Cisco IOS ACL Support

Note

- For complete information about configuring Cisco IOS ACLs, see this publication:

  http://www.cisco.com/en/US/docs/ios-xml/ios/sec_data_acl/configuration/15-sy/sec-data-acl-15-sy-book.html

- For complete syntax and usage information for the commands used in this chapter, see these publications:

  http://www.cisco.com/en/US/products/ps11846/prod_command_reference_list.html

- Cisco IOS Release 15.1SY supports only Ethernet interfaces. Cisco IOS Release 15.1SY does not support any WAN features or commands.

Tip

For additional information about Cisco Catalyst 6500 Series Switches (including configuration examples and troubleshooting information), see the documents listed on this page:

http://www.cisco.com/en/US/products/hw/switches/ps708/tsd_products_support_series_home.html

Participate in the Technical Documentation Ideas forum

Restrictions for Cisco IOS ACLs

- You can apply Cisco IOS ACLs directly to Layer 3 ports and to VLAN interfaces.
- You can apply VLAN ACLs and port ACLs to Layer 2 interfaces and VLANs (see Chapter 1, "Port ACLs (PACLs)" and Chapter 1, "VLAN ACLs (VACLs)").
- Each type of ACL (IP, IPX, and MAC) filters only traffic of the corresponding type. A Cisco IOS MAC ACL never matches IP or IPX traffic unless the **mac packet-classify** configuration command is enabled. By default, the **mac packet-classify** configuration command is disabled.
- When you enter the **mac packet-classify** configuration command, MAC ACLs will be applied to all protocol traffic.
- The PFC does not provide hardware support for Cisco IOS IPX ACLs. Cisco IOS IPX ACLs are supported in software on the route processor (RP).
- By default, the RP sends Internet Control Message Protocol (ICMP) unreachable messages when a packet is denied by an access group.

  With the **ip unreachables** command enabled (which is the default), the switch drops most of the denied packets in hardware and sends only a small number of packets to the RP to be dropped in software, which generates ICMP-unreachable messages.

  The **ip unreachables** command does not impact the hardware behavior for ACL drop packets and the leaking of ACL deny packets is enabled by default. You can enter the **no ip unreachables** interface configuration command to disable ICMP unreachable messages.
- ICMP unreachable messages are not sent if a packet is denied by a VACL or a PACL.
- Use named ACLs (instead of numbered ACLs) because this causes less CPU usage when creating or modifying ACL configurations and during system restarts. When you create ACL entries (or modify existing ACL entries), the software performs a CPU-intensive operation called an ACL merge to load the ACL configurations into the PFC hardware. An ACL merge also occurs when the startup configuration is applied during a system restart.

  With named ACLs, the ACL merge is triggered only when the user exits the **named-acl** configuration mode. However, with numbered ACLs, the ACL merge is triggered for every ACE definition and results in a number of intermediate merges during ACL configuration.

Restrictions for Layer 4 Operators in ACLs

- Determining Layer 4 Operation Usage, page 1-2
- Determining Logical Operation Unit Usage, page 1-3

Determining Layer 4 Operation Usage

You can specify these types of operations:

- gt (greater than)
- lt (less than)
- neq (not equal)
- eq (equal)
- range (inclusive range)

We recommend that you do not specify more than *nine different* operations on the same ACL. If you exceed this number, each new operation might cause the affected ACE to be translated into more than one ACE.

Use the following two guidelines to determine Layer 4 operation usage:

- Layer 4 operations are considered different if the operator or the operand differ. For example, in this ACL there are three different Layer 4 operations ("gt 10" and "gt 11" are considered two different Layer 4 operations):

```
... gt 10 permit
... lt 9 deny
... gt 11 deny
```

Note There is no limit to the use of "eq" operators as the "eq" operator does not use a logical operator unit (LOU) or a Layer 4 operation bit. See the "Determining Logical Operation Unit Usage" section on page 1-3 for a description of LOUs.

