Experiment (9)

Cisco IOS access lists

Objective

In this experiment you will learn

- ACL types.
- Filter traffic using ACL
- Use ACL for a variety of operations
- Configure ACL access list

Introduction:

Access lists are an integral part of working with routers, and they're vital to security. In the Cisco IOS, an access control list is a record that identifies and manages traffic. After identifying that traffic, an administrator can specify various events that can happen to that traffic.
This experiment provides sample configurations for commonly used IP Access Control Lists (ACLs), which filter IP packets based on:

- Source address
- Destination address
- Type of packet
- Any combination of these items

In order to filter network traffic, ACLs control whether routed packets are forwarded or blocked at the router interface. Your router examines each packet in order to determine whether to forward or drop the packet based on the criteria that you specify within the ACL. ACL criteria include:

- Source address of the traffic
- Destination address of the traffic
- Upper-layer protocol

Complete these steps in order to construct an ACL as the examples in this document show:

1. Create an ACL.
2. Apply the ACL to an interface.

The IP ACL is a sequential collection of permit and deny conditions that apply to an IP packet. The router tests packets against the conditions in the ACL one at a time.
The first match determines whether the Cisco IOS® Software accepts or rejects the packet. Because the Cisco IOS Software stops testing conditions after the first match, the order of the conditions is critical. If no conditions match, the router rejects the packet because of an implicit deny all clause.

The most common type of ACL:

IP ACLs are the most popular type of access lists because IP is the most common type of traffic.
There are two types of IP ACLs: standard and extended.

- Standard IP ACLs can only control traffic based on the SOURCE IP address.
- Extended IP ACLs are far more powerful; they can identify traffic based on source IP, source port, destination IP, and destination port.

The most common numbers for IP ACLs?

The most common numbers used for IP ACLs are 1 to 99 for standard lists and 100 to 199 for extended lists. However, many other ranges are also possible.

- Standard IP ACLs: 1 to 99 and 1300 to 1999
- Extended IP ACLs: 100 to 199 and 2000 to 2699

Filter traffic using ACLs:

You can use ACLs to filter traffic according to the "three P's"—per protocol, per interface, and per direction. You can only have one ACL per protocol (e.g., IP or IPX), one ACL per interface (e.g., FastEthernet0/0), and one ACL per direction (i.e., IN or OUT).

ACL can help protect your network from viruses

You can use an ACL as a packet sniffer to list packets that meet a certain requirement. For example, if there's a virus on your network that's sending out traffic over IRC port 194, you could create an extended ACL (such as number 101) to identify that traffic. You could then use the *debug ip packet 101 detail* command on your Internet-facing router to list all of the source IP addresses that are sending packets on port 194.

The order of operations in an ACL

Routers process ACLs from top to bottom. When the router evaluates traffic against the list, it starts at the beginning of the list and moves down, either permitting or denying traffic as it goes. When it has worked its way through the list, the processing stops.
That means whichever rule comes first takes precedence. If the first part of the ACL denies traffic, but a lower part of the ACL allows it, the router will still deny the traffic. Let's look at an example:

```
Access-list 1 permit any
Access-list 1 deny host 10.1.1.1
Access-list 1 deny any
```

What does this ACL permit? The first line permits anything. Therefore, all traffic meets this requirement, so the router will permit all traffic, and processing will then stop.

*What about traffic you don't specifically address in an ACL?

