Implementing Access Lists and Prefix Lists

Understanding Access Lists

Access lists perform packet filtering to control which packets move through the network and where. Such controls help to limit network traffic and restrict the access of users and devices to the network. Access lists have many uses, and therefore many commands accept a reference to an access list in their command syntax. Access lists can be used to do the following:

An access control list (ACL) consists of one or more access control entries (ACE) that collectively define the network traffic profile. This profile can then be referenced by Cisco IOS XR software features such as traffic filtering, route filtering, QoS classification, and access control. There are 2 types of ACLs:

- Standard ACLs- Verifies only the source IP address of the packets. Traffic is controlled by the comparison of the address or prefix configured in the ACL, with the source address found in the packet.
- Extended ACLs- Verifies more than just the source address of the packets. Attributes such as destination address, specific IP protocols, UDP or TCP port numbers, DSCP, and so on are validated. Traffic is

controlled by a comparison of the attributes stated in the ACL with those in the incoming or outgoing packets.

Cisco IOS XR does not differentiate between standard and extended access lists. Standard access list support is provided for backward compatibility.

Purpose of IP Access Lists

- Filter incoming or outgoing packets on an interface.
- Filter packets for mirroring.
- Redirect traffic as required.
- Restrict the contents of routing updates.
- Limit debug output based on an address or protocol.
- Control vty access.
- Identify or classify traffic for advanced features, such as congestion avoidance, congestion management, and priority and custom queueing.

How an IP Access List Works

An access list is a sequential list consisting of permit and deny statements that apply to IP addresses and possibly upper-layer IP protocols. The access list has a name by which it is referenced. Many software commands accept an access list as part of their syntax.

An access list can be configured and named, but it is not in effect until the access list is referenced by a command that accepts an access list. Multiple commands can reference the same access list. An access list can control traffic arriving at the router or leaving the router, but not traffic originating at the router.

Source address and destination addresses are two of the most typical fields in an IP packet on which to base an access list. Specify source addresses to control packets from certain networking devices or hosts. Specify destination addresses to control packets being sent to certain networking devices or hosts.

You can also filter packets on the basis of transport layer information, such as whether the packet is a TCP, UDP, ICMP, or IGMP packet.

ACL Workflow

The following image illustrates the workflow of an ACL.

IP Access List Process and Rules

Use the following process and rules when configuring an IP access list:

- The software tests the source or destination address or the protocol of each packet being filtered against the conditions in the access list, one condition (permit or deny statement) at a time.
- If a packet does not match an access list statement, the packet is then tested against the next statement in the list.
- If a packet and an access list statement match, the remaining statements in the list are skipped and the packet is permitted or denied as specified in the matched statement. The first entry that the packet matches

determines whether the software permits or denies the packet. That is, after the first match, no subsequent entries are considered.

- If the access list denies the address or protocol, the software discards the packet and returns an Internet Control Message Protocol (ICMP) Host Unreachable message. ICMP is configurable in the Cisco IOS XR software.
- If no conditions match, the software drops the packet because each access list ends with an unwritten or implicit deny statement. That is, if the packet has not been permitted or denied by the time it was tested against each statement, it is denied.
- The access list should contain at least one permit statement or else all packets are denied.
- Because the software stops testing conditions after the first match, the order of the conditions is critical. The same permit or deny statements specified in a different order could result in a packet being passed under one circumstance and denied in another circumstance.
- Only one access list per interface, per protocol, per direction is allowed.
- Inbound access lists process packets arriving at the router. Incoming packets are processed before being routed to an outbound interface. An inbound access list is efficient because it saves the overhead of routing lookups if the packet is to be discarded because it is denied by the filtering tests. If the packet is permitted by the tests, it is then processed for routing. For inbound lists, permit means continue to process the packet after receiving it on an inbound interface; **deny** means discard the packet.
- Outbound access lists process packets before they leave the router. Incoming packets are routed to the outbound interface and then processed through the outbound access list. For outbound lists, permit means send it to the output buffer; deny means discard the packet.
- An access list can not be removed if that access list is being applied by an access group in use. To remove an access list, remove the access group that is referencing the access list and then remove the access list.
- Before removing an interface, which is configured with an ACL that denies certain traffic, you must remove the ACL and commit your configuration. If this is not done, then some packets are leaked through the interface as soon as the **no interface <interface-name>** command is configured and committed.
- An access list must exist before you can use the **ipv4 access group** command.

ACL Filtering by Wildcard Mask and Implicit Wildcard Mask

Address filtering uses wildcard masking to indicate whether the software checks or ignores corresponding IP address bits when comparing the address bits in an access-list entry to a packet being submitted to the access list. By carefully setting wildcard masks, an administrator can select a single or several IP addresses for permit or deny tests.

Wildcard masking for IP address bits uses the number 1 and the number 0 to specify how the software treats the corresponding IP address bits. A wildcard mask is sometimes referred to as an *inverted mask*, because a 1 and 0 mean the opposite of what they mean in a subnet (network) mask.

- A wildcard mask bit 0 means *check* the corresponding bit value.
- A wildcard mask bit 1 means *ignore* that corresponding bit value.

You do not have to supply a wildcard mask with a source or destination address in an access list statement. If you use the **host** keyword, the software assumes a wildcard mask of 0.0.0.0.

Unlike subnet masks, which require contiguous bits indicating network and subnet to be ones, wildcard masks allow noncontiguous bits in the mask.

You can also use CIDR format (/x) in place of wildcard bits. For example, the IPv4 address 1.2.3.4 0.255.255.255 corresponds to 1.2.3.4/8 and for IPv6 address 2001:db8:abcd:0012:0000:0000:0000:0000 corresponds to 2001:db8:abcd:0012::0/64.

Including Comments in Access Lists

You can include comments (remarks) about entries in any named IP access list using the remark access list configuration command. The remarks make the access list easier for the network administrator to understand and scan. Each remark line is limited to 255 characters.

The remark can go before or after a **permit** or **deny** statement. You should be consistent about where you put the remark so it is clear which remark describes which **permit** or **deny** statement. For example, it would be confusing to have some remarks before the associated **permit** or **deny** statements and some remarks after the associated statements. Remarks can be sequenced.

Remember to apply the access list to an interface or terminal line after the access list is created.

User-Defined TCAM Keys for IPv4 and IPv6

Access-lists on the use a TCAM (internal and external) to perform the lookup and action resolution on each packet. The TCAM is a valuable and constrained resource in hardware, which must be shared by multiple features. Therefore, the space (key width) available for these key definitions is also constrained. A key definition specifies which qualifier and action fields are available to the ACL feature when performing the lookup. Not all available qualifier and action fields can be included in each key definition.

The key definitions are specific to a given ACL type, which can depend on the following attributes of the access-list:

- Direction of attachment, whether ingress or egress
- Protocol type (IPv4/IPv6/L2)
- Compression level (0:uncompressed, 3:compressed)

Because the default key definitions are constrained (do not include all qualifier/action fields), User-Defined Key (UDK) definitions are supported for the following types:

- Traditional Ingress IPv4 ACL (uncompressed)
- Traditional Ingress IPv6 ACL (uncompressed)

The User-Defined TCAM Key (UDK) functionality provides the flexibility to define your own TCAM key for one of the three possible reasons (for ingress, traditional, Ipv4/Ipv6 ACL only):

To include qualifier fields which are not included in the default TCAM key

To change the ACL mode from *shared* to *unique* to support a greater number of unique ACLs, unique counters, etc.

To reduce the size of the TCAM key (number of banks consumed)

A UDK can be configured using the following command:

```
hw-module profile tcam format access-list [ipv4 | ipv6] qualifiers [location rack/slot/cpu0]
```

User-Defined Fields

A TCAM key consists of several qualifiers, where the set of qualifiers are used to filter packets for a given ACL. The User-Defined Field (UDF) allows you to define a custom qualifier by specifying the location and size of the field, using the following UDF command:

```
udf udf-name header [ inner | outer ] [ l2 | l3 | l4 ] offset byte-offset length no of bytes
```

The UDF can then be added to a UDK as follows.

```
hw-module profile tcam format access-list [ipv4 | ipv6] qualifiers [udf1 udf-name udf2 udf-name] [location rack/slot/cpu0]
```

Note Up to 8 UDFs can be defined system wide. Currently, UDFs are globally defined.

IPv4 and IPv6 Key Formats for Traditional Ingress ACL

User-defined TCAM key (UDK) definition is supported for ingress, traditional (uncompressed) IPv4 and IPv6 ACLs.

The following table shows the qualifier fields that are supported in the IPv4 and IPv6 key formats. If the default TCAM key is set as *Enabled*, then the Qualifier field is enabled by default. If the default TCAM key is set as *Disabled*, then the Qualifier field must use a UDK.

Table 1: Qualifier Fields Supported in IPv4 and IPv6 Key Formats

Parameter	Default TCAM Key	
	IPv4	IPv6
Source Address	Enabled	Enabled
Destination Address	Enabled	Enabled
Source Port	Enabled	Enabled
Destination Port	Enabled	Enabled
Port Range	Enabled	Not supported
Protocol/Next Header	Enabled	Enabled
Fragment bit	Enabled	Not supported
Packet length	Disabled	Disabled
Precedence/DSCP	Disabled	Enabled
TCP Flags	Enabled	Enabled
TTL Match	Disabled	Disabled
Interface based	Disabled	Not supported

Parameter	Default TCAM Key	
	IPv4	IPv6
UDF 1-8	Disabled	Disabled
ACL ID	Enabled	Enabled
common ACL bit	Enabled by default for IPv4/IPv6 on shared mode. Disabled by default for IPv4/IPv6 on unique mode.	Enabled by default for IPv4/IPv6 on shared mode. Disabled by default for IPv4/IPv6 on unique mode.
Interface-based (RIF)	Disabled	Disabled

The following table shows the action fields supported in the IPv4 and IPv6 key formats.

Table 2: Action Fields Supported in IPv4 and IPv6 Key Formats

Parameter	Default Action Field	
	IPv4	IPv6
Permit	Enabled	Enabled
Deny	Enabled	Enabled
Log	Enabled	Enabled
Capture	Enabled	Enabled
Stats Counter	Deny stats is always Enabled (permit stats has its own hw-module command)	Deny stats is always Enabled
TTL Set	Enabled	Enabled

Note

- The Capture parameter is not supported on Jericho 2 ASIC line cards.
- For Jericho 2 ASIC line cards, both the Permit statsand Deny stats are always enabled. There is no need to use the hw-module command to enable Permit stats.

Configuring IPv4 ACLs

This section describes the basic configuration of IPv4 ingress and egress ACLs.

Notes and Restrictions for Configuring IPv4 Ingress ACLs

IPv4 ingress ACLs are characterized by the following behavior.

- Ingress IPv4 ACLs are supported on all interfaces except management interfaces.