- Layer 4 operations are considered different if the same operator/operand couple applies once to a source port and once to a destination port. For example, in this ACL there are two different Layer 4 operations because one ACE applies to the source port and one applies to the destination port.

```
... Src gt 10 ...
... Dst gt 10
```

Determining Logical Operation Unit Usage

Logical operation units (LOUs) are registers that store operator-operand couples. All ACLs use LOUs. There can be up to 64 LOUs and each LOU can store two different operator-operand couples, making the total number of LOU registers to be 128. LOU usage per Layer 4 operation is as follows:

- gt uses 1/2 LOU
- lt uses 1/2 LOU
- neq uses 1/2 LOU
- range uses 1 LOU
- eq does not require a LOU

For example, this ACL would use a single LOU to store two different operator-operand couples:

```
... Src gt 10 ...
... Dst gt 10
```

A more detailed example follows:

```
ACL1
... (dst port) gt 10 permit
... (dst port) lt 9 deny
... (dst port) gt 11 deny
... (dst port) neq 6 permit
... (src port) neq 6 deny
... (dst port) gt 10 deny

ACL2
... (dst port) gt 20 deny
... (src port) lt 9 deny
... (src port) range 11 13 deny
... (dst port) neq 6 permit
```

The Layer 4 operations and LOU usage is as follows:

- ACL1 Layer 4 operations: 5
- ACL2 Layer 4 operations: 4
- LOUs: 4

An explanation of the LOU usage follows:

- LOU 1 stores “gt 10” and “lt 9”
- LOU 2 stores “gt 11” and “neq 6”
- LOU 3 stores “gt 20” (with space for one more)
- LOU 4 stores “range 11 13” (range needs the entire LOU)

Information About ACL Support

ACLs can be processed in hardware by the Policy Feature Card (PFC), any Distributed Forwarding Cards (DFCs), or in software by the route processor (RP):

- ACL flows that match a “deny” statement in standard and extended ACLs (input and output) are dropped in hardware if “ip unreachables” is disabled.
- ACL flows that match a “permit” statement in standard and extended ACLs (input and output) are processed in hardware.
- VLAN ACL (VACL) and port ACL (PACL) flows are processed in hardware. If a field that is specified in a VACL or PACL is not supported by hardware processing, then that field is ignored (for example, the **log** keyword in an ACL), or the whole configuration is rejected (for example, a VACL containing IPX ACL parameters).
- IPv6 ACLs use 32 bit encoding.
- VACL logging is processed in software.
- VACL is not supported for IPX access lists.
- VACL supports only deny packet logging.
- Dynamic ACL flows are processed in hardware.
- Idle timeout is processed in software.

  Note Idle timeout is not configurable. Cisco IOS Release 15.1SY does not support the **access-enable host timeout** command.

- Except on MPLS interfaces, reflexive ACL flows are processed in hardware after the first packet in a session is processed in software on the RP.
- Reflexive ACL flows are not accelerated in hardware for traffic from IP to various tags and traffic from various tags to IP. Reflexive ACL flows are also not accelerated in hardware for any traffic coming in and going out of all tunnel interfaces.
- IP accounting for an ACL access violation on a given port is supported only for the ACL packets that are deny leaked, by forwarding all denied packets for that port to the RP for software processing without impacting other flows.
- MAC ACLs are supported in hardware on switch ports (MAC PACLs) or on VLANs as part of VACLs.

- The PFC does not provide hardware support for Cisco IOS IPX ACLs. Cisco IOS IPX ACLs are supported in software on the RP.
- Extended name-based MAC address ACLs are supported in hardware.
- The following ACL types are processed in software:
  - Internetwork Packet Exchange (IPX) access lists
  - Standard XNS access list
  - Extended XNS access list
  - DECnet access list
  - Protocol type-code access list

Note IP packets with a header length of less than five will not be access controlled.

- Unless you configure optimized ACL logging (OAL), flows that require logging are processed in software without impacting nonlogged flow processing in hardware (see the "Optimized ACL Logging" section on page 1-13).
- The forwarding rate for software-processed flows is substantially less than for hardware-processed flows.
- With the hardware statistics feature enabled, when you enter the **show ip access-list** command, the match count displayed includes packets processed in hardware.
- When you enter the **ip unreachables config** command on the PFC interface, the hardware behavior remains unaltered.

  Note See the "Restrictions for eFSU" section on page 1-2 for information about some release-specific restrictions.

Policy-Based ACLs (PBACLs)

Restrictions for PBACLs

- PBACLs are supported on Layer 3 interfaces (such as routed interfaces and VLAN interfaces).
- The PBACL feature only supports IPv4 ACEs.
- The PBACL feature supports only Cisco IOS ACLs. It is not supported with any other features. The keywords **reflexive** and **evaluate** are not supported.
- The PBACL feature supports only named Cisco IOS ACLs. It does not support numbered ACLs.
- Feature interactions for policy-based ACLs are the same as for Cisco IOS ACLs.

Information about PBACLs

PBACLs provide the capability to apply access control policies across object groups. An object group is a group of users or servers.

You define an object group as a group of IP addresses or as a group of protocol ports. You then create access control entries (ACEs) that apply a policy (such as permit or deny) to the object group. For example, a policy-based ACE can permit a group of users to access a group of servers.