At the end of an ACL is an implicit *deny* statement. Whether you see the statement or not, the router denies all traffic that doesn't meet a condition in the ACL. Here's an example:

```
Access-list 1 deny host 10.1.1.1
```

```
Access-list 1 deny 192.168.1.0 0.0.0.255
```

What traffic does this ACL permit? None: The router denies all traffic because of the implicit *deny* statement. In other words, the ACL really looks like this:

```
Access-list 1 deny host 10.1.1.1
Access-list 1 deny 192.168.1.0 0.0.0.255
Access-list 1 deny ANY
```

ACL name

Numbers—who needs numbers? You can also name your ACLs so you can more easily identify their purpose. You can name both standard and extended ACLs. Here's an example of using a named ACL:

```
router(config)# ip access-list ?
  extended        Extended Access List
  log-update      Control access list log updates
  logging         Control access list logging
  resequence      Resequence Access List
  standard        Standard Access List
router(config)# ip access-list extended test
router(config-ext-nacl)#
router(config-ext-nacl)# 10 deny ip any host 192.168.1.1
router(config-ext-nacl)# exit
router(config)# exit
router# show ip access-list
Extended              IP              access              list              test
    10 deny ip any host 192.168.1.1
```

A numbering sequence

In the "old days," you couldn't edit an ACL—you could only copy it to a text editor (such as Notepad), remove it, edit it in notepad, and then re-create it. In fact, this is still a good way to edit some Cisco configurations.

However, this approach can also create a security risk. During the time you've removed the ACL to modify it, the router isn't controlling traffic as needed. But it's possible to edit a numbered ACL with commands. Here's an example:

```
router(config)# access-list 75 permit host 10.1.1.1
router(config)#^Z
router# conf t
Enter configuration commands, one per line.  End with CNTL/Z.


router(config)# ip access-list standard 75


router(config-std-nacl)# 20 permit any
router(config-std-nacl)# no 10 permit 10.1.1.1
router(config-std-nacl)#^Z


router# show ip access-lists 75
Standard              IP              access              list              75
    20 permit any
router#
```

Other uses of an ACL

ACLs aren't just for filtering traffic. You can also use them for a variety of operations. Let's look at some of their possible other uses:

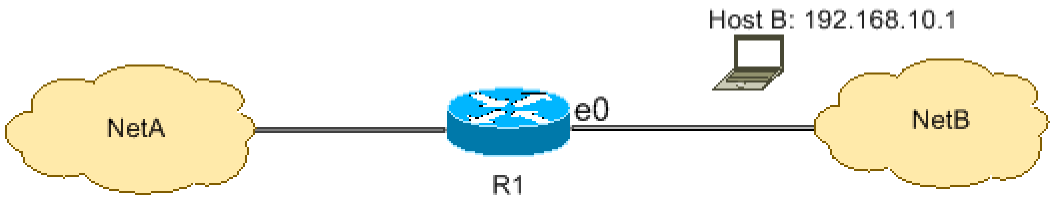
- **To control debug output:** You can use the *debug list X* command to control debug output. By using this command before another *debug* command, the command only applies to what you've defined in the list.
- **To control route access:** You can use a routing distribute-list ACL to only permit or deny certain routes either into or out of your routing protocol.
- **As a BGP AS-path ACL:** You can use regular expressions to permit or deny BGP routes.
- **For router management:** You can use an ACL to control which workstation or network manages your router with an ACL and an *access-class* statement to your VTY lines.
- **For encryption:** You can use ACLs to determine how to encrypt traffic. When encrypting traffic between two routers or a router and a firewall, you must tell the router what traffic to encrypt, what traffic to send unencrypted, and what traffic to drop.

Configure

These configuration examples use the most common IP ACLs.

Allow a Select Host to Access the Network

This figure shows a select host being granted permission to access the network. All traffic sourced from Host B destined to NetA is permitted, and all other traffic sourced from NetB destined to NetA is denied.



The output on the R1 table shows how the network grants access to the host. This output shows that:

- The configuration allows only the host with the IP address 192.168.10.1 through the Ethernet 0 interface on R1.
- This host has access to the IP services of NetA.
- No other host in NetB has access to NetA.
- No deny statement is configured in the ACL.

By default, there is an implicit deny all clause at the end of every ACL. Anything that is not explicitly permitted is denied.