- ACL-based Forwarding (ABF) is supported only in the ingress direction.
- The total number of ACLs allowed by default per NPU is 31.
- The number of attached ACEs allowed per line card is 4000.
- ACL logging with input interface (using the **log-input** keyword) is not supported.
- The **show access-lists ipv4 acl_name stats** command is not supported for ACE with logs, to check statistics. Therefore, use the **show access-lists** *acl-name* **hardware [ingress | egress] detail location***loc* command to check statistics for ACE with logs.
- Ingress ACL matching stats using **show access-lists ipv4**<*ACL name*>**hardware ingress location 0/RP0/CPU0** command may take more latency (up to 15 seconds) to reflect.

Notes and Restrictions for Configuring IPv4 Egress ACLs

IPv4 egress ACLs are characterized by the following behavior.

- Egress IPv4 ACLs are supported on main physical interfaces and bundle interfaces.

Note Egress ACLs are not directly supported on sub-interfaces. However, If you configure an egress ACL on a main interface that has sub-interfaces, the ACL action is also applied to the sub-interface traffic. This egress ACL behavior holds true even if the sub-interfaces are configured after the ACL is applied to the main interface.

- The total number of egress ACLs allowed per NPU is 255.
- ACL is not supported on Management interface on egress direction.
- The number of attached ACEs allowed per line card is 4000.
- ACL logging (using the **log** command) and ACL logging with input interface (using the **log-input** command) is not supported.

Configuring an Ingress IPv4 ACL on a Gigabit Ethernet Interface

Use the following configuration to configure an ingress IPv4 ACL on a GigE interface.

```
/* Configure a GigE interface with an IPv4 address */
Router(config)# interface TenGigE 0/11/0/0
Router(config-if)# ipv4 address 10.1.1.1 255.255.255.0
Router(config-if)# no shut
Router(config-if)# commit
Thu Jan 25 10:07:54.700 IST
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv4 interface brief
Thu Jan 25 10:08:49.087 IST

Interface                      IP-Address      Status          Protocol
Vrf-NameTenGigE0/11/0/0         10.1.1.1        Up              Up       default

/* Configure an IPv4 ingress ACL */
```

```
Router(config)# ipv4 access-list V4-ACL-INGRESS
Router(config-ipv4-acl)# 10 permit tcp 10.2.1.1 0.0.0.255 any
Router(config-ipv4-acl)# 20 deny udp any any
Router(config-ipv4-acl)# 30 permit ipv4 10.2.0.0 0.255.255.255 any
Router(config-ipv4-acl)# commit
Thu Jan 25 10:16:11.473 IST

/* Verify the ingress ACL creation */
Router(config)# do show access-lists ipv4
Thu Jan 25 10:25:19.896 IST
...
ipv4 access-list V4-ACL-INGRESS
 10 permit tcp 10.2.1.0 0.0.0.255 any
 20 deny udp any any
 30 permit ipv4 10.0.0.0 0.255.255.255 any


/* Apply the ingress ACL to the GigE interface */
Router(config)# interface TenGigE0/11/0/0
Router(config-if)# ipv4 access-group V4-ACL-INGRESS ingress
Router(config-if)# commit
Thu Jan 25 10:28:19.671 IST
Router(config-if)# exit

/* Verify if the ingress ACL has been successfully applied to the interface */
Router(config)# do show ipv4 interface
Thu Jan 25 10:29:44.944 IST
TenGigE0/11/0/0 is Up, ipv4 protocol is Up
  Vrf is default (vrfid 0x60000000)
  Internet address is 10.1.1.1/24
  MTU is 1514 (1500 is available to IP)
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is not set
  Inbound  common access list is not set, access list is V4-ACL-INGRESS
  Proxy ARP is disabled
  ICMP redirects are never sent
  ICMP unreachables are always sent
  ICMP mask replies are never sent
  Table Id is 0xe0000000
```

You have successfully configured an IPv4 ingress ACL on a Gigabit Ethernet interface.

Configuring an Egress IPv4 ACL on a Gigabit Ethernet Interface

Use the following configuration to configure an egress IPv4 ACL on a GigE interface.

```
/* Configure a GigE interface with an IPv4 address */
Router(config)# interface TenGigE 0/11/0/0
Router(config-if)# ipv4 address 20.1.1.1 255.255.255.0
Router(config-if)# no shut
Router(config-if)# commit
Thu Jan 25 10:08:38.767 IST
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv4 interface brief
Thu Jan 25 10:08:49.087 IST

Interface                      IP-Address      Status          Protocol Vrf-Name
TenGigE0/11/0/0        10.1.1.1        Up              Up       default
TenGigE0/11/0/1        20.1.1.1        Up              Up       default
```

```
/* Configure an IPv4 egress ACL */
Router(config)# ipv4 access-list V4-ACL-EGRESS
Router(config-ipv4-acl)# 10 permit ipv4 10.2.0.0 0.255.255.255 20.2.0.0 0.255.255.255
Router(config-ipv4-acl)# 20 deny ipv4 any any
Router(config-ipv4-acl)# commit
Thu Jan 25 10:25:04.655 IST

/* Verify the egress ACL creation */
Router(config)# do show access-lists ipv4
Thu Jan 25 10:25:19.896 IST
ipv4 access-list V4-ACL-EGRESS
 10 permit ipv4 10.0.0.0 0.255.255.255 20.0.0.0 0.255.255.255
 20 deny ipv4 any any
...

/* Apply the egress ACL to the GigE interface */
Router(config)# interface TenGigE 0/11/0/1
Router(config-if)# ipv4 access-group V4-ACL-EGRESS egress
Router(config-if)# commit
Thu Jan 25 10:28:45.937 IST
Router(config-if)# exit

/* Verify if the egress ACL has been successfully applied to the interface */
Router(config)# do show ipv4 interface
Thu Jan 25 10:29:44.944 IST
TenGigE 0/11/0/1 is Up, ipv4 protocol is Up
  Vrf is default (vrfid 0x60000000)
  Internet address is 20.1.1.1/24
  MTU is 1514 (1500 is available to IP)
  Helper address is not set
  Directed broadcast forwarding is disabled
  Outgoing access list is V4-ACL-EGRESS
  Inbound  common access list is not set, access list is not set
  Proxy ARP is disabled
  ICMP redirects are never sent
  ICMP unreachables are always sent
  ICMP mask replies are never sent
  Table Id is 0xe0000000
...
```

You have successfully configured an IPv4 egress ACL on a Gigabit Ethernet interface.

Configuring IPv6 ACLs

This section describes the steps to configure ingress and egress IPv6 ACLs over gigabit ethernet and bundle interfaces.

Notes and Restrictions for Configuring IPv6 Ingress ACLs

IPv6 ingress ACLs are characterized by the following behavior.

- Ingress IPv6 ACLs are supported on all interfaces:
- ACL-based Forwarding (ABF) is supported only in the ingress direction.
- The total number of ACLs allowed per NPU is 31.
- The number of attached ACEs allowed per line card is 2047.

- ACL logging with input interface (using the **log-input** keyword) is not supported.
- Packet Length (using the **pkt-length** keyword) is not supported.
- The **show access-lists ipv4 acl_name stats** command is not supported for ACE with logs, to check statistics. Therefore, use the **show access-lists** *acl-name* **hardware [ingress | egress] detail location***loc* command to check statistics for ACE with logs.
- Ingress ACL matching stats using **show access-lists ipv4**<*ACL name*>**hardware ingress location 0/RP0/CPU0** command may take more latency (up to 15 seconds) to reflect.

Notes and Restrictions for Configuring IPv6 Egress ACLs

IPv6 egress ACLs are characterized by the following behavior:

- Configuring packet length is not supported on egress ACLs.
- TCP flags are not supported on egress ACLs.
- Egress ACLs are not supported on BVI interfaces and L2 interfaces.
- Configuring qos-group is not supported on egress ACLs.
- A throughput of 50% or less is supported on egress ACLs.
- Apart from the throughput limitation, router-generated traffic is not be affected by egress IPv6 ACLs.
- The total number of egress ACLs allowed per NPU is 255.
- The total number of attached ACEs allowed per line card is 2000.
- Configuring dynamic TCAM key is not supported on egress ACLs.
- Upto 160GB of total IPv6 egress ACL is supported per NPU.

Configuring an Ingress IPv6 ACL on a Gigabit Ethernet Interface

Use the following configuration to configure an ingress IPv6 ACL on a GigE interface.

```
/* Configure a GigE interface with an IPv6 address */
Router(config)# interface TenGigE 0/11/0/0
Router(config-if)# ipv6 address 1001::1/64
Router(config-if)# no shut
Router(config-if)# commit
Thu Jan 25 10:07:54.700 IST
Router(config-if)# exit

/* Verify if the interface is up */
Router(config)# do show ipv6 interface brief
Thu Jan 25 12:38:35.742 IST
TenGigE 0/11/0/0 [Up/Up]
    fe80::bd:b9ff:fea9:5606
    1001::1
…

/* Configure an IPv6 ingress ACL */
Router(config)# ipv6 access-list V6-INGRESS-ACL
Router(config-ipv6-acl)# 10 permit ipv6 any any
Router(config-ipv6-acl)# 20 deny udp any any
Router(config-ipv6-acl)# commit
```

```
Thu Jan 25 11:31:24.488 IST
Router(config-ipv6-acl)# exit


/* Verify the ingress ACL creation */
Router(config)# do show access-lists ipv6
Thu Jan 25 11:34:56.911 IST
ipv6 access-list V6-INGRESS-ACL
 10 permit ipv6 any any
 20 deny udp any any


/* Apply the ingress ACL to the GigE interface */
Router(config)# interface TenGigE 0/11/0/0
Router(config-if)# ipv6 access-group V6-INGRESS-ACL ingress
Router(config-if)# commit
Thu Jan 25 11:32:55.194 IST
Router(config-if)# exit

/* Verify if the ingress ACL has been successfully applied to the interface */
Router(config)# do show ipv6 interface
Thu Jan 25 11:34:08.028 IST
TenGigE 0/11/0/0 is Up, ipv6 protocol is Up, Vrfid is default (0x60000000)
  IPv6 is enabled, link-local address is fe80::bd:b9ff:fea9:5606
  Global unicast address(es):
    1001::1, subnet is 1001::/64
  Joined group address(es): ff02::1:ff00:1 ff02::1:ffa9:5606 ff02::2
      ff02::1
  MTU is 1514 (1500 is available to IPv6)
  ICMP redirects are disabled
  ICMP unreachables are enabled
  ND DAD is enabled, number of DAD attempts 1
  ND reachable time is 0 milliseconds
  ND cache entry limit is 1000000000
  ND advertised retransmit interval is 0 milliseconds
  Hosts use stateless autoconfig for addresses.
  Outgoing access list is not set
  Inbound  common access list is not set, access list is V6-INGRESS-ACL
  Table Id is 0xe0800000
  Complete protocol adjacency: 0
  Complete glean adjacency: 0
  Incomplete protocol adjacency: 0
  Incomplete glean adjacency: 0
  Dropped protocol request: 0
  Dropped glean request: 0
…
```

You have successfully configured an IPv6 ingress ACL on a Gigabit Ethernet interface.