An ACE defined using a group name is equivalent to multiple ACEs (one applied to each entry in the object group). The system expands the PBACL ACE into multiple Cisco IOS ACEs (one ACE for each entry in the group) and populates the ACEs in the TCAM. Therefore, the PBACL feature reduces the number of entries you need to configure but does not reduce TCAM usage.

If you make changes in group membership, or change the contents of an ACE that uses an access group, the system updates the ACEs in the TCAM. The following types of changes trigger the update:

- Adding a member to a group
- Deleting a member from a group
- Modifying the policy statements in an ACE that uses an access group

You configure a PBACL using extended Cisco IOS ACL configuration commands. As with regular ACEs, you can associate the same access policy with one or more interfaces.

When you configure an ACE, you can use an object group to define the source, the destination, or both.

How to Configure PBACLs

- Configuring a PBACL IP Address Object Group, page 1-6
- Configuring a PBACL Protocol Port Object Group, page 1-7
- Configuring ACLs with PBACL Object Groups, page 1-7
- Configuring PBACL on an Interface, page 1-8

Configuring a PBACL IP Address Object Group

To create or modify a PBACL IP address object group, perform this task:

	Command	Purpose
Step 1	`Router(config)#` **`object-group ip address`** *`object_group_name`*	Defines object group name and enters IP-address object-group configuration mode.
Step 2	`Router(config-ipaddr-ogroup)# {`*`ip_address mask`*`} | {`**`host`** `{`*`name`* `|` *`ip_address`*`} }`	Configures a member of the group. The member is either a network address plus mask or a host (identified by host name or IP address).
Step 3	`Router(config-ipaddr-ogroup)# {`**`end`**`} | {`**`exit`**`}`	To leave the configuration mode, enter the **end** command. To leave the IP-address object-group configuration mode, enter the **exit** command.

The following example creates an object group with three hosts and a network address:

```
Router(config)# object-group ip address myAG
Router(config-ipaddr-pgroup)# host 10.20.20.1
Router(config-ipaddr-pgroup)# host 10.20.20.5
Router(config-ipaddr-pgroup)# 10.30.0.0 255.255.0.0
```

Configuring a PBACL Protocol Port Object Group

To create or modify a PBACL protocol port object group, perform this task:

Command	Purpose
`Router(config)#` **`object-group ip port`** *`object_group_name`*	Defines object group name and enters port object-group configuration mode.
`Router(config-port-ogroup)# {`**`eq`** *`number`*`}	{`**`gt`** *`number`*`}	{`**`lt`** *`number`*`}	{`**`neq`** *`number`*`}	{`**`range`** *`number number`*`}`	Configures a member of the group. The member is either equal to or not equal to a port number, less than or greater than a port number, or a range of port numbers.
`Router(config-port-ogroup)#` **`end`** `|` **`exit`**	To leave the configuration mode, enter the **end** command. To leave the port object-group configuration mode, enter the **exit** command.

The following example creates a port object group that matches protocol port 100 and any port greater than 200, except 300:

```
Router(config)# object-group ip port myPG
Router(config-port-pgroup)# eq 100
Router(config-port-pgroup)# gt 200
Router(config-port-pgroup)# neq 300
```

Configuring ACLs with PBACL Object Groups

To configure an ACL to use a PBACL object group, perform this task:

	Command	Purpose
Step 1	`Router(config)#` **`ip access-list extended`** *`acl_name`*	Defines an extended ACL with the specified name and enters extended-ACL configuration mode.
Step 2	`Router(config-ext-nacl)#` **`permit tcp addrgroup`** *`object_group_name`* **`addrgroup`** *`object_group_name`*	Configures an ACE for TCP traffic using IP address object group as the source policy and an object group as the destination policy.
Step 3	`Router(config-ext-nacl)#` **`exit`**	Exits extended ACL configuration mode.

The following example creates an access list that permits packets from the users in myAG if the protocol ports match the ports specified in myPG:

```
Router(config)# ip access-list extended my-pbacl-policy
Router(config-ext-nacl)# permit tcp addrgroup myAG portgroup myPG any
Router(config-ext-nacl)# deny tcp any any
Router(config-ext-nacl)# exit
Router(config)# exit
Router# show ip access-list my-pbacl-policy
Extended IP access list my-pbacl-policy
10  permit tcp addrgroup AG portgroup PG any
20  permit tcp any any
```

```
Router# show ip access-list my-pbacl-policy expand
Extended IP access list my-pbacl-policy expanded
20 permit tcp host 10.20.20.1 eq 100 any
20 permit tcp host 10.20.20.1 gt 200 any
20 permit tcp host 10.20.20.1 neq 300 any
20 permit tcp host 10.20.20.5 eq 100 any
20 permit tcp host 10.20.20.5 gt 200 any
20 permit tcp host 10.20.20.5 neq 300 any
20 permit tcp 10.30.0.0 255.255.0.0 eq 100 any
20 permit tcp 10.30.0.0 255.255.0.0 gt 200 any
20 permit tcp 10.30.0.0 255.255.0.0 neq 300 any
```

