be resolved in two ways, depending on the type of expression on the RHS:

? If the RHS is a bitmask or a list of bitmasks, the expression resolves into a binary operation with the inequality operator, like this: LHS & RHS != 0.

? In any other case, the equality operator is simply inserted.

For the non-trivial first case, the JSON API supports the in operator.

COUNTER

```
{ "counter": {
        "packets": NUMBER,
        "bytes": NUMBER
}}
{ "counter": STRING }
```

This object represents a byte/packet counter. In input, no properties are required. If given, they act as initial values for the counter.

The first form creates an anonymous counter which lives in the rule it appears in. The second form specifies a reference to a named counter object.

packets
Packets counted.

bytes
Bytes counted.

MANGLE

{ "mangle": {

"key": EXPRESSION,

"value": EXPRESSION

}}

This changes the packet data or meta info.

key

The packet data to be changed, given as an exthdr, payload, meta, ct or ct helper expression.

value

Value to change data to.

QUOTA

{ "quota": {

"val": NUMBER,

"val_unit": STRING,

"used": NUMBER,

"used_unit": STRING,

"inv": BOOLEAN

}}

{ "quota": STRING }

The first form creates an anonymous quota which lives in the rule it appears in. The second form specifies a reference to a named quota object.

val

Quota value.

val_unit

Unit of val, e.g. "kbytes" or "mbytes". If omitted, defaults to "bytes".

used

Quota used so far. Optional on input. If given, serves as initial value.

used_unit

Unit of used. Defaults to "bytes".

inv

If true, will match if quota was exceeded. Defaults to false.

LIMIT

```
{ "limit": {
        "rate": NUMBER,
        "rate_unit": STRING,
        "per": STRING,
        "burst": NUMBER,
        "burst_unit": STRING,
        "inv": BOOLEAN
}}
{ "limit": STRING }
```

The first form creates an anonymous limit which lives in the rule it appears in. The second form specifies a reference to a named limit object.

rate

Rate value to limit to.

rate_unit

Unit of rate, e.g. "packets" or "mbytes". Defaults to "packets".

per

Denominator of rate, e.g. "week" or "minutes".

burst

Burst value. Defaults to 0.

burst_unit

Unit of burst, ignored if rate_unit is "packets". Defaults to "bytes".

inv

If true, matches if the limit was exceeded. Defaults to false.

FWD

```
{ "fwd": {
        "dev": EXPRESSION,
        "family": FWD_FAMILY,
        "addr": EXPRESSION
}}
```

FWD_FAMILY := "ip" | "ip6"

Forward a packet to a different destination.

dev

Interface to forward the packet on.

family

Family of addr.

addr

IP(v6) address to forward the packet to.

Both family and addr are optional, but if at least one is given, both must be present.

NOTRACK

{ "notrack": null }

Disable connection tracking for the packet.

DUP

{ "dup": {
"addr": EXPRESSION,
"dev": EXPRESSION
}}

Duplicate a packet to a different destination.

addr

Address to duplicate packet to.

dev

Interface to duplicate packet on. May be omitted to not specify an interface explicitly.

NETWORK ADDRESS TRANSLATION

{ "snat": {
"addr": EXPRESSION,
"family": STRING,
"port": EXPRESSION,
"flags": FLAGS
}}

{ "dnat": {
"addr": EXPRESSION,

"family": STRING,

"port": EXPRESSION,

"flags": FLAGS

}}

{ "masquerade": {

"port": EXPRESSION,

"flags": FLAGS

}}

{ "redirect": {

"port": EXPRESSION,

"flags": FLAGS

}}

FLAGS := FLAG | [FLAG_LIST]

FLAG_LIST := FLAG [, FLAG_LIST]

FLAG := "random" | "fully-random" | "persistent"

Perform Network Address Translation.

addr

Address to translate to.

family

Family of addr, either ip or ip6. Required in inet table family.

port

Port to translate to.

flags

Flag(s).

All properties are optional and default to none.

REJECT

{ "reject": {

"type": STRING,

"expr": EXPRESSION

}}

Reject the packet and send the given error reply.

type

Type of reject, either "tcp reset", "icmpx", "icmp" or "icmpv6".

expr

ICMP code to reject with.

All properties are optional.

SET

{ "set": {

"op": STRING,

"elem": EXPRESSION,

"set": STRING

}}

Dynamically add/update elements to a set.

op

Operator on set, either "add" or "update".

elem

Set element to add or update.

set

Set reference.