R1

```
hostname R1
```

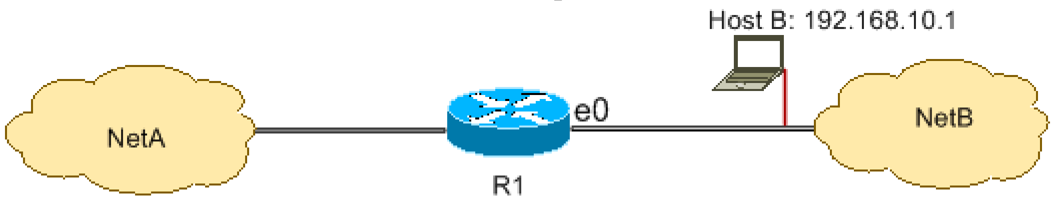
```
!
interface ethernet0
ip access-group 1 in
!
access-list 1 permit host 192.168.10.1
```

Note: The ACL filters IP packets from NetB to NetA, except packets sourced from NetB. Packets sourced from Host B to NetA are still permitted.

Note: The ACL **access-list 1 permit 192.168.10.1 0.0.0.0** is another way to configure the same rule.

Deny a Select Host to Access the Network

This figure shows that traffic sourced from Host B destined to NetA is denied, while all other traffic from the NetB to access NetA is permitted.



This configuration denies all packets from host 192.168.10.1/32 through Ethernet 0 on R1 and permits everything else. You must use the command **access list 1 permit any** to explicitly permit everything else because there is an implicit deny all clause with every ACL.

R1

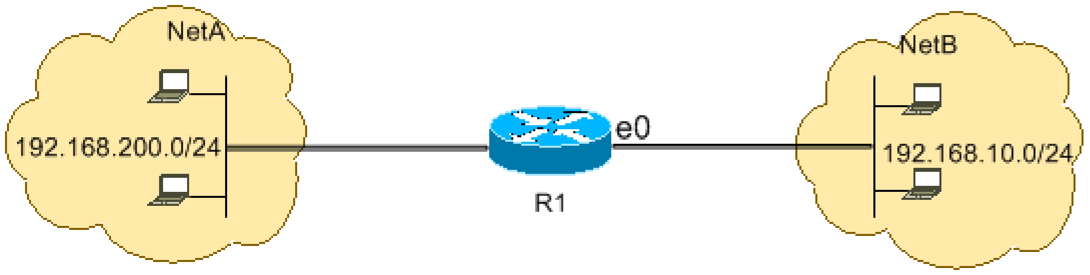
```
hostname R1
!
interface ethernet0
ip access-group 1 in
!
access-list 1 deny host 192.168.10.1
access-list 1 permit any
```

Note: The order of statements is critical to the operation of an ACL. If the order of the entries is reversed as this command shows, the first line matches every packet source address. Therefore, the ACL fails to block host 192.168.10.1/32 from accessing NetA.

```
access-list 1 permit any
access-list 1 deny host 192.168.10.1
```

Allow Access to a Range of Contiguous IP Addresses

This figure shows that all hosts in NetB with the network address 192.168.10.0/24 can access network 192.168.200.0/24 in NetA.



This configuration allows the IP packets with an IP header that has a source address in the network 192.168.10.0/24 and a destination address in the network 192.168.200.0/24 access to NetA. There is the implicit deny all clause at the end of the ACL which denies all other traffic passage through Ethernet 0 inbound on R1.