Configuring an Egress IPv6 ACL on a Gigabit Ethernet Interface

Use the following configuration to configure an egress IPv6 ACL on a GigE interface.

```
/* Configure a GigE interface with an IPv6 address */
Router(config)# interface TenGigE 0/11/0/1
Router(config-if)# ipv6 address 2001::1/64
Router(config-if)# no shut
Router(config-if)# commit
Thu Jan 25 11:41:25.778 IST
Router(config-if)# exit

/* Verify if the interface is up */
```

```
Router(config)# do show ipv6 interface brief
Thu Jan 25 12:38:35.742 IST
TenGigE 0/11/0/0 [Up/Up]
    fe80::bd:b9ff:fea9:5606
    1001::1
TenGigE 0/11/0/1 [Up/Up]
    fe80::23:e9ff:fea8:a44e
    2001::1

/* Configure an IPv6 egress ACL */
Router(config)# ipv6 access-list V6-EGRESS-ACL
Router(config-ipv6-acl)# 10 permit ipv6 any any
Router(config-ipv6-acl)# 20 deny udp any any
Router(config-ipv6-acl)# commit
Thu Jan 25 11:44:03.969 IST
Router(config-ipv6-acl)# exit

/* Verify the egress ACL creation */
Router(config)# do show access-lists ipv6
Thu Jan 25 11:45:53.823 IST
ipv6 access-list V6-EGRESS-ACL
 10 permit ipv6 any any
 20 deny udp any any
…

/* Apply the egress ACL to the GigE interface */
Router(config)# interface TenGigE 0/11/0/1
Router(config-if)# ipv6 access-group V6-EGRESS-ACL egress
Router(config-if)# commit
Thu Jan 25 11:45:12.682 IST
Router(config-if)# exit

/* Verify if the egress ACL has been successfully applied to the interface */
Router(config)# do show ipv6 interface
Thu Jan 25 11:46:43.234 IST
…
TenGigE 0/11/0/1 is Up, ipv6 protocol is Up, Vrfid is default (0x60000000)
  IPv6 is enabled, link-local address is fe80::23:e9ff:fea8:a44e
  Global unicast address(es):
    2001::1, subnet is 2001::/64
  Joined group address(es): ff02::1:ff00:1 ff02::1:ffa8:a44e ff02::2
      ff02::1
  MTU is 1514 (1500 is available to IPv6)
  ICMP redirects are disabled
  ICMP unreachables are enabled
  ND DAD is enabled, number of DAD attempts 1
  ND reachable time is 0 milliseconds
  ND cache entry limit is 1000000000
  ND advertised retransmit interval is 0 milliseconds
   Hosts use stateless autoconfig for addresses.
  Outgoing access list is V6-EGRESS-ACL
  Inbound  common access list is not set, access list is not set
  Table Id is 0xe0800000
  Complete protocol adjacency: 0
  Complete glean adjacency: 0
  Incomplete protocol adjacency: 0
  Incomplete glean adjacency: 0
  Dropped protocol request: 0
  Dropped glean request: 0
...
```

You have successfully configured an IPv6 egress ACL on a Gigabit Ethernet interface.

Configuring Ingress and Egress IPv6 ACLs on Bundle Interfaces

Use the following configuration to configure ingress and egress IPv6 ACLs on a bundle interface.

```
/* Configure a bundle interface with an IPv6 address */
Router(config)# interface Bundle-Ether 1
Router(config-if)# ipv6 address 3001::1/64
Router(config-if)# no shut
Router(config-if)# commit
Thu Jan 25 13:53:47.435 IST
Router(config-if)# exit

/* Configure an IPv6 egress ACL */
Router(config)# ipv6 access-list V6-EGRESS-ACL-bundle interface
Router(config-ipv6-acl)# 10 permit tcp any any range 3000 4000
Router(config-ipv6-acl)# 20 permit ipv6 any any
Router(config-ipv6-acl)# commit
Thu Jan 25 13:57:14.960 IST
Router(config-ipv6-acl)# exit

/* Configure an IPv6 ingress ACL to deny ingress traffic on the bundle interface */
Router(config)# ipv6 access-list V6-DENY-INGRESS-ACL
Router(config-ipv6-acl)# 10 deny ipv6 any any
Router(config-ipv6-acl)# commit
Thu Jan 25 13:59:23.198 IST
Router(config-ipv6-acl)# exit

/* Verify the egress and ingress ACL creation */
Router(config)# do show access-lists ipv6
Thu Jan 25 14:00:24.055 IST
ipv6 access-list V6-DENY-INGRESS-ACL
 10 deny ipv6 any any
ipv6 access-list V6-EGRESS-ACL-BI
 10 permit tcp any any range 3000 4000
 20 permit ipv6 any any
...

/* Apply the egress and ingress ACLs to the bundle interface */
Router(config)# interface Bundle-Ether 1
Router(config-if)# ipv6 access-group V6-EGRESS-ACL-BI egress
Router(config-if)# ipv6 access-group V6-DENY-INGRESS-ACL ingress
Router(config-if)# commit
Thu Jan 25 14:04:19.536 IST
Router(config-if)# exit

/* Verify if the ACLs have been successfully applied to the interface */
Router(config)# do show ipv6 interface
Thu Jan 25 11:46:43.234 IST
…
Thu Jan 25 14:04:51.322 IST
Bundle-Ether1 is Down, ipv6 protocol is Down, Vrfid is default (0x60000000)
  IPv6 is enabled, link-local address is fe80::1:10ff:fe87:8d04 [TENTATIVE]
  Global unicast address(es):
    3001::1, subnet is 3001::/64 [TENTATIVE]
  Joined group address(es): ff02::2 ff02::1
  MTU is 1514 (1500 is available to IPv6)
  ICMP redirects are disabled
  ICMP unreachables are enabled
  ND DAD is enabled, number of DAD attempts 1
  ND reachable time is 0 milliseconds
  ND cache entry limit is 1000000000
  ND advertised retransmit interval is 0 milliseconds
  ND router advertisements are sent every 160 to 240 seconds
```

```
ND router advertisements live for 1800 seconds
Hosts use stateless autoconfig for addresses.
Outgoing access list is V6-EGRESS-ACL-BI
Inbound  common access list is not set, access list is V6-DENY-INGRESS-ACL
Table Id is 0xe0800000
Complete protocol adjacency: 0
Complete glean adjacency: 0
Incomplete protocol adjacency: 0
Incomplete glean adjacency: 0
Dropped protocol request: 0
Dropped glean request: 0
```

You have successfully configured ingress and egress IPv6 ACLs on a bundle interface.

Modifying ACLs

This section describes a sample configuration for modification of ACLs.

```
*/ Create an Access List*/
Router(config)#ipv4 access-list acl_1

*/Add entries (ACEs) to the ACL*/
Router(config-ipv4-acl)#10 permit ip host 10.3.3.3 host 172.16.5.34
Router(config-ipv4-acl)#20 permit icmp any any
Router(config-ipv4-acl)#30 permit tcp any host 10.3.3.3
Router(config-ipv4-acl)#end

*/Verify the entries of the ACL*/:
Router#show access-lists ipv4 acl_1
ipv4 access-list acl_1
10 permit ip host 10.3.3.3 host 172.16.5.34
20 permit icmp any any
30 permit tcp any host 10.3.3.3

*/Add new entries, one with a sequence number "15" and another without a sequence number
to the ACL. Delete an entry with the sequence number "30":*/
Router(config)#ipv4 access-list acl_1
Router(config-ipv4-acl)# 15 permit 10.5.5.5 0.0.0.255
Router(config-ipv4-acl)# no 30
Router(config-ipv4-acl)# permit 10.4.4.4 0.0.0.255
Router(config-ipv4-acl)# commit


*/When an entry is added without a sequence number, it is automatically given a sequence
number
that puts it at the end of the access list. Because the default increment is 10, the entry
 will have a sequence
number 10 higher than the last entry in the existing access list*/

*/Verify the entries of the ACL:*/
Router(config)#show access-lists ipv4 acl_1
ipv4 access-list acl_1
 10 permit ipv4 host 10.3.3.3 host 172.16.5.34

15 permit 10.5.5.5 0.0.0.255---*/newly added ACE (with the sequence number)*/
20 permit icmp any any
30 permit ipv4 10.4.4.0 0.0.0.255 any ---*/newly added ACE (without the sequence number)*/

*/The entry with the sequence number 30, that is, "30 permit tcp any host 10.3.3.3" is
```

```
deleted from the ACL*/
```

You have successfully modified ACLs in operation.

Configuring ACL-based Forwarding

Converged networks carry voice, video and data. Users may need to route certain traffic through specific paths instead of using the paths computed by routing protocols. This is achieved by specifying the next-hop address in ACL configurations, so that the configured next-hop address from ACL is used for fowarding packet towards its destination instead of routing packet-based destination address lookup. This feature of using next-hop in ACL configurations for forwarding is called ACL Based Forwarding (ABF).

ACL-based forwarding enables you to choose service from multiple providers for broadcast TV over IP, IP telephony, data, and so on, which provides a cafeteria-like access to the Internet. Service providers can divert user traffic to various content providers.

Feature Highlights

- ABF is only supported on ingress ACL.
- ABF supports nexthop modifications. You can modify a nexthop, remove a nexthop, or make changes between existing nexthops.

Note While defining an ACE rule, you must specify the VRF for all nexthops unless the nexthop is in the default VRF. This will ensure that the packets take the right path towards the nexthop.

- VRF-aware ABF is supported for IPv4 and IPv6 with up to three next hops.
- IPv4 ABF nexthops routed over GRE interfaces are supported.
- As ABF is ACL-based, packets that do not match an existing rule (ACE) in the ACL are subject to the default ACL rule (drop all). If the ACL is being used for ABF-redirect only (not for security), then include an explicit ACE rule at the end of the ACL (lowest user priority) to match and "permit" all traffic. This ensures that all traffic that does not match an ABF rule is permitted and forwarded as normal.
- ABF is supported on permit rules only.
- VRF-select (where only the VRF is configured for the nexthop) is not supported.
- ABF default route is not supported.
- Packets punted in the ingress direction from the NPU to the linecard CPU are not subjected to ABF treatment due to lack of ABF support in the slow path. These packets will be forwarded normally based on destination-address lookup by the software dataplane. Some examples of these types of packets are (but are not limited to) packets with IPv4 options, IPv6 extension headers, and packets destined for glean (unresolved/incomplete) adjacencies.
- Packets destined to the local IP interface ("for-us" packets) are subjected to redirect if they match the rule containing the ABF action. This can be avoided by either designing the rule to be specific enough to avoid matching the “for-us” packets or placing an explicit permit ACE rule (with higher priority) into the ACL before the matching ABF rule.