LOG

{ "log": {

"prefix": STRING,

"group": NUMBER,

"snaplen": NUMBER,

"queue-threshold": NUMBER,

"level": LEVEL,

"flags": FLAGS

}}

LEVEL := "emerg" | "alert" | "crit" | "err" | "warn" | "notice" |

"info" | "debug" | "audit"

FLAGS := FLAG | [FLAG_LIST]

FLAG_LIST := FLAG [, FLAG_LIST]

FLAG := "tcp sequence" | "tcp options" | "ip options" | "skuid" |

"ether" | "all"

Log the packet.

prefix

Prefix for log entries.

group

Log group.

snaplen

Snaplen for logging.

queue-threshold

Queue threshold.

level

Log level. Defaults to "warn".

flags

Log flags.

All properties are optional.

CT HELPER

{ "ct helper": EXPRESSION }

Enable the specified conntrack helper for this packet.

ct helper

CT helper reference.

METER

```
{ "meter": {
        "name": STRING,
        "key": EXPRESSION,
        "stmt": STATEMENT
}}
```

Apply a given statement using a meter.

name

Meter name.

key

Meter key.

stmt

Meter statement.

QUEUE

```
{ "queue": {
        "num": EXPRESSION,
```

"flags": FLAGS

}}

FLAGS := FLAG | [FLAG_LIST]

FLAG_LIST := FLAG [, FLAG_LIST]

FLAG := "bypass" | "fanout"

Queue the packet to userspace.

num

Queue number.

flags

Queue flags.

VERDICT MAP

{ "vmap": {

"key": EXPRESSION,

"data": EXPRESSION

}}

Apply a verdict conditionally.

key

Map key.

data

Mapping expression consisting of value/verdict pairs.

CT COUNT

{ "ct count": {

"val": NUMBER,

"inv": BOOLEAN

}}

Limit the number of connections using conntrack.

val

Connection count threshold.

inv

If true, match if val was exceeded. If omitted, defaults to false.

CT TIMEOUT

{ "ct timeout": EXPRESSION }

Assign connection tracking timeout policy.

ct timeout

CT timeout reference.

CT EXPECTATION

{ "ct expectation": EXPRESSION }

Assign connection tracking expectation.

ct expectation

CT expectation reference.

XT

{ "xt": null }

This represents an xt statement from xtables compat interface. Sadly, at this point, it is not possible to provide any further information about its content.

EXPRESSIONS

Expressions are the building blocks of (most) statements. In their most basic form, they are just immediate values represented as a JSON string, integer or boolean type.

IMMEDIATES

STRING

NUMBER

BOOLEAN

Immediate expressions are typically used for constant values. For strings, there are two special cases:

@STRING

The remaining part is taken as set name to create a set reference.

*

Construct a wildcard expression.

LISTS

ARRAY

List expressions are constructed by plain arrays containing of an arbitrary number of expressions.

CONCAT

{ "concat": CONCAT }

CONCAT := [EXPRESSION_LIST]

EXPRESSION_LIST := EXPRESSION [, EXPRESSION_LIST]

Concatenate several expressions.

SET

{ "set": SET }

SET := EXPRESSION | [EXPRESSION_LIST]

This object constructs an anonymous set. For mappings, an array of arrays with exactly two elements is expected.

MAP

```
{ "map": {
        "key": EXPRESSION,
        "data": EXPRESSION
}}
```

Map a key to a value.

key

Map key.

data

Mapping expression consisting of value/target pairs.

PREFIX

```
{ "prefix": {
        "addr": EXPRESSION,
        "len": NUMBER
}}
```

Construct an IPv4 or IPv6 prefix consisting of address part in addr and prefix length in len.

RANGE

{ "range": [EXPRESSION , EXPRESSION] }

Construct a range of values. The first array item denotes the lower boundary, the second one the upper boundary.

PAYLOAD

```
{ "payload": {
        "base": BASE,
        "offset": NUMBER,
        "len": NUMBER
```

}}

{ "payload": {

"protocol": STRING,

"field": STRING

}}

BASE := "ll" | "nh" | "th"

Construct a payload expression, i.e. a reference to a certain part of packet data. The first form creates a raw payload expression to point at a random number (len) of bytes at a certain offset (offset) from a given reference point (base). The following base values are accepted:

"ll"

The offset is relative to Link Layer header start offset.

"nh"

The offset is relative to Network Layer header start offset.

"th"

The offset is relative to Transport Layer header start offset.

The second form allows to reference a field by name (field) in a named packet header (protocol).

EXTHDR

{ "exthdr": {

"name": STRING,

"field": STRING,

"offset": NUMBER

}}

Create a reference to a field (field) in an IPv6 extension header (name). offset is used only for rt0 protocol.