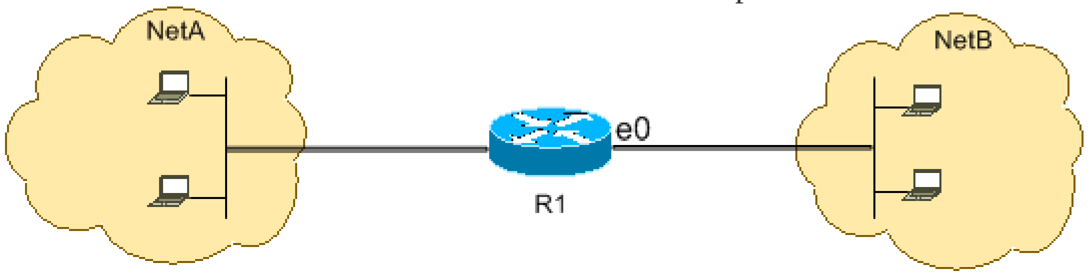
R1

```
hostname R1
!
interface ethernet0
ip access-group 101 in
!
access-list 101 permit ip 192.168.10.0  0.0.0.255
  192.168.200.0  0.0.0.255
```

Note: In the command **access-list 101 permit ip 192.168.10.0 0.0.0.255 192.168.200.0 0.0.0.255**, the "0.0.0.255" is the inverse mask of network 192.168.10.0 with mask 255.255.255.0. ACLs use the inverse mask to know how many bits in the network address need to match. In the table, the ACL permits all hosts with source addresses in the 192.168.10.0/24 network and destination addresses in the 192.168.200.0/24 network.

Deny Telnet Traffic (TCP, Port 23)

In order to meet higher security concerns, you might have to disable Telnet access to your private network from the public network. This figure shows how Telnet traffic from NetB (public) destined to NetA (private) is denied, which permits NetA to initiate and establish a Telnet session with NetB while all other IP traffic is permitted.



Telnet uses TCP, port 23. This configuration shows that all TCP traffic destined to NetA for port 23 is blocked, and all other IP traffic is permitted.

R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 deny tcp any any eq 23
access-list 102 permit ip any any
```

Allow Only Internal Networks to Initiate a TCP Session

This figure shows that TCP traffic sourced from NetA destined to NetB is permitted, while TCP traffic from NetB destined to NetA is denied.

The purpose of the ACL in this example is to:

- Allow hosts in NetA to initiate and establish a TCP session to hosts in NetB.
- Deny hosts in NetB from initiating and establishing a TCP session destined to hosts in NetA.

This configuration allows a datagram to pass through interface Ethernet 0 inbound on R1 when the datagram has:

- Acknowledged (ACK) or reset (RST) bits set (indicating an established TCP session)
- A destination port value greater than 1023

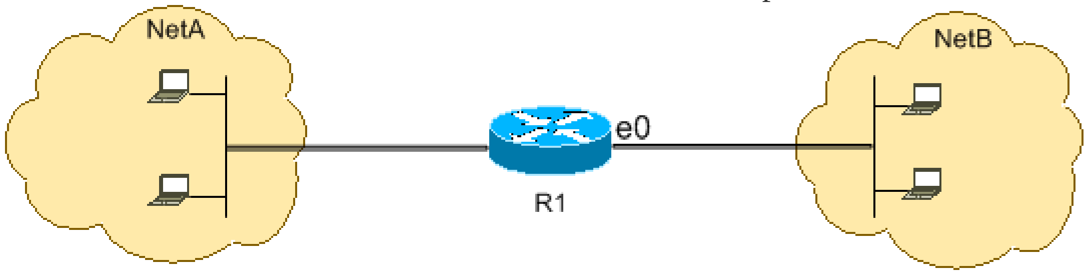
R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 permit tcp any any gt 1023 established
```

Since most of the well-known ports for IP services use values less than 1023, any datagram with a destination port less than 1023 or an ACK/RST bit not set is denied by ACL 102. Therefore, when a host from NetB initiates a TCP connection by sending the first TCP packet (without synchronize/start packet (SYN/RST) bit set) for a port number less than 1023, it is denied and the TCP session fails. The TCP sessions initiated from NetA destined to NetB are permitted because they have ACK/RST bit set for returning packets and use port values greater than 1023.