Configuration Example

To configure ACL-based forwarding, perform this task:

```
/* Enter IPv4 access list configuration mode and configure an ACL: */
router# configure
router(config)# ipv4 access-list abf-acl

/* Set the conditions for the ACL and configure ABF: */
/* The next hop for this entry is specified. */
router(config-ipv4-acl)# 10 permit ipv4 192.168.18.0 0.255.255.255 any nexthop1 ipv4
192.168.20.2
router(config-ipv4-acl)# 15 permit ipv4 192.168.21.0 0.0.0.255 any
router(config-ipv4-acl)# 20 permit ipv4 192.168.22.0 0.0.255.255 any nexthop1 ipv4
192.168.23.2
/* More than two nexthops */
router(config-ipv4-acl)# 25 permit tcp any range 2000 3000 any range 4000 5000 nexthop1
ipv4 192.168.23.1 nexthop2 ipv4 192.168.24.1 nexthop3 ipv4 192.168.25.1

/* VRF support on ABF */
router(config-ipv4-acl)# 30 permit tcp any eq www host 192.168.12.2 precedence immediate
nexthop1 vrf vrf1_ipv4 ipv4 192.168.13.2 nexthop2 vrf vrf1_ipv4 ipv4 192.168.14.2

router(config-ipv4-acl)# 35 permit ipv4 any any

router(config-ipv4-acl)# commit

/* (Optional) Display ACL information: */
router# show access-lists ipv4 abf-acl
```

Running Configuration

```
ipv4 access-list abf-acl
10 permit ipv4 192.168.18.0 0.255.255.255 any nexthop1 192.168.20.2
15 permit ipv4 192.168.21.0 0.0.0.255 any
20 permit ipv4 192.168.22.0 0.0.255.255 any nexthop1 192.168.23.2
25 permit tcp any range 2000 3000 any range 4000 5000 nexthop1 ipv4 192.168.23.1 nexthop2
ipv4 192.168.24.1 nexthop3 ipv4 192.168.25.1
30 permit tcp any eq www host 192.168.12.2 precedence immediate nexthop1 vrf vrf1_ipv4 ipv4
 192.168.13.2 nexthop2 vrf vrf1_ipv4 ipv4 192.168.14.2
35 permit ipv4 any any
commit
!
```

Verification

Use the following command to verify the IP nexthop state in ABF to ensure that the expected nexthop is up:

```
Router# show access-lists ipv4 abf nexthops client pfilter_ea location 0/0/CPU0
Wed Jan 24 14:18:58.667 UTC

ACL name : abf-acl
ACE seq.   NH-1   NH-2   NH-3
 --------- --------------- --------------- ---------------
 10        192.168.13.2
 status          UP
 at status  Not Present
 exist      No
 vrf        default
 track
 pd     ctx  Present
```

```
25   192.168.14.2    192.168.11.1  192.168.12.1
status   UP   Down   Down
at status  Not Present  Not Present  Not Present
exist   No   Yes   Yes
vrf   default  default  default
track
pd ctx   Present  Not present  Not present
30   192.168.15.1  192.168.12.7
status   Unknown  Unknown
at status  Not Present  Not Present
exist   No   Yes
vrf   vrf1_ipv4  vrf1_ipv4
track
pd ctx   Not present  Not present
```

Use the following command to verify if ABF is currently attached to any interfaces at any linecard:

```
show access-lists usage pfilter location all
```

ACLs on Bridge Virtual Interfaces

Bridge Virtual Interfaces (BVIs) provide a bridge between the routing and bridging domains on a router. A BVI is configured with an IP address and operates as a regular routed interface. You can configure an ACL on a BVI to filter the traffic for the network that uses the interface.

Note Do not delete an ACL attached to a BVI interface when the BVI interface is not part of a bridge domain. Later, if you add the BVI interface to the bridge domain then the traffic is dropped.

Increased TCAM Consumption with Configuring ACLs on BVIs

The consumption of TCAM resources is impacted in the following manner when ACLs are configured on BVIs.

- For ingress ACLs, the TCAM entries for the same ACL are shared across interfaces on the same NPU.
- For egress ACLs, the TCAM entries for the same ACL are unique for all interfaces. This leads to greater consumption of TCAM resources.

Restrictions for Configuring ACLs on BVIs

You must be aware of the following restrictions before proceeding to configure ACLs on BVIs.

- Egress IPv6 ACLs are not supported on BVIs.
- When an egress IPv4 ACL is enabled on a BVI through the **hw-module** command, no other interface types are supported for the ACL (non-BVI interfaces are not supported for the ACL in this mode).

Prerequisites for Configuring IPv4 Egress ACLs on BVIs

By default, an IPv4 egress ACL on a BVI is disabled, and ACL filtering does not take place even when the ACL is attached to the BVI. Hence, we use the **hw-module** command, which enables the ACL when the line cards are reloaded.

Note IPv4 and IPv6 ingress ACLs do not require this configuration.

Use the following configuration to enable an IPv4 egress ACL on a BVI on the hardware and reload the line cards.

```
/* Enable an IPv4 egress ACL on BVI */
RP/0/RP0/CPU0:router(config)# hw-module profile acl egress layer3 interface-based
/* Enable permit statistics for the egress ACL (by default, only deny statistics are shown)*/
RP/0/RP0/CPU0:router(config)# hw-module profile stats acl-permit
RP/0/RP0/CPU0:router(config)# commit
RP/0/RP0/CPU0:router(config)# end
RP/0/RP0/CPU0:router# reload location all
Wed Apr 5 23:05:46.193 UTC
Proceed with reload? [confirm]
```

Configuration

The following section describes the procedure for configuring IPv4 ingress and egress ACLs on BVIs.

To configure IPv4 ingress and egress ACLs on a BVI, use the following procedure with sample configuration.

1. Enter the Global Configuration mode, and configure an IPv4 ingress ACL.

```
RP/0/RP0/CPU0:router(config)# ipv4 access-list v4-acl-ingress
RP/0/RP0/CPU0:router(config-ipv4-acl)# 10 permit tcp any 10.1.1.0/24 dscp cs6
RP/0/RP0/CPU0:router(config-ipv4-acl)# 20 deny udp any any eq ssh
RP/0/RP0/CPU0:router(config-ipv4-acl)# 30 permit ipv4 any any
RP/0/RP0/CPU0:router(config-ipv4-acl)# commit
RP/0/RP0/CPU0:router(config-ipv4-acl)# exit
```

2. Configure an IPv4 egress ACL.

```
RP/0/RP0/CPU0:router(config)# ipv4 access-list v4-acl-egress
RP/0/RP0/CPU0:router(config-ipv4-acl)# 10 deny ipv4 any any fragments log
RP/0/RP0/CPU0:router(config-ipv4-acl)# 20 deny tcp any any ack
RP/0/RP0/CPU0:router(config-ipv4-acl)# 30 permit ipv4 any any
RP/0/RP0/CPU0:router(config-ipv4-acl)# commit
RP/0/RP0/CPU0:router(config-ipv4-acl)# exit
```

3. Configure the Gigabit Ethernet interface that must be mapped to the BVI, and enable it for Layer 2 transport.

```
RP/0/RP0/CPU0:router(config)# interface GigabitEthernet 0/0/0/0
RP/0/RP0/CPU0:router(config-if)# l2transport
RP/0/RP0/CPU0:router(config-if-l2)# commit
```

4. Attach the ingress and egress ACLs to the BVI.

```
RP/0/RP0/CPU0:router(config)# interface BVI1
RP/0/RP0/CPU0:router(config-if)# ipv4 access-group v4-acl-ingress ingress
RP/0/RP0/CPU0:router(config-if)# ipv4 access-group v4-acl-egress egress
RP/0/RP0/CPU0:router(config-if)# commit
RP/0/RP0/CPU0:router(config-if)# exit
```

5. Configure the bridge domain with the Gigabit Ethernet interface and BVI.

```
RP/0/RP0/CPU0:router(config)# l2vpn
RP/0/RP0/CPU0:router(config-l2vpn)# bridge group BG1
RP/0/RP0/CPU0:router(config-l2vpn-bg)# bridge-domain B1
RP/0/RP0/CPU0:router(config-l2vpn-bg-bd)# interface GigabitEthernet 0/0/0/0
RP/0/RP0/CPU0:router(config-l2vpn-bg-bd-ac)# routed interface BVI1
RP/0/RP0/CPU0:router(config-l2vpn-bg-bd)# commit
RP/0/RP0/CPU0:router(config-l2vpn-bg-bd)# exit
RP/0/RP0/CPU0:router(config-l2vpn-bg)# exit
RP/0/RP0/CPU0:router(config-l2vpn)# exit
```

6. Confirm that your configuration has been successfully committed.

```
RP/0/RP0/CPU0:router(config)# show run
...
!
ipv4 access-list v4-acl-egress
 10 deny ipv4 any any fragments log
 20 deny tcp any any ack
 30 permit ipv4 any any
!
ipv4 access-list v4-acl-ingress
 10 permit tcp any 10.1.1.0/24 dscp cs6
 20 deny udp any any eq ssh
 30 permit ipv4 any any
!
interface GigabitEthernet0/0/0/0
 l2transport
 !
!
interface BVI1
ipv4 address 209.165.200.224/27
 ipv4 access-group v4-acl-ingress ingress
 ipv4 access-group v4-acl-egress egress

!
l2vpn
 bridge group BG1
  bridge-domain B1
   interface GigabitEthernet0/0/0/0
   !
   routed interface BVI1
  !
 !
!
end
```

7. Exit to the Executive Privileged mode and confirm that the ACLs are in operation.

```
RP/0/RP0/CPU0:router# show access-lists interface bvi1
Tue May 9 10:01:25.732 EDT
Input ACL (common): GigabitEthernet 0/0/0/0 (interface): v4-acl-ingress
Output ACL: v4-acl-egress

RP/0/RP0/CPU0:router# show access-lists summary
Tue May 9 10:02:01.167 EDT
ACL Summary:
Total ACLs configured: 2
Total ACEs configured: 6


RP/0/RP0/CPU0:router# show access-lists ipv4 v4-acl-egress hardware egress location
0/0/CPU0
```

```
ipv4 access-list v4-acl-egress
10 deny ipv4 any any fragments log (15214 matches)
20 deny tcp any any ack (15214 matches)
30 permit ipv4 any any (15214 matches)
```

The output clearly shows the configured ACLs, the total number of ACEs (three per ACL), and also the ACE matches in hardware.