Configuring PBACL on an Interface

To configure a PBACL on an interface, use the **ip access-group** command. The command syntax and usage is the same as for Cisco IOS ACLs. For additional information, see the "Restrictions for Cisco IOS ACLs" section on page 1-2.

The following example shows how to associate access list my-pbacl-policy with VLAN 100:

```
Router(config)# int vlan 100
Router(config-if)# ip access-group mp-pbacl-policy in
```

MAC ACLs

- How to Configure Protocol-Independent MAC ACL Filtering, page 1-8
- How to Enable VLAN-Based MAC QoS Filtering, page 1-10
- Configuring MAC ACLs, page 1-10

Note You can use MAC ACLs with VLAN ACLs (VACLs). For more information, see Chapter 1, "VLAN ACLs (VACLs)."

How to Configure Protocol-Independent MAC ACL Filtering

Protocol-independent MAC ACL filtering applies MAC ACLs to all ingress traffic types (for example, IPv4 traffic, IPv6 traffic, and MPLS traffic, in addition to MAC-layer traffic).

You can configure these interface types for protocol-independent MAC ACL filtering:

- VLAN interfaces
- Routed interfaces
- Physical LAN ports
- Logical LAN subinterfaces

Ingress traffic permitted or denied by a MAC ACL on an interface configured for protocol-independent MAC ACL filtering is processed by egress interfaces as MAC-layer traffic. You cannot apply egress IP ACLs to traffic that was permitted or denied by a MAC ACL on an interface configured for protocol-independent MAC ACL filtering.

To configure protocol-independent MAC ACL filtering, perform this task:

	Command	Purpose
Step 1	`Router(config)# interface {{vlan vlan_ID} | {type slot/port[.subinterface]} | {port-channel number[.subinterface]}}`	Selects the interface to configure.
Step 2	`Router(config-if)# [no] mac packet-classify {input | output | use {ce_cos {input | output} | dscp {input | output}}}`	Enables protocol-independent MAC ACL filtering on the interface. By default, the **mac packet-classify** configuration command is disabled.

- Do not configure protocol-independent MAC ACL filtering on VLAN interfaces where you have configured an IP address.
- When the mac acl filtering is enabled, all other protocol features such as RACL, microflow policing will be ignored in the hardware.

This example shows how to configure VLAN interface 4018 for protocol-independent MAC ACL filtering and how to verify the configuration:

```
Router(config)# interface vlan 4018
Router(config-if)# mac packet-classify
Router(config-if)# end
Router# show running-config interface vlan 4018 | begin 4018
interface Vlan4018
mtu 9216
ipv6 enable
mac packet-classify
end
```

This example shows how to configure Gigabit Ethernet interface 6/1 for protocol-independent MAC ACL filtering and how to verify the configuration:

```
Router(config)# interface gigabitethernet 6/1
Router(config-if)# mac packet-classify
Router(config-if)# end
Router# show running-config interface gigabitethernet 6/1 | begin 6/1
interface GigabitEthernet6/1
mtu 9216
no ip address
mac packet-classify
mpls l2transport route 4.4.4.4 4094
end
```

This example shows how to configure Gigabit Ethernet interface 3/24, subinterface 4000, for protocol-independent MAC ACL filtering and how to verify the configuration:

```
Router(config)# interface gigabitethernet 3/24.4000
Router(config-if)# mac packet-classify
Router(config-if)# end
Router# show running-config interface gigabitethernet 3/24.4000 | begin 3/24.4000
interface GigabitEthernet3/24.4000
encapsulation dot1Q 4000
mac packet-classify
mpls l2transport route 4.4.4.4 4000
end
```

How to Enable VLAN-Based MAC QoS Filtering

You can globally enable or disable VLAN-based QoS filtering in MAC ACLs. VLAN-based QoS filtering in MAC ACLs is disabled by default.

To enable VLAN-based QoS filtering in MAC ACLs, perform this task:

Command	Purpose
`Router(config)# mac packet-classify use outer-vlan`	Enables VLAN-based QoS filtering in MAC ACLs. The VLAN field in MAC ACLs will be matched for outer-vlan tag. The options are **in** (apply to ingress MAC ACLs) and **out** (apply to egress MAC ACLs).