Deny FTP Traffic (TCP, Port 21)

This figure shows that FTP (TCP, port 21) and FTP data (port 20) traffic sourced from NetB destined to NetA is denied, while all other IP traffic is permitted.



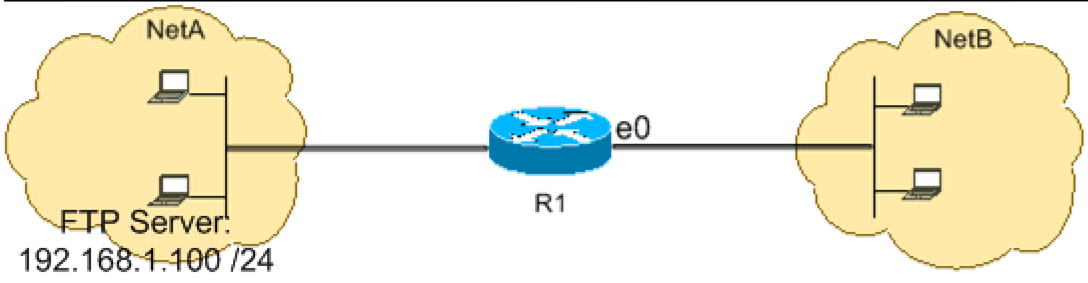
FTP uses port 21 and port 20. TCP traffic destined to port 21 and port 20 is denied and everything else is explicitly permitted.

R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 deny tcp any any eq ftp
access-list 102 deny tcp any any eq ftp-data
access-list 102 permit ip any any
```

Allow FTP Traffic (Active FTP)

FTP can operate in two different modes named active and passive. Refer to FTP Operation to understand how active and passive FTP works.
When FTP operates in active mode, the FTP server uses port 21 for control and port 20 for data. FTP server (192.168.1.100) is located in NetA. This figure shows that FTP (TCP, port 21) and FTP data (port 20) traffic sourced from NetB destined to FTP server (192.168.1.100) is permitted, while all other IP traffic is denied.

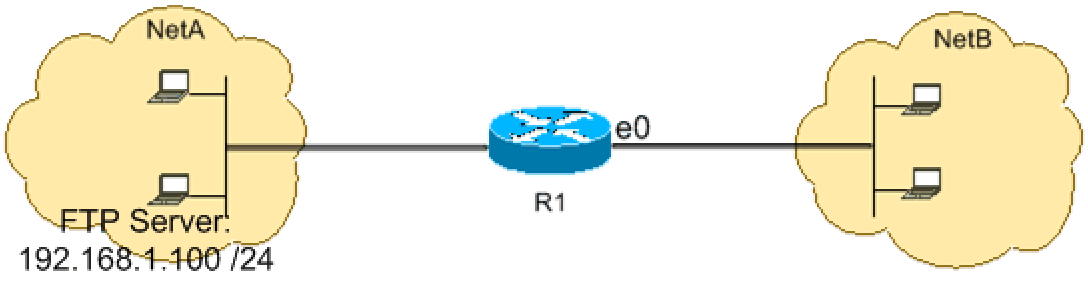


R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 permit tcp any host 192.168.1.100 eq ftp
access-list 102 permit tcp any host 192.168.1.100 eq ftp-data established
!
interface ethernet1
 ip access-group 110 in
!
access-list 110 permit host 192.168.1.100 eq ftp any established
access-list 110 permit host 192.168.1.100 eq ftp-data any
```

Allow FTP Traffic (Passive FTP)

FTP can operate in two different modes named active and passive. Refer to FTP Operation in order to understand how active and passive FTP works.
When FTP operates in passive mode, the FTP server uses port 21 for control and the dynamic ports greater than or equal to 1024 for data. FTP server (192.168.1.100) is located in NetA. This figure shows that FTP (TCP, port 21) and FTP data (ports greater than or equal to 1024) traffic sourced from NetB destined to FTP server (192.168.1.100) is permitted, while all other IP traffic is denied.