You have successfully configured and enabled IPv4 ingress and egress ACL on a BVI.

Configuring ACLs with Fragment Control

The non-fragmented packets and the initial fragments of a packet were processed by IP extended access lists (if you apply this access list), but non-initial fragments were permitted, by default. However, now, the IP Extended Access Lists with Fragment Control feature allows more granularity of control over non-initial fragments of a packet. Using this feature, you can specify whether the system examines non-initial IP fragments of packets when applying an IP extended access list.

As non-initial fragments contain only Layer 3 information, these access-list entries containing only Layer 3 information, can now be applied to non-initial fragments also. The fragment has all the information the system requires to filter, so the access-list entry is applied to the fragments of a packet.

This feature adds the optional **fragments** keyword to the following IP access list commands: **deny** and **permit** . By specifying the **fragments** keyword in an access-list entry, that particular access-list entry applies only to non-initial fragments of packets; the fragment is either permitted or denied accordingly.

The behavior of access-list entries regarding the presence or absence of the **fragments** keyword can be summarized as follows:

If the Access-List Entry has...	Then...

...no **fragments** keyword and all of the access-list entry information matches	For an access-list entry containing only Layer 3 information:
	• The entry is applied to non-fragmented packets, initial fragments, and non-initial fragments.
	For an access-list entry containing Layer 3 and Layer 4 information:
	• The entry is applied to non-fragmented packets and initial fragments.
	  • If the entry matches and is a **permit** statement, the packet or fragment is permitted.
	  • If the entry matches and is a **deny** statement, the packet or fragment is denied.
	• The entry is also applied to non-initial fragments in the following manner. Because non-initial fragments contain only Layer 3 information, only the Layer 3 portion of an access-list entry can be applied. If the Layer 3 portion of the access-list entry matches, and
	  • If the entry is a **permit** statement, the non-initial fragment is permitted.
	  • If the entry is a **deny** statement, the next access-list entry is processed.
	Note The deny statements are handled differently for non-initial fragments versus non-fragmented or initial fragments.

...the **fragments** keyword and all of the access-list entry information matches	The access-list entry is applied only to non-initial fragments. **Note** The **fragments** keyword cannot be configured for an access-list entry that contains any Layer 4 information.

You should not add the **fragments** keyword to every access-list entry, because the first fragment of the IP packet is considered a non-fragment and is treated independently of the subsequent fragments. Because an initial fragment will not match an access list permit or deny entry that contains the **fragments** keyword, the packet is compared to the next access list entry until it is either permitted or denied by an access list entry that does not contain the **fragments** keyword. Therefore, you may need two access list entries for every deny entry. The first deny entry of the pair will not include the **fragments** keyword, and applies to the initial fragment. The second deny entry of the pair will include the **fragments** keyword and applies to the subsequent fragments. In the cases where there are multiple **deny** access list entries for the same host but with different Layer 4 ports, a single deny access-list entry with the **fragments** keyword for that host is all that has to be added. Thus all the fragments of a packet are handled in the same manner by the access list.

Packet fragments of IP datagrams are considered individual packets and each fragment counts individually as a packet in access-list accounting and access-list violation counts.

Note The **fragments** keyword cannot solve all cases involving access lists and IP fragments.

Note Within the scope of ACL processing, Layer 3 information refers to fields located within the IPv4 header; for example, source, destination, protocol. Layer 4 information refers to other data contained beyond the IPv4 header; for example, source and destination ports for TCP or UDP, flags for TCP, type and code for ICMP.

Configuring an IPv4 ACL to Match on Fragment Type

Most DoS (Denial of Service) attacks work by flooding the network with fragmented packets. By filtering the incoming fragments of the packet in a network, an extra layer of protection can be added against such attacks.

You can configure an IPv4 ACL to match on the fragment type, and perform an appropriate action. You can use the following sample configuration with the different fragment options:

```
/* Enter the global configuraton mode and configure an IPv4 access list */
Router# config
Router(config)# ipv4 access-list TEST
Router(config-ipv4-acl)# 10 permit tcp any any

/* Configure an ACE to match on the dont-fragment flag (indicates a non-fragmented packet)
 and forward the packet to the default (pre-configured) next hop  */
Router(config-ipv4-acl)# 20 permit tcp any any fragment-type dont-fragment default

/* Configure an ACE to match on the is-fragment flag (indicates a fragmented packet)
 and forward the packet to a next hop of 10.10.10.1  */
Router(config-ipv4-acl)# 30 permit udp any any fragment-type is-fragment nexthop1 ipv4
10.10.10.1

/* Configure an ACE to match on the first-fragment flag (indicates the first fragment of a
 fragmented packet)
 and forward the packet to a next hop of 20.20.20.1  */
Router(config-ipv4-acl)# 40 permit ospf any any fragment-type first-fragment nexthop1 ipv4
 20.20.20.1


/* Configure an ACE to match on the last-fragment flag (indicates the last fragment of a
fragmented packet)
 and forward the packet to a next hop of 30.30.30.1  */
Router(config-ipv4-acl)# 50 permit icmp any any fragment-type last-fragment nexthop1 ipv4
30.30.30.1
Router(config-ipv4-acl)# commit
```

Use Case: Configuring an IPv4 ACL to Match on the First Fragment and Last Fragment

This section describes an use case, where you configure an ACL to forward a fragment if it is the first fragment of the packet and discard a fragment if it is the last fragment of the packet.

In this configuration, the ACL checks the fragment offset value ('0' for the first fragment). If the fragment is the first fragment of the packet, the packet is forwarded. If the fragment is the last fragment of the packet, it is dropped at the interface.

```
/* Enter the global configuraton mode and configure an IPv4 access list */
Router# config
Thu Jan 11 11:56:27.221 IST
Router(config)# ipv4 access-list ACLFIRSTFRAG

/* Configure an ACE to match on the first fragment.
```

```
If the fragment offset value equals 0, the fragment is forwarded to the 192.168.1.2 next
hop */
Router(config-ipv4-acl)# 10 permit tcp any any fragment-type first-fragment nexthop1 ipv4
192.168.1.2

/* Configure an ACE to match on the last fragment, and drop the fragment at the interface.
 */
Router(config-ipv4-acl)# 20 deny tcp any any fragment-type last-fragment
Router(config-ipv4-acl)# commit
Thu Jan 11 12:01:33.297 IST

/* Validate the configuration */
Router(config-ipv4-acl)# do show access-lists
Thu Jan 11 12:05:23.646 IST
ipv4 access-list ACLFIRSTFRAG
 10 permit tcp any any fragment-type first-fragment nexthop1 ipv4 192.168.1.20
 20 deny tcp any any fragment-type last-fragment
```

You have successfully configured an IPv4 ACL to match on the fragment type.

Matching by Fragment Offset in ACLs

You can configure an access control list (ACL) rule to filter packets by the fragment-offset value. Depending on whether a packet matches the criteria in a permit or deny statement, the packet is either processed or dropped respectively at the interface. Fragment-offset filtering is supported only on ingress direction with compression mode of an ACL.

For more information about this feature, see the *Implementing Access Lists and Prefix Lists* chapter in the .For complete command reference, see the *Access List Commands* chapter in

Configuring ACL Matching by Fragment Offset

To configure fragment-offset match in ACL, use the **fragment-offset** option in **permit** or **deny** command in IPv4 or IPv6 access-list configuration mode.

Note For fragment-offset filtering, you must attach the particular ACL to an interface with compression level 3. Else, the configuration is rejected.

Configuration

This example shows how to specify an ACL rule based on the fragment-offset per IPv4 header. Here, the packet is permitted only if the fragment-offset in the IPv4 header of the packet is within the range of 300-400. The value *300-400* is based on the 8-byte unit, which is same as fragment-offset of *2400-3200* bytes.

```
/* Configure ACL */
Router# configure
Router(config)# ipv4 access-list fragment-offset-acl
Router(config-ipv4-acl)# 10 permit ipv4 any any fragment-offset range 300 400
Router# commit

/* Attach the ACL to the interface */
Router# configure
Router(config)# interface Bundle-Ether70
```

```
Router(config-if)# ipv4 access-group fragment-offset-acl ingress compress level 3
Router# commit
```

Running Configuration

```
ipv4 access-list fragment-offset-acl
 10 permit ipv4 any any fragment-offset range 300 400
!

interface Bundle-Ether70
 ipv4 address 192.0.2.1 255.255.255.0
 ipv6 address 2001:DB8::1:1::1/48
 ipv4 access-group fragment-offset-acl ingress compress level 3
!
```

Verify Fragment-offset Match in ACL

```
Router#
show access-lists ipv4 fragment-offset-acl usage pfilter loc 0/0/CPU0

Wed Apr 12 19:49:54.457 UTC
Interface : Bundle-Ether70
    Input  ACL : Common-ACL : N/A  ACL : fragment-offset-acl  (comp-lvl 3)
    Output ACL : N/A



Router#
show access-lists ipv4 fragment-offset-acl hardware ing int Bundle-Ether70 loc 0/0/CPU0

Wed Apr 12 19:51:07.837 UTC
ipv4 access-list fragment-offset-acl
 10 permit ipv4 any any fragment-offset range 300 400
```

Associated Commands

- ipv4 access-list
- ipv6 access-list
- deny (IPv4)
- deny (IPv6)
- fragment-offset
- permit (IPv4)
- permit (IPv6)

Configuring ACL Filtering by IP Packet Length

You can configure an access control list to filter packets by the packet length at an ingress interface. Depending on whether a packet matches the packet-length condition in a permit or deny statement, the packet is either processed or dropped respectively at the interface.

To configure packet length filtering in ACL, use the **packet-length** option in **permit** or **deny** command in IPv4 or IPv6 access-list configuration mode.

Restrictions

Packet length filtering feature in ACL is subjected to these restrictions:

- Packet length filtering is supported only on ingress direction, for both simple (non-compression) and hybrid (compression) ACLs.
- IPv6 packet length filtering is supported only for hybrid ACLs; not for simple ACLs.
- Only quantized (value divisible by 16) packet length filtering is supported for simple ACLs on IPv4.
- Packet length filtering is not supported in the default TCAM key, but instead requires a User-Defined TCAM Key (UDK) that can be specified using the `hw-module profile tcam format` command as described in the configuration section.

Configuring Simple IPv4 ACLs to Filter by Packet Length

To configure a simple ACL to filter by packet length in IPv4 networks, use the following steps.

1. Enable packet length filtering in the global configuration mode by using the `hw-module` command.

```
Router# config
Router(/config)# hw-module profile tcam format access-list ipv4 dst-addr dst-port proto
 packet-length frag-bit port-range
```

2. Enter the global configuration mode and configure a simple IPv4 access list to filter packets by the packet length value.