To disable VLAN-based QoS filtering in MAC ACLs, perform this task:

Command	Purpose
`Router(config)# no mac packet-classify use outer-vlan`	Disables VLAN-based QoS filtering in MAC ACLs.

Configuring MAC ACLs

You can configure named ACLs that filter IP, IPX, DECnet, AppleTalk, VINES, or XNS traffic based on MAC addresses.

You can configure MAC ACLs that do VLAN-based filtering or CoS-based filtering or both.

You can globally enable or disable VLAN-based QoS filtering in MAC ACLs (disabled by default).

To configure a MAC ACL, perform this task:

	Command	Purpose
Step 1	Router(config)# **mac host** *name mac_addr*	(Optional) Assigns a name to a MAC address.
Step 2	Router(config)# **mac access-list extended** *list_name*	Configures a MAC ACL.
Step 3	Router(config-ext-macl)# {**permit** | **deny**} {*src_mac_mask* | {**host name** *src_mac_name*} | **any**} {*dest_mac_mask* | {**host name** *dst_mac_name*} | **any**} [{*protocol_keyword* | {*ethertype_number ethertype_mask*}} [**vlan** *vlan_ID*] [**cos** *cos_value*]]	Configures an access control entry (ACE) in a MAC ACL. The source and destination MAC addresses can be specified by MAC address masks or by names created with the **mac host** command.

- Cisco IOS Release 15.1SY supports the **vlan** and **cos** keywords.
- The **vlan** keyword for VLAN-based QoS filtering in MAC ACLs can be globally enabled or disabled and is disabled by default.
- You can enter MAC addresses as three 2-byte values in dotted hexadecimal format. For example, 0030.9629.9f84.
- You can enter MAC address masks as three 2-byte values in dotted hexadecimal format. Use 1 bits as wildcards. For example, to match an address exactly, use 0000.0000.0000 (can be entered as 0.0.0).
- You can enter an EtherType and an EtherType mask as hexadecimal values.

- Entries without a protocol parameter match any protocol.
- ACL entries are scanned in the order you enter them. The first matching entry is used. To improve performance, place the most commonly used entries near the beginning of the ACL.
- An implicit **deny any any** entry exists at the end of an ACL unless you include an explicit **permit any any** entry at the end of the list.
- All new entries to an existing list are placed at the end of the list. You cannot add entries to the middle of a list.
- This list shows the EtherType values and their corresponding protocol keywords:
  - 0x0600—xns-idp—Xerox XNS IDP
  - 0x0BAD—vines-ip—Banyan VINES IP
  - 0x0baf—vines-echo—Banyan VINES Echo
  - 0x6000—etype-6000—DEC unassigned, experimental
  - 0x6001—mop-dump—DEC Maintenance Operation Protocol (MOP) Dump/Load Assistance
  - 0x6002—mop-console—DEC MOP Remote Console
  - 0x6003—decnet-iv—DEC DECnet Phase IV Route
  - 0x6004—lat—DEC Local Area Transport (LAT)
  - 0x6005—diagnostic—DEC DECnet Diagnostics
  - 0x6007—lavc-sca—DEC Local-Area VAX Cluster (LAVC), SCA
  - 0x6008—amber—DEC AMBER
  - 0x6009—mumps—DEC MUMPS
  - 0x0800—ip—Malformed, invalid, or deliberately corrupt IP frames
  - 0x8038—dec-spanning—DEC LANBridge Management
  - 0x8039—dsm—DEC DSM/DDP
  - 0x8040—netbios—DEC PATHWORKS DECnet NETBIOS Emulation
  - 0x8041—msdos—DEC Local Area System Transport
  - 0x8042—etype-8042—DEC unassigned
  - 0x809B—appletalk—Kinetics EtherTalk (AppleTalk over Ethernet)
  - 0x80F3—aarp—Kinetics AppleTalk Address Resolution Protocol (AARP)

This example shows how to create a MAC-Layer ACL named mac_layer that denies dec-phase-iv traffic with source address 0000.4700.0001 and destination address 0000.4700.0009, but permits all other traffic:

```
Router(config)# mac access-list extended mac_layer
Router(config-ext-macl)# deny 0000.4700.0001 0.0.0 0000.4700.0009 0.0.0 dec-phase-iv
Router(config-ext-macl)# permit any any
```

ARP ACLs

This section describes how to configure ARP ACLs. You can configure named ACLs that filter ARP traffic (EtherType 0x0806). To configure an ARP ACL, perform this task:

	Command	Purpose
Step 1	`Router(config)# arp access-list list_name`	Configures an ARP ACL.
Step 2	`Router(config-arp-nacl)# {permit | deny} {ip {any | host sender_ip | sender_ip sender_ip_wildcardmask} mac any`	Configures an access control entry (ACE) in an ARP ACL.