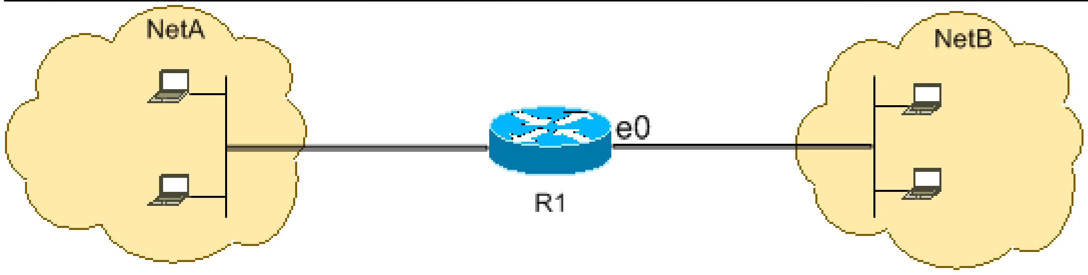


R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 permit tcp any host 192.168.1.100 eq ftp
access-list 102 permit tcp any host 192.168.1.100 gt 1024
!
interface ethernet1
 ip access-group 110 in
!
access-list 110 permit host 192.168.1.100 eq ftp any established
access-list 110 permit host 192.168.1.100 gt 1024 any established
```

Allow Pings (ICMP)

This figure shows that ICMP sourced from NetA destined to NetB is permitted, and pings sourced from NetB destined to NetA are denied.



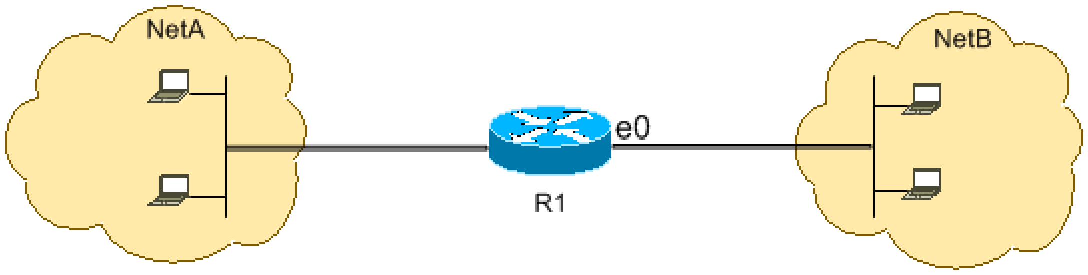
This configuration permits only echo-reply (ping response) packets to come in on interface Ethernet 0 from NetB towards NetA. However, the configuration blocks all echo-request ICMP packets when pings are sourced in NetB and destined to NetA. Therefore, hosts in NetA can ping hosts in NetB, but hosts in NetB cannot ping hosts in NetA.

R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 permit icmp any any echo-reply
```

Allow HTTP, Telnet, Mail, POP3, FTP

This figure shows that only HTTP, Telnet, Simple Mail Transfer Protocol (SMTP), POP3, and FTP traffic are permitted, and the rest of the traffic sourced from NetB destined to NetA is denied.



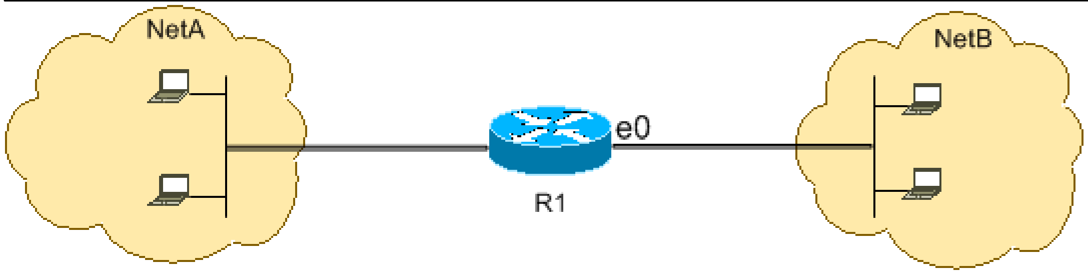
This configuration permits TCP traffic with destination port values that match WWW (port 80), Telnet (port 23), SMTP (port 25), POP3 (port 110), FTP (port 21), or FTP data (port 20). Notice an implicit deny all clause at the end of an ACL denies all other traffic, which does not match the permit clauses.