   In this particular example, we configure a set of statements to process only those packets that match the specified packet length condition. All other packets are dropped when this ACL is applied to an ingress interface.

```
Router# config
Router(config)# ipv4 access-list pktlen-v4
Router(config-ipv4-acl)# 10 permit tcp any any packet-length eq 1664
Router(config-ipv4-acl)# 20 permit udp any any packet-length range 1600 2000
Router(config-ipv4-acl)# 30 deny ipv4 any any
```

3. Commit the ACL and exit the IPv4 ACL configuration mode.

```
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# end
```

4. Apply the ACL to the required Ethernet interface.

```
Router(config)# interface Te0/0/0/0
Router(config-if)# ipv4 access-group pktlen-v4 ingress
```

5. Commit the configuration and exit the interface configuration mode.

```
Router(config-if)# commit
Router(config-if)# end
```

6. Verify your configuration.

```
Router# show access-lists pktlen-v4


ipv4 access-list pktlen-v4
10 permit ipv4 host 10.0.0.10 any packet-length lt 1008
20 permit ipv4 host 10.0.0.9 any packet-length gt 992
```

7. Verify the ACL matches in hardware.

```
Router# show access-lists pktlen-v4 hardware ingress location 0/0/CPU0

ipv4 access-list pklen-v4
10 permit ipv4 host 10.0.0.10 any packet-length lt 1008
20 permit ipv4 host 10.0.0.9 any packet-length gt 992
```

You have successfully configured a simple IPv4 ACL to filter by packet length.

Configuring Scaled IPv4 ACLs to Filter by Packet Length

To configure a scaled ACL to filter by packet length in IPv4 networks, use the following steps.

1. Enable packet length filtering in the global configuration mode by using the `hw-module` command.

```
Router# config
Router(/config)# hw-module profile tcam format access-list ipv4 dst-addr dst-port proto
 packet-length frag-bit port-range
```

2. Enter the global configuration mode and create an object group for configuring a scaled ACL.

```
Router(config)# object-group network ipv4 netobject1
Router(config-object-group-ipv4)# 50.0.0.0/24
Router(config-object-group-ipv4)# commit
```

3. From the global configuration mode, configure an IPv4 access list to filter packets by the packet length value.

   In this particular example, we configure a statement to process only those packets that match the specified packet length condition. All other packets are dropped when this ACL is applied to an ingress interface.

```
Router# configure
Router(config)# ipv4 access-list scaled_acl1
Router(config-ipv4-acl)# 10 permit ipv4 net-group netobject1 any packet-length eq 1000
```

4. Commit the ACL and exit the IPv4 ACL configuration mode.

```
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# end
```

5. Apply the ACL to the required Gigabit Ethernet interface.

```
Router(config)# interface Te0/0/0/3
Router(config-if)#ipv4 access-group scaled_acl1 ingress
```

6. Commit the configuration and exit the interface configuration mode.

```
Router(config-if)# commit
Router(config-if)# end
```

7. Verify your configuration.

```
Router# show access-lists scaled_acl1
ipv4 access-list scaled_acl1
10 permit ipv4 net-group netobject1 any packet-length eq 1000
```

8. Verify the ACL matches in hardware.

```
Router# show access-lists scaled_acl1 hardware ingress location 0/0/CPU0
ipv4 access-list scaled_acl1
10 permit ipv4 net-group netobject1 any packet-length eq 1000 (1500 hw matches)
```

You have successfully configured a scaled IPv4 ACL to filter by packet length.

Configuring Scaled IPv6 ACLs to Filter by Packet Length

To configure a scaled ACL to filter by packet length in IPv6 networks, use the following steps.

1. Enable packet length filtering in the global configuration mode by using the `hw-module` command.

```
Router# config
Router(/config)# hw-module profile tcam format access-list ipv4 dst-addr dst-port proto
 packet-length frag-bit port-range
```

2. Enter the global configuration mode and create an object group for configuring a scaled ACL.

```
Router(config)# object-group  network ipv6 netobject2
Router(config-object-group-ipv6)# 2001::0/128
Router(config-object-group-ipv6)# commit
```

3. From the global configuration mode, configure a scaled IPv6 access list to filter packets by the packet length value.

   In this particular example, we configure a statement to process only those packets that match the specified packet length condition. All other packets are dropped when this ACL is applied to an ingress interface.

```
Router(config)# ipv6 access-list scaled_acl2
Router(config-ipv6-acl)# 10 permit ipv6 net-group netobject2 any packet-length eq 1000
Router(config-ipv6-acl)# commit
```

4. Commit the ACL and exit the IPv6 ACL configuration mode.

```
Router(config-ipv6-acl)# commit
Router(config-ipv6-acl)# end
```

5. Apply the ACL to the required Gigabit Ethernet interface.

```
Router# config
Router(config)# interface Te/0/0/0/3
Router(config-if)# ipv6 access-group scaled_acl2 ingress
```

6. Commit the configuration and exit the interface configuration mode.

```
Router(config-if)# commit
Router(config-if)# end
```

7. Verify your configuration.

```
Router# show access-lists ipv6 scaled_acl2
ipv6 access-list scaled_acl2
10 permit ipv6 net-group netobject2 any packet-length eq 1000
```

8. Verify the ACL matches in hardware.

```
Router# show access-lists ipv6 scaled_acl2 hardware ingress location 0/0/CPU0
ipv6 access-list scaled_acl2
10 permit ipv6 net-group netobject2 any packet-length eq 1000 (2000 hw matches)
```

You have successfully configured a scaled IPv6 ACL to filter by packet length.

Understanding Object-Group ACLs

You can use object-group ACLs to classify users, devices, or protocols into groups so you can have a group-level access control policy. Instead of specifying individual IP addresses, protocols, and port numbers in multiple ACEs, you can specify just the object group in a single ACL.

This feature is very beneficial in large scale networks which currently contain hundreds of ACLs. By using the object-group ACL feature, the number of ACEs per ACL are significantly reduced. Object-group ACLs are also more readable, and easier to manage than conventional ACLs. Using object-group ACLs instead of conventional ACLs optimizes the storage needed in TCAM.

Types of Object-Group ACLs

You can create two types of object-group ACLs on Cisco IOS XR:

- **Network object-group ACLs**: Consist of groups of host IP Addresses and network IP addresses.
- **Port object-group ACLs**: Consist of groups of ports and supporting Layer 3/Layer 4 protocols.

Compressing ACLs

Object-group ACLs use compression to accommodate the large number of ACEs. Compression is achieved by compressing the following three fields of an ACE:

- Source IP prefix
- Destination IP prefix
- Source port number

There are only two compression levels in the access-group configuration for an ACL on an ingress interface:

- **Compress level 0**: No compression is done on the ACE fields.

In this mode, the object-group ACL behaves like a traditional ACL. Internal TCAM resources are utilized and there will be a huge impact on system resources and time taken for processing the ACL.

- **Compress level 3**: All three fields (source IP, destination IP, and source port) in an ACE are compressed.

  In this mode, external TCAM is used for prefix lookup, and internal TCAM is used for ACE lookup. This mode supports 16-bit based packet length filtering and fragment offset filtering.

Configuring an Object-Group ACL

Before You Begin

You must be aware of the following information that apply to object-group ACLs:

- You can configure ACLs that contain both conventional and object-group ACEs.
- You can modify the objects in an object group dynamically without redefining the object group or the ACE that references the object group.
- You can configure an object-group ACL multiple times with a source group, or a destination group, or both source and destination groups.

Restrictions

Configuring object-group ACLs involves the following restrictions:

- Object-group ACLs can only be configured to an interface. They cannot be used or referenced by applications like SSH, SNMP, NTP.
- To delete an object-group, you must first delete it from all ACLs.
- You cannot configure object-group ACLs along with QoS policies.
- Object-group ACLs are not supported in any policy based configuration.
- Nested object-groups are not supported from Release 6.2.1.

Configuring a Network Object-Group ACL

A network object group can contain a single or multiple network objects.

Configuration

Use the following set of configuration statements to configure a network object-group ACL for an IPv4 address.

```
/* From the global configuration mode, create a network object group. */
Router(config)# object-group network ipv4 netobj1
Router(config-object-group-ipv4)# description my-network-object
Router(config-object-group-ipv4)# host 10.1.1.1
Router(config-object-group-ipv4)# 10.2.1.0 255.255.255.0
Router(config-object-group-ipv4)# range 10.3.1.10 10.3.1.50

/* Create an access list referencing the object group. */
Router(config)# ipv4 access-list network-object-acl permit ipv4 net-group netobj1 any
```

```
/* Apply the access list containing the object group to the desired interface and commit
your configuration. */
Router(config)# interface TenGigE0/0/0/10/3
Router(config-if)# ipv4 address 1.1.1.1/24
Router(config-if)# no shut
Router(config-if)# ipv4 access-group network-object-acl ingress compress level 3
Router(config-if)# commit
Tue Mar 28 10:23:34.106 IST

RP/0/0/CPU0:Mar 28 10:37:48.570 : ifmgr[397]: %PKT_INFRA-LINK-3-UPDOWN : Interface
TenGigE0/0/0/10/3 , changed state to Down
RP/0/0/CPU0:Mar 28 10:37:48.608 : ifmgr[397]: %PKT_INFRA-LINK-3-UPDOWN : Interface
TenGigE0/0/0/10/3 , changed state to Up

Router(config-if)# exit
```

Use the following set of configuration statements to configure a network object-group ACL for an IPv6 address.

```
/* From the global configuration mode, create a network object group. */
Router(config)# object-group network ipv6 netobj1
Router(config-object-group-ipv6)# description my-network-object
Router(config-object-group-ipv6)# host 2001:DB8:1::1
Router(config-object-group-ipv6)# 2001:DB8::1 2001:DB8:0:ABCD::1
Router(config-object-group-ipv6)# range 2001:DB8::2 2001:DB8::5

/* Create an access list referencing the object group. */
Router(config)# ipv6 access-list network-object-acl permit ipv6 net-group netobj1 any

/* Apply the access list containing the object group to the desired interface and commit
your configuration. */
Router(config)# interface TenGigE0/0/0/10/3
Router(config-if)# ipv6 address 2001:DB8::1/32
Router(config-if)# no shut
Router(config-if)# ipv6 access-group network-object-acl ingress compress level 3
Router(config-if)# commit
Tue Mar 28 10:23:34.106 IST

RP/0/0/CPU0:Mar 28 10:37:48.570 : ifmgr[397]: %PKT_INFRA-LINK-3-UPDOWN : Interface
TenGigE0/0/0/10/3 , changed state to Down
RP/0/0/CPU0:Mar 28 10:37:48.608 : ifmgr[397]: %PKT_INFRA-LINK-3-UPDOWN : Interface
TenGigE0/0/0/10/3 , changed state to Up

Router(config-if)# exit
```

Running Configuration

Confirm your configuration.

```
Router(config)# show run
Tue Mar 28 10:37:55.737 IST

Building configuration...
!! IOS XR Configuration 0.0.0
...

!
object-group network ipv4 netobj1
 10.2.1.0/24
```

```
 host 10.1.1.1
 range  10.3.1.10 10.3.1.50
 description my-network-object
!
!
ipv4 access-list network-object-acl
 10 permit ipv4 net-group netobj1 any
!
interface Te0/0/0/0/3
 ipv4 address 1.1.1.1 255.255.255.0
 ipv4 access-group network-object-acl ingress compress level 3
!
```

You have successfully configured a network object-group ACL.