If the field property is not given, the expression is to be used as a header existence check in a match statement with a boolean on the right hand side.

TCP OPTION

{ "tcp option": {

"name": STRING,

"field": STRING

}}

Create a reference to a field (field) of a TCP option header (name).

If the field property is not given, the expression is to be used as a TCP option existence check in a match statement with a boolean on the right hand side.

SCTP CHUNK

{ "sctp chunk": {
"name": STRING,
"field": STRING
}}

Create a reference to a field (field) of an SCTP chunk (name).

If the field property is not given, the expression is to be used as an SCTP chunk existence check in a match statement with a boolean on the right hand side.

META

{ "meta": {
"key": META_KEY
}}

META_KEY := "length" | "protocol" | "priority" | "random" | "mark" | "iif" | "iifname" | "iiftype" | "oif" | "oifname" | "oiftype" | "skuid" | "skgid" | "nftrace" | "rtclassid" | "ibriport" | "obriport" | "ibridgename" | "obridgename" | "pkttype" | "cpu" | "iifgroup" | "oifgroup" | "cgroup" | "nfproto" | "l4proto" | "secpath"

Create a reference to packet meta data.

RT

{ "rt": {
"key": RT_KEY,
"family": RT_FAMILY
}}

RT_KEY := "classid" | "nexthop" | "mtu"

RT_FAMILY := "ip" | "ip6"

Create a reference to packet routing data.

The family property is optional and defaults to unspecified.

CT

```
{ "ct": {
        "key": STRING,
        "family": CT_FAMILY,
        "dir": CT_DIRECTION
}}
CT_FAMILY := "ip" | "ip6"
CT_DIRECTION := "original" | "reply"
```

Create a reference to packet conntrack data.

Some CT keys do not support a direction. In this case, dir must not be given.

NUMGEN

```
{ "numgen": {
        "mode": NG_MODE,
        "mod": NUMBER,
        "offset": NUMBER
}}
NG_MODE := "inc" | "random"
```

Create a number generator.

The offset property is optional and defaults to 0.

HASH

```
{ "jhash": {
        "mod": NUMBER,
        "offset": NUMBER,
        "expr": EXPRESSION,
        "seed": NUMBER
}}
{ "symhash": {
        "mod": NUMBER,
        "offset": NUMBER
}}
```

Hash packet data.

The offset and seed properties are optional and default to 0.

FIB

{ "fib": {

"result": FIB_RESULT,

"flags": FIB_FLAGS

}}

FIB_RESULT := "oif" | "oifname" | "type"

FIB_FLAGS := FIB_FLAG | [FIB_FLAG_LIST]

FIB_FLAG_LIST := FIB_FLAG [, FIB_FLAG_LIST]

FIB_FLAG := "saddr" | "daddr" | "mark" | "iif" | "oif"

Perform kernel Forwarding Information Base lookups.

BINARY OPERATION

{ "|": [EXPRESSION, EXPRESSION] }

{ "^": [EXPRESSION, EXPRESSION] }

{ "&": [EXPRESSION, EXPRESSION] }

{ "<<": [EXPRESSION, EXPRESSION] }

{ ">>": [EXPRESSION, EXPRESSION] }

All binary operations expect an array of exactly two expressions, of

which the first element denotes the left hand side and the second one

the right hand side.

VERDICT

{ "accept": null }

{ "drop": null }

{ "continue": null }

{ "return": null }

{ "jump": { "target": STRING }}

{ "goto": { "target": STRING }}

Same as the verdict statement, but for use in verdict maps.

jump and goto verdicts expect a target chain name.

ELEM

{ "elem": {

"val": EXPRESSION,

```
        "timeout": NUMBER,
        "expires": NUMBER,
        "comment": STRING
    }}
```

Explicitly set element object, in case timeout, expires or comment are desired. Otherwise, it may be replaced by the value of val.

SOCKET

```
    { "socket": {
        "key": SOCKET_KEY
    }}
    SOCKET_KEY := "transparent"
```

Construct a reference to packet?s socket.

OSF

```
    { "osf": {
        "key": OSF_KEY,
        "ttl": OSF_TTL
    }}
    OSF_KEY := "name"
    OSF_TTL := "loose" | "skip"
```

Perform OS fingerprinting. This expression is typically used in the LHS of a match statement.

key

Which part of the fingerprint info to match against. At this point, only the OS name is supported.

ttl

Define how the packet?s TTL value is to be matched. This property is optional. If omitted, the TTL value has to match exactly. A value of loose accepts TTL values less than the fingerprint one. A value of skip omits TTL value comparison entirely.

AUTHOR

Phil Sutter <phil@nwl.cc>

Author.