R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 102 permit tcp any any eq www
access-list 102 permit tcp any any eq telnet
access-list 102 permit tcp any any eq smtp
access-list 102 permit tcp any any eq pop3
access-list 102 permit tcp any any eq 21
access-list 102 permit tcp any any eq 20
```

Allow DNS

This figure shows that only Domain Name System (DNS) traffic is permitted, and the rest of the traffic sourced from NetB destined to NetA is denied.



This configuration permits TCP traffic with destination port value 53. The implicit deny all clause at the end of an ACL denies all other traffic, which does not match the permit clauses.

R1

```
hostname R1
!
interface ethernet0
ip access-group 102 in
!
access-list 112 permit udp any any eq domain
access-list 112 permit udp any eq domain any
access-list 112 permit tcp any any eq domain
access-list 112 permit tcp any eq domain any
```

Permit Routing Updates

When you apply an in-bound ACL on to an interface, ensure that routing updates are not filtered out. Use the relevant ACL from this list to permit routing protocol packets:

Enter this command in order to permit Routing Information Protocol (RIP):

```
access-list 102 permit udp any any eq rip
```

Enter this command in order to permit Interior Gateway Routing Protocol (IGRP):

```
access-list 102 permit igrp any any
```

Enter this command in order to permit Enhanced IGRP (EIGRP):

```
access-list 102 permit eigrp any any
```

Enter this command in order to permit Open Shortest Path First (OSPF):

```
access-list 102 permit ospf any any
```

Enter this command in order to permit Border Gateway Protocol (BGP):

```
access-list 102 permit tcp any any eq 179
access-list 102 permit tcp any eq 179 any
```

Debug Traffic Based on ACL

The use of **debug** commands requires the allocation of system resources like memory and processing power and in extreme situations can cause a heavily-loaded system to stall. Use **debug** commands with care. Use an ACL in order to selectively define the traffic that needs to be examined to reduce the impact of the**debug** command. Such a configuration does not filter any packets.

This configuration turns on the ***debug ip packet*** *command only for packets between the hosts 10.1.1.1 and 172.16.1.1.*

```
R1(config)#access-list 199 permit tcp host 10.1.1.1 host 172.16.1.1
   R1(config)#access-list 199 permit tcp host 172.16.1.1 host 10.1.1.1
   R1(config)#end
   R1#debug ip packet 199 detail
   IP packet debugging is on (detailed) for access list 199
```

MAC Address Filtering

You can filter frames with a particular MAC-layer station source or destination address. Any number of addresses can be configured into the system without a performance penalty. In order to filter by MAC-layer address, use this command in global configuration mode:

```
Router#config terminal
```

```
       bridge irb
       bridge 1 protocol ieee
       bridge 1 route ip
```

Apply the bridge protocol to an interface that you need to filter traffic along with the access list created:

```
Router#int fa0/0
       no ip address
       bridge-group 1 {input-address-list 700 | output-address-list 700}
       exit
```

Create a Bridged Virtual Interface and apply the IP address that is assigned to the Ethernet interface:

```
Router#int bvi1
       ip address
       exit
                                                      !
                                                      !
       access-list 700 deny <mac address> 0000.0000.0000
       access-list 700 permit 0000.0000.0000 ffff.ffff.ffff
```

With this configuration, the router only allows the MAC addresses configured on the access-list 700. With the access list, deny the MAC adddress that can not have access and then permit the rest.