Configuring a Port Object-Group ACL

A port object group can contain a single or multiple port objects.

Configuration

Use the following set of configuration statements to configure a port object-group ACL.

```
/* From the global configuration mode, create a port object group, and commit your
configuration. */
RP/0/RP0/CPU0:router(config)# object-group port portobj1
RP/0/RP0/CPU0:router(config-object-group-ipv4)# description my-port-object
RP/0/RP0/CPU0:router(config-object-group-ipv4)# eq bgp
RP/0/RP0/CPU0:router(config-object-group-ipv4)# range 100 200
RP/0/RP0/CPU0:router(config-object-group-ipv4)# commit
RP/0/RP0/CPU0:router(config-object-group-ipv4)# exit

/* Create an access list referencing the object group. */
RP/0/RP0/CPU0:router(config)# ipv4 access-list port-object-acl permit ipv4 net-group portobj1


/* Apply the access list containing the object group to the desired interface and commit
your configuration. */
RP/0/RP0/CPU0:router(config)# interface Te0/0/0/3
RP/0/RP0/CPU0:router(config-if)# ipv4 address 2.2.2.2/24
RP/0/RP0/CPU0:router(config-if)# ipv4 access-group port-obj-acl ingress compress level 3
RP/0/RP0/CPU0:router(config-if)# no shut
RP/0/RP0/CPU0:router(config-if)# commit
Tue Mar 28 10:23:34.106 IST

RP/0/0/CPU0:Mar 28 10:37:48.570 : ifmgr[397]: %PKT_INFRA-LINK-3-UPDOWN : Interface
TenGigE0/0/0/10/3 , changed state to Down
RP/0/0/CPU0:Mar 28 10:37:48.608 : ifmgr[397]: %PKT_INFRA-LINK-3-UPDOWN : Interface
TenGigE0/0/0/10/3 , changed state to Up

RP/0/RP0/CPU0:router(config-if)# exit
```

Running Configuration

Confirm your configuration.

```
RP/0/RP0/CPU0:router(config)# show run
Tue Mar 28 10:37:55.737 IST
```

```
Building configuration...
!! IOS XR Configuration 0.0.0
...
object-group port portobj1
 eq bgp
 range 100 200
!

ipv4 access-list port-object-acl
 10 permit tcp net-group portobj1
!
interface Te/0/0/0/3
 ipv4 access-group port-obj-acl ingress compress level 3
!
end
!
```

You have successfully configured a port object-group ACL.

Verifying Object-Group ACL Compression

You can use the commands described in this section to verify the configured object-group ACLs in operation and the compression of the ACEs in the ACL.

Note The outputs provided in this section are a standalone sample and are not related to the configurations provided in the preceding sections.

Verification

Use the following set of verification commands to verify object-group ACL compression.

```
/* Verify the entries of the ACL in operation. */

Router# show access-lists ipv4 network-object-acl hardware ingress location 0/0/CPU0
ipv4 access-list network-object-acl
40 permit ospf net-group n_192.168.0.0_16 any (20898463272 matches)
70 permit tcp any net-group CORP_ALL_V4 established
100 permit udp net-group INTERNAL port-group KERBEROS_UDP net-group CORP_ALL_V4
130 permit udp net-group INTERNAL port-group DNS_UDP net-group CORP_ALL_V4
160 permit udp net-group INTERNAL port-group NTP net-group CORP_ALL_V4
190 permit udp net-group INTERNAL port-group LDAP_UDP net-group CORP_ALL_V4
...
1500 permit udp net-group VLAN60_SECURITY net-group h_192.168.77.242 port-group
UDP_50000-50100
1530 deny ipv4 net-group VLAN60_SECURITY any log (20891956640 matches)
...


/* Verify the ACE compression in the ACL. */
Router# show access-lists ipv4 network-object-acl hardware ingress verify location 0/0/CPU0
Verifying TCAM entries for network-object-acl
Please wait...



    INTF    NPU lookup  ACL # intf Total  compression Total   result failed(Entry) TCAM
entries
```

```
               type    ID  shared ACES   prefix-type Entries        ACE SEQ #     verified
---------- --- ------- --- ------ ------ ----------- ------- ------ ------------ ------------

TenGigE0_0_0_10_3 (ifhandle: 0x1c8)

             1 IPV4      2      1    247 COMPRESSED       810 passed
  810
                                         SRC IP          2746 passed
 2746
                                         DEST IP         3413 passed
 3413
                                         SRC PORT         340 passed
  340
```

You have successfully verified the compression of ACEs within an ACL.

Note The command `show access-lists` *access-list-name* `hardware ingress detail location` *location* displays compressed output for source and destination IP addresses when the `detail` keyword is used while attaching ACLs to interfaces.

Configuring TTL Matching and Rewriting for IPv4 ACLs

You can configure ACLs to match on the TTL value specified in the IPv4 header. You can specify the TTL match condition to be based on a single value, or multiple values. You can also rewrite the TTL value in the IPv4 header by using the **set ttl** command.

Limitations for using TTL matching and rewriting for IPv4 ACLs

Using TTL matching and rewriting for IPv4 ACLs is known to have the following limitations.

- TTL matching is supported only for ingress ACLs.
- ACL logging is not supported for ingress ACLs after a User-Defined TCAM Key (UDK) is configured with the **enable-set-ttl** option.
- If a TTL rewrite is applied to the outer IPv4 header of an IP-in-IP header, then when the outer IPv4 header is decapsulated, (by GRE decapsulation) the TTL rewrite is also applied to the inner IPv4 header.
- TTL matching is not supported in the default TCAM key, but instead requires a User-Defined TCAM Key (UDK) using the **hw-module profile tcam format** command as described in the configuration section.

Configuration

Use the following steps to configure TTL matching and rewriting for IPv4 ACLs.

```
/* Enable TTL matching and rewriting in the global configuration mode by using the hw-module
 command */
Router(config)# hw-module profile tcam format access-list ipv4 dst-addr dst-port proto
port-range enable-set-ttl ttl-match

/* Configure an IPv4 ACL with the TTL parameters */
Router(config)# ipv4 access-list acl-v4
```

```
Router(config-ipv4-acl)# 10 deny tcp any any ttl eq 100
Router(config-ipv4-acl)# 20 permit tcp any any ttl range 1 50 set ttl 200
Router(config-ipv4-acl)# 30 permit tcp any any ttl neq 100 set ttl 255
Router(config-ipv4-acl)# commit
Thu Nov  2 12:22:58.948 IST

/* Attach the IPv4 ACL to the GigE interface */
Router(config)# interface Te0/0/0/0
Router(config-if)# ipv4 address 15.1.1.1 255.255.255.0
Router(config-if)# ipv4 access-group acl-v4 ingress
Router(config-if)# commit
```

Running Configuration

Validate your configuration by using the **show run** command.

```
Router(config)# show run
Thu Nov  2 14:01:53.376 IST
Building configuration...
!! IOS XR Configuration 0.0.0
!! Last configuration change at Thu Nov  2 12:22:59 2017 by annseque
!
hw-module profile tcam format access-list ipv4 dst-addr dst-port proto port-range
enable-set-ttl ttl-match
!
ipv4 access-list acl-v4
 10 deny tcp any any ttl eq 100
 20 permit tcp any any ttl range 1 50 set ttl 200
 30 permit tcp any any ttl neq 100 set ttl 255
!
interface Te0/0/0/0
 ipv4 address 15.1.1.1 255.255.255.0
 ipv4 access-group acl-v4 ingress
!
```

You have successfully configured TTL matching and rewriting for IPv4 ACLs.

Configuring Interface-Based Unique IPv4 ACLs

ACLs that are shared across interfaces and use the same TCAM space are known as shared ACLs. However, you can configure only 31 unique, shared ACLs. To configure more unique ACLs, ACL sharing must be disabled by using the **interface-based** command. By making the ACLs unique for an interface, you can configure more than 31 ACLs.

Configuration

Use the following configuration to create unique, interface-based IPv4 ACLs.

Note

- A reboot of the line cards is required after entering the **hw-module profile** command to activate the command.
- TCAM is allocated for each ACL on an interface, and is not shared across ACLs. Hence, for instance, if an ACL utitlizes 10 TCAM entries, and is applied to 100 interfaces, the total number of TCAM entries will be 1000.

```
/* Enable interface-based, unique IPv4 ACLs */
Router(config)# hw-module profile tcam format access-list ipv4 src-addr src-port dst-addr
dst-port interface-based

/* Configure an IPv4 ACL with the TTL parameters */
Router(config)# ipv4 access-list acl-v4
Router(config-ipv4-acl)# 10 deny tcp any any ttl eq 100
Router(config-ipv4-acl)# 20 permit tcp any any ttl range 1 50 set ttl 200
Router(config-ipv4-acl)# 30 permit tcp any any ttl neq 100 set ttl 255
Router(config-ipv4-acl)# commit
Thu Nov  2 12:22:58.948 IST

/* Attach the IPv4 ACL to the GigE interface */
Router(config)# interface Te0/0/0/0
Router(config-if)# ipv4 address 15.1.1.1 255.255.255.0
Router(config-if)# ipv4 access-group acl-v4 ingress
Router(config-if)# commit
```

Running Configuration

Validate your configuration by using the **show run** command.

```
Router(config)# show run
Thu Nov  2 14:01:53.376 IST
Building configuration...
!! IOS XR Configuration 0.0.0
!! Last configuration change at Thu Nov  2 12:22:59 2017 by annseque
!
hw-module profile tcam format access-list ipv4 src-addr src-port dst-addr dst-port
interface-based
!
ipv4 access-list acl-v4
 10 deny tcp any any ttl eq 100
 20 permit tcp any any ttl range 1 50 set ttl 200
 30 permit tcp any any ttl neq 100 set ttl 255
!
interface Te0/0/0/0
 ipv4 address 15.1.1.1 255.255.255.0
 ipv4 access-group acl-v4 ingress
!
```

You have successfully configured unique, interface-based IPv4 ACLs.

Configuring TTL Matching and Rewriting for IPv6 ACLs

You can configure ACLs to match on the TTL value specified in the IPv6 header. You can specify the TTL match condition to be based on a single value, or multiple values. You can also rewrite the TTL value in the IPv6 header by using the **set ttl** command.