- This publication describes the ARP ACL syntax that is supported in hardware by the PFC. Any other ARP ACL syntax displayed by the CLI help when you enter a question mark ("?") is not supported and cannot be used to filter ARP traffic for QoS.
- ACLs entries are scanned in the order you enter them. The first matching entry is used. To improve performance, place the most commonly used entries near the beginning of the ACL.
- An implicit **deny ip any mac any** entry exists at the end of an ACL unless you include an explicit **permit ip any mac any** entry at the end of the list.
- All new entries to an existing list are placed at the end of the list. You cannot add entries to the middle of a list.
- The PFC does not apply IP ACLs to ARP traffic.
- You cannot apply microflow policing to ARP traffic.

This example shows how to create an ARP ACL named arp_filtering that only permits ARP traffic from IP address 1.1.1.1:

```
Router(config)# arp access-list arp_filtering
Router(config-arp-nacl)# permit ip host 1.1.1.1 mac any
```

Configuring IPv6 Address Compression

ACLs are implemented in hardware in the PFC and DFCs, which uses the source or destination IP address and port number in the packet to index the ACL table. The index has a maximum address length of 128 bits.

The IP address field in an IPv6 packet is 128 bits, and the port field is 16 bits. To use full IPv6 addresses in the ACL hardware table, you can turn on compression of IPv6 addresses using the **mls ipv6 acl compress address unicast** command. This feature compresses the IPv6 address (including port) into 128 bits by removing 16 unused bits from the IPv6 address. Compressible address types can be compressed without losing any information. See Table 1-1 for details about the compression methods.

By default, the command is set for no compression.

Caution Do not enable the compression mode if you have noncompressible address types in your network. A list of compressible address types and the address compression method are listed in Table 1-1.

Table 1-1 Compressible Address Types and Methods

Address Type	Compression Method
EUI-64 based on MAC address	This address is compressed by removing 16 bits from bit locations [39:24]. No information is lost when the hardware compresses these addresses.
Embedded IPv4 address	This address is compressed by removing the upper 16 bits. No information is lost when the hardware compresses these addresses.
Link Local	These addresses are compressed by removing the zeros in bits [95:80] and are identified using the same packet type as the embedded IPv4 address. No information is lost when the hardware compresses these addresses.
Others	If the IPv6 address does not fall into any of the above categories, it is classified as other. If the IPv6 address is classified as other, the following occurs: • If the compress mode is on, the IPv6 address is compressed similarly to the EUI-64 compression method (removal of bits [39:24]) to allow for the Layer 4 port information to be used as part of the key used to look up the QoS TCAM, but Layer 3 information is lost. • If the global compression mode is off, the entire 128 bits of the IPv6 address are used. The Layer 4 port information cannot be included in the key to look up the QoS TCAM because of the size constraints on the IPv6 lookup key.

To turn on the compression of IPv6 addresses, enter the **mls ipv6 acl compress address unicast** command. To turn off the compression of IPv6 addresses, enter the **no** form of this command.

This example shows how to turn on address compression for IPv6 addresses:

```
Router(config)# mls ipv6 acl compress address unicast
Router(config)#
```

This example shows how to turn off address compression for IPv6 addresses:

```
Router(config)# no mls ipv6 acl compress address unicast
Router(config)#
```

Optimized ACL Logging

- Restrictions for OAL, page 1-13
- Information about OAL, page 1-14
- How to Configure OAL, page 1-14

Restrictions for OAL

- OAL and VACL capture are incompatible. Do not configure both features on the switch. With OAL configured, use SPAN to capture traffic.
- OAL checks for conflicts with other features using capture like VACL capture, Lawful Intercept (LI), and IPv6 learning.
- OAL supports only IPv4 unicast packets.
- OAL is not supported with port ACLs (PACLs).

- OAL does not provide hardware support for the following:
  - Reflexive ACLs
  - ACLs used to filter traffic for other features (for example, QoS)
  - ACLs for unicast reverse path forwarding (uRPF) check exceptions
  - Exception packets (for example, TTL failure and MTU failure)
  - Packets with IP options
  - Packets addressed at Layer 3 to the router
  - Packets sent to the RP to generate ICMP unreachable messages
  - Packets being processed by features not accelerated in hardware
- To provide OAL support for denied packets, enter the **mls rate-limit unicast ip icmp unreachable acl-drop 0** command.
- OAL and the **mls verify ip length minimum** command are incompatible. Do not configure both.

Information about OAL

OAL provides hardware support for ACL logging. Unless you configure OAL, packets that require logging are processed completely in software on the RP. OAL permits or drops packets in hardware on the PFC or DFC and uses an optimized routine to send information to the RP to generate the logging messages.

How to Configure OAL

- Configuring OAL Global Parameters, page 1-14
- Configuring OAL on an Interface, page 1-15
- Displaying OAL Information, page 1-15
- Clearing Cached OAL Entries, page 1-15

Configuring OAL Global Parameters

To configure global OAL parameters, perform this task:

Command	Purpose
`Router(config)#` **`logging ip access-list cache`** `{{`**`entries`** *`number_of_entries`*`}	{`**`interval`** *`seconds`*`}	{`**`rate-limit`** *`number_of_packets`*`}	{`**`threshold`** *`number_of_packets`*`}}`	Sets OAL global parameters.

- **entries** *number_of_entries*
  - Sets the maximum number of entries cached.
  - Range: 0–1,048,576 (entered without commas).
  - Default: 8192.

- **interval** *seconds*
  - Sets the maximum time interval before an entry is sent to be logged. Also if the entry is inactive for this duration it is removed from the cache.
  - Range: 5–86,400 (1440 minutes or 24 hours, entered without commas).
  - Default: 300 seconds (5 minutes).
- **rate-limit** *number_of_packets*
  - Sets the number of packets logged per second in software.
  - Range: 10–1,000,000 (entered without commas).
  - Default: 0 (rate limiting is off and all packets are logged).
- **threshold** *number_of_packets*
  - Sets the number of packet matches before an entry is logged.
  - Range: 1–1,000,000 (entered without commas).
  - Default: 0 (logging is not triggered by the number of packet matches).

Configuring OAL on an Interface

To configure OAL on an interface, perform this task:

	Command	Purpose
Step 1	`Router(config)#` **`interface`** `{{`*`type slot/port`*`}`	Specifies the interface to configure.
Step 2	`Router(config-if)#` **`logging ip access-list cache in`**	Enables OAL for ingress traffic on the interface.
Step 3	`Router(config-if)#` **`logging ip access-list cache out`**	Enables OAL for egress traffic on the interface.

Displaying OAL Information

To display OAL information, perform this task:

Command	Purpose
`Router#` **`show logging ip access-list cache`**	Displays OAL information.

Clearing Cached OAL Entries

To clear cached OAL entries, perform this task:

Command	Purpose
`Router#` **`clear logging ip access-list cache`**	Clears cached OAL entries.

Dry Run Support for ACLs

- Restrictions for Dry Run Support, page 1-16

Restrictions for Dry Run Support

- Dry Run is supported only for IPv4 RACLs and can only be applied on interfaces.
- Dry Run is supported only with named ACLs (Standard or Extended) and not with numbered ACLs.
- Only one Dry Run session is allowed at a time with a single ACL or multiple ACLs in the Dry Run session.
- The Dry Run session ACL or ACLs are removed when an ACL or ACLs are changed under configuration mode.
- Exiting the Dry Run session does not clear the existing configuration. Clear the existing session before starting a new configuration.
- The validation process may abort if there are configuration or hardware changes made during the validation process.
- Dry Run mode does not support committing the changes to the running configuration.
- Dry Run is not supported for ACLs used in QoS policies.
- Dry Run is not supported for ACLs having hardware statistics enabled.
- You can access the switch using another Telnet session while a Dry Run session is in progress.

Information About Dry Run Support

In other releases, when a new feature is applied on an interface configured along with other features, and if the new feature does not fit in the TCAM, then existing features are also affected and removed from the TCAM. To incrementally update the feature and see whether the feature fits into the TCAM without installing it, the switch provides a Dry Run support, where applications can send regular requests to test whether the request can be programmed successfully or not. The switch receives the dry run request and calculates the total TCAM resources required for the request and compares those resources against the available free resources. If the request fits in successfully, then the switch returns a success, else the switch returns a failure. The Dry Run support helps applications make intelligent decisions.

How to Configure Dry Run Support for ACLs

To configure the Dry Run support, perform this task:

	Command	Purpose
Step 1	`Router(config)# `**`configure session`** *`session_name`*	Creates a configuration session and enters the dry run mode
Step 2	`Router(dry-run-config)# {default | exit | ip | no | validate}`	Choose the option to configure the dry run session
Step 3	`Router(dry-run-config)# `**`ip access-list`**` {extended | standard}` *`acl_name`*	Choose the type of ACL

The following example configures the dry run support for a session with an existing ACL RACL10K:

```
Router(config)# configure session test
Router(dry-run-config)# ip access-list extended RACL10K
Router(dr-config-ext-nacl)# permit tcp host 10.20.0.1 host 11.20.0.1
Router(dr-config-ext-nacl)# permit tcp host 10.20.0.2 host 11.20.0.2
Router(dr-config-ext-nacl)# permit tcp host 10.20.0.3 host 11.20.0.3
Router(dr-config-ext-nacl)# permit tcp host 10.20.0.4 host 11.20.0.4
Router(dr-config-ext-nacl)# permit tcp host 10.20.0.5 host 11.20.0.5

Router(dr-config-ext-nacl)# exit

Router(dry-run-config)# validate

Router(dry-run-config)# exit
Router#
.Feb 23 2010 13:46:52.528: Validation is in progress !!
.Feb 23 2010 13:46:52.528: Please try again later.
.Feb 23 2010 13:46:53.136: %FM-6-SESSION_VALIDATION_RESULT_INFO: Session Validation Result
:  "Validation Completed Sucessfully."
. Please use 'show config session test status' to get more details of the config
validation status

Router# show configuration session test status
====================================
Status of last config validation:
Timestamp: 2010-02-23@13:46:51
======================================
SLOT = [1]    Result = Configuration will fit in TCAM
SLOT = [2]    Result = Configuration will fit in TCAM
SLOT = [5]    Result = Configuration will fit in TCAM

Router# clear configuration session test

Router# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.

Router(config)# ip access-list extended RACL10K
Router(config-ext-nacl)# permit tcp host 10.20.0.1 host 11.20.0.1
Router(config-ext-nacl)# permit tcp host 10.20.0.2 host 11.20.0.2
Router(config-ext-nacl)# permit tcp host 10.20.0.3 host 11.20.0.3
Router(config-ext-nacl)# permit tcp host 10.20.0.4 host 11.20.0.4
Router(config-ext-nacl)# permit tcp host 10.20.0.5 host 11.20.0.5
Router(config-ext-nacl)# end

Router#
```

Hardware ACL Statistics

- Restrictions for Hardware ACL Statistics, page 1-17
- Information About Hardware ACL Statistics, page 1-18
- How to Configure Hardware ACL Statistics, page 1-18

Restrictions for Hardware ACL Statistics

- Hardware ACL statistics are supported for IPv4 and IPv6 RACLs in both ingress and egress directions.

- In IPv4, hardware ACL statistics are supported for both numbered and named ACLs.
- The hardware statistics is retrieved by polling the hardware once every 60 seconds.
- The hardware statistics is lost after SSO (Stateful Switchover).
- The hardware statistics is maintained on an ACL basis. If there are multiple interfaces using the same ACL, the statistics will be aggregated.
- Hardware statistics is disabled when ODM (Order Dependent Merge) optimizations are enabled.

Information About Hardware ACL Statistics

Using the hardware ACL statistics, the hardware counters for a given ACL are gathered, aggregated, and displayed in the IOS access-list output.

The ACE hit count is retrieved from the hardware and can be viewed using the following commands:

show ip access-list and **show ipv6 access-list**

Hardware statistics is disabled by default. To enable or disable hardware statistics, enter the command for hardware statistics.

How to Configure Hardware ACL Statistics

The following example enables hardware statistics for ACL racl1:

```
Router(config)# ip access-list extended racl1
Router(config-ext-nacl)# [no] hardware statistics
Router(config-ext-nacl)# permit ip host 1.1.1.1 host 2.2.2.2
Router(config-ext-nacl)# permit ip host 3.3.3.3 host 4.4.4.4
Router(config-ext-nacl)# deny ip any any
Router(config-ext-nacl)# end
```

The following example displays the hardware statistics for ACL racl1:

```
Router# show ip access-lists racl1
Extended IP access list racl1
    hardware statistics
    10 permit ip host 1.1.1.1 host 2.2.2.2
acl hw hit count 5
    20 permit ip host 3.3.3.3 host 4.4.4.4
acl hw hit count 20
    30 deny ip any any
```

The hardware statistics for each ACE is seen after the **acl hw hit count** string and indicates the number of packets switched in hardware.

Tip For additional information about Cisco Catalyst 6500 Series Switches (including configuration examples and troubleshooting information), see the documents listed on this page:

http://www.cisco.com/en/US/products/hw/switches/ps708/tsd_products_support_series_home.html

Participate in the Technical Documentation Ideas forum