Note A reboot of the line cards is required after entering the **hw-module profile** command to activate the command.

Limitations for using TTL matching and rewriting for IPv6 ACLs

Using TTL matching and rewriting for IPv6 ACLs is known to have the following limitations.

- TTL matching is supported only for ingress ACLs.
- ACL logging is not supported for ingress ACLs after a User-Defined TCAM Key (UDK) is configured with the **enable-set-ttl** option.
- If a TTL rewrite is applied to the outer IPv6 header of an IP-in-IP header, then when the outer IPv6 header is decapsulated, (by GRE decapsulation) the TTL rewrite is also applied to the inner IPv6 header.
- TTL matching is not supported in the default TCAM key, but instead requires a User-Defined TCAM Key (UDK) using the **hw-module profile tcam format** command as described in the Configuration section.

Configuration

Use the following steps to configure TTL matching and rewriting for IPv6 ACLs.

```
/* Enable TTL matching and rewriting in the global configuration mode by using the hw-module
 command */
Router(config)# hw-module profile tcam format access-list ipv6 dst-addr dst-port src-port
next-hdr enable-set-ttl ttl-match

/* Configure an IPv6 ACL with the TTL parameters */
Router(config)# ipv6 access-list acl-v6
Router(config-ipv6-acl)# 10 deny tcp any any ttl eq 50
Router(config-ipv6-acl)# 20 permit tcp any any ttl lt 50 set ttl 255
Router(config-ipv6-acl)# 30 permit tcp any any ttl gt 50 set ttl 200
Router(config-ipv6-acl)# commit
Thu Nov  2 12:22:58.948 IST

/* Attach the IPv6 ACL to the GigE interface */
Router(config)# interface Te0/0/0/0
Router(config-if)# ipv6 address 2001:2:1::1/64
Router(config-if)# ipv6 access-group acl-v6 ingress
Router(config-if)# commit
```

Running Configuration

Validate your configuration by using the **show run** command.

```
Router(config)# show run
Thu Nov  2 14:01:53.376 IST
Building configuration...
!! IOS XR Configuration 0.0.0
!! Last configuration change at Thu Nov  2 12:22:59 2017 by annseque
!hw-module profile tcam format access-list ipv6 dst-addr dst-port src-port next-hdr
enable-set-ttl ttl-match
!
ipv6 access-list acl-v6
 10 deny tcp any any ttl eq 50
20 permit tcp any any ttl lt 50 set ttl 255
30 permit tcp any any ttl gt 50 set ttl 200
!
interface Te0/0/0/0
 ipv6 address 2001:2:1::1/64
 ipv6 access-group acl-v6 ingress
!
```

You have successfully configured TTL matching and rewriting for IPv6 ACLs.

Configuring Interface-Based Unique IPv6 ACLs

ACLs that are shared across interfaces and use the same TCAM space are known as shared ACLs. However, you can configure only 31 unique, shared ACLs. To configure more unique ACLs, ACL sharing must be disabled by using the **interface-based** command. By making the ACLs unique for an interface, you can configure more than 31 ACLs.

Configuration

Use the following configuration to create unique, interface-based IPv6 ACLs.

Note

- A reboot of the line cards is required after entering the hw-module profile command to activate the command.
- TCAM is allocated for each ACL on an interface, and is not shared across ACLs. Hence, for instance, if an ACL utitlizes 10 TCAM entries, and is applied to 100 interfaces, the total number of TCAM entries will be 1000.

```
/* Enable interface-based, unique IPv6 ACLs */
Router(config)# hw-module profile tcam format access-list ipv6 src-addr src-port dst-addr
dst-port next-hdr interface-based

/* Configure an IPv6 ACL with the TTL parameters */
Router(config)# ipv6 access-list acl-v6
Router(config-ipv6-acl)# 10 deny tcp any any ttl eq 100
Router(config-ipv6-acl)# 20 permit tcp any any ttl range 1 50 set ttl 200
Router(config-ipv6-acl)# 30 permit tcp any any ttl neq 100 set ttl 255
Router(config-ipv6-acl)# commit
Thu Nov  2 12:22:58.948 IST

/* Attach the IPv6 ACL to the GigE interface */
Router(config)# interface Te0/0/0/0
Router(config-if)# ipv6 address 2001:2:1::1/64
Router(config-if)# ipv6 access-group acl-v6 ingress
Router(config-if)# commit
```

Running Configuration

Validate your configuration by using the **show run** command.

```
Router(config)# show run
Thu Nov  2 14:01:53.376 IST
Building configuration...
!! IOS XR Configuration 0.0.0
!! Last configuration change at Thu Nov  2 12:22:59 2017 by annseque
!
hw-module profile tcam format access-list ipv6 src-addr src-port dst-addr dst-port next-hdr
 interface-based
!
ipv6 access-list acl-v6
 10 deny tcp any any ttl eq 100
 20 permit tcp any any ttl range 1 50 set ttl 200
 30 permit tcp any any ttl neq 100 set ttl 255
!
```

```
interface Te0/0/0/0
 ipv6 address 2001:2:1::1/64
 ipv6 access-group acl-v6 ingress
!
```

You have successfully configured unique, interface-based IPv6 ACLs.

Understanding IP Access List Logging Messages

Cisco IOS XR software can provide logging messages about packets permitted or denied by a standard IP access list. That is, any packet that matches the access list causes an informational logging message about the packet to be sent to the console. The level of messages logged to the console is controlled by the **logging console** command in global configuration mode.

The first packet that triggers the access list causes an immediate logging message, and subsequent packets are collected over 5-minute intervals before they are displayed or logged. The logging message includes the access list number, whether the packet was permitted or denied, the source IP address of the packet, and the number of packets from that source permitted or denied in the prior 5-minute interval.

However, you can use the { **ipv4 | ipv6** } **access-list log-update threshold** command to set the number of packets that, when they match an access list (and are permitted or denied), cause the system to generate a log message. You might do this to receive log messages more frequently than at 5-minute intervals.

Caution If you set the *update-number* argument to 1, a log message is sent right away, rather than caching it; every packet that matches an access list causes a log message. A setting of 1 is not recommended because the volume of log messages could overwhelm the system.

Even if you use the { **ipv4 | ipv6} access-list log-update threshold** command, the 5-minute timer remains in effect, so each cache is emptied at the end of 5 minutes, regardless of the number of messages in each cache. Regardless of when the log message is sent, the cache is flushed and the count reset to 0 for that message the same way it is when a threshold is not specified.

Note The logging facility might drop some logging message packets if there are too many to be handled or if more than one logging message is handled in 1 second. This behavior prevents the router from using excessive CPU cycles because of too many logging packets. Therefore, the logging facility should not be used as a billing tool or as an accurate source of the number of matches to an access list.

Understanding Prefix Lists

Prefix lists are used in route maps and route filtering operations and can be used as an alternative to access lists in many Border Gateway Protocol (BGP) route filtering commands. A prefix is a portion of an IP address, starting from the far left bit of the far left octet. By specifying exactly how many bits of an address belong to a prefix, you can then use prefixes to aggregate addresses and perform some function on them, such as redistribution (filter routing updates).

BGP Filtering Using Prefix Lists

Prefix lists can be used as an alternative to access lists in many BGP route filtering commands. It is configured under the Global configurations of the BGP protocol. The advantages of using prefix lists are as follows:

- Significant performance improvement in loading and route lookup of large lists.
- Incremental updates are supported.
- More user friendly CLI. The CLI for using access lists to filter BGP updates is difficult to understand and use because it uses the packet filtering format.
- Greater flexibility.

Before using a prefix list in a command, you must set up a prefix list, and you may want to assign sequence numbers to the entries in the prefix list.

How the System Filters Traffic by Prefix List

Filtering by prefix list involves matching the prefixes of routes with those listed in the prefix list. When there is a match, the route is used. More specifically, whether a prefix is permitted or denied is based upon the following rules:

- An empty prefix list permits all prefixes.
- An implicit deny is assumed if a given prefix does not match any entries of a prefix list.
- When multiple entries of a prefix list match a given prefix, the longest, most specific match is chosen.

Sequence numbers are generated automatically unless you disable this automatic generation. If you disable the automatic generation of sequence numbers, you must specify the sequence number for each entry using the *sequence-number* argument of the **permit** and **deny** commands in IPv4 prefix list configuration command. Use the **no** form of the **permit** or **deny** command with the *sequence-number* argument to remove a prefix-list entry.

The **show** commands include the sequence numbers in their output.

Configuring Prefix Lists

Configuration Example

Creates a prefix-list "pfx_2" with a remark "Deny all routes with a prefix of 10/8". This prefix-list denies all prefixes matching /24 in 128.0.0.0/8.

```
Router#configure
Router(config)#ipv4 prefix-list pfx_2


Router(config-ipv4_pfx)#10 remark Deny all routes with a prefix of 10/8
Router(config-ipv4_pfx)#20 deny 128.0.0.0/8 eq 24
/* Repeat the above step as necessary. Use the no sequence-number command to delete an
entry. */

Router(config-ipv4_pfx)#commit
```

Running Configuration

```
Router#show running-config ipv4 prefix-list pfx_2
ipv4 prefix-list pfx_2
 10 remark Deny all routes with a prefix of 10/8
 20 deny 128.0.0.0/8 eq 24
!
```

Verification

Verify that the permit and remark settings are according to the set configuration.

```
Router# show prefix-list pfx_2
ipv4 prefix-list pfx_2
 10 remark Deny all routes with a prefix of 10/8
 20 deny 128.0.0.0/8 eq 24
RP/0/RP0/CPU0:ios#
```

Associated Commands

Sequencing Prefix List Entries and Revising the Prefix List

Configuration Example

Assigns sequence numbers to entries in a named prefix list and how to add or delete an entry to or from a prefix list. It is assumed a user wants to revise a prefix list. Resequencing a prefix list is optional.

```
Router#config
Router(config)#ipv4 prefix-list cl_1


Router(config)#10 permit 172.16.0.0 0.0.255.255
/* Repeat the above step as necessary adding statements by sequence number where you planned;
 use the no sequence-number command to delete an entry */

Router(config)#commit
end
Router#resequence prefix-list ipv4 cl_1 20 15
```

Running Configuration

```
/*Before resequencing/*
Router#show running-config ipv4 prefix-list cl_1
ipv4 prefix-list cl_1
 10 permit 172.16.0.0/16
!
/* After resequencing using the resequence prefix-list ipv4 cl_1 20 15 command: */
Router#show running-config ipv4 prefix-list cl_1
ipv4 prefix-list cl_1
 35 permit 172.16.0.0/16
!
```

Verification

Verify that the prefix list has been resequenced:

```
Router#show prefix-list cl_1
ipv4 prefix-list cl_1
 35 permit 172.16.0.0/16
```

Associated